powsybl-core

Release 7.2.0-RC1

Author name not set

Mar 19, 2026

CONTENTS

PowSyBl (Power System Blocks) is an open source library written in Java, dedicated to electrical grid modeling, simulation and visualization, licensed under the Mozilla Public License version 2.0. It is part of LF Energy, an open source foundation focused on the energy sector, hosted within The Linux Foundation.

PowSyBl is used through Python scripts using the library Pypowsybl, but also to build state-of-the-art applications.

Check the *Getting started* and *Configuration* pages to learn how to install and configure PowSyBl.

CHAPTER

ONE

GRID EXCHANGE FORMATS

This section is dedicated to the description of the various grid formats available in PowSyBl. While IIDM is at the core of it, it is possible to import and/or export data in a number of formats that may be dedicated to European data exchanges or network simulation via different tools: check them out below.

Data format	description	import	export
CIM-CGMES	the standard format for European grid data exchange	✓	✓
UCTE-DEF	the legacy format for European grid data exchange	✓	*
IIDM (XIIDM, JIDM, BIIDM)	the internal data model of PowSyBl in a XML / JSON / binary format	✓	✓
IEEE-CDF	a IEEE standard format	✓	
PSS®E	the format for power flow analysis on Siemens PSS®E software	✓	✓
PowerFactory	the format for DIgSILENT PowerFactory software	✓	
Matpower	the format for the free and open-source Matlab toolbox dedicated to power system simulation and optimization	✓	
AMPL	a data separated value format easy to parse with AMPL		✓

* Note that updated export is available, that is, export is possible if the file was imported with the same format. For instance, if you import a UCTE-DEF file in powsybl, you can update some elements and then export it back to UCTE-DEF format, but you cannot export to UCTE-DEF format a file imported from another format.

1.1 CIM-CGMES

1.1.1 Format specification

Current supported versions of CGMES are 2.4.15 and 3.0. To learn more about the standard, read the documents in the Common Grid Model Exchange Standard (CGMES) Library.

1.1.2 Triple store

A triplestore or RDF store is a purpose-built database for the storage and retrieval of triples through semantic queries. A triple is a data entity composed of subject-predicate-object such as “Generator is in France” or in RDF/XML:

```
<rdf:description rdf:about="generator">
    <generator:in>France</generator:in>
</rdf:description>
```

Input CGMES data read from CIM/XML files is stored natively in a purpose-specific database for RDF statements (a Triplestore). There are multiple open-source implementations of Triplestore engines that could be easily plugged in PowSyBl. The only supported Triplestore engine used by PowSyBl is RDF4J. Loading from RDF/XML files to the

Triplestore is highly optimized by these engines. Furthermore, the Triplestore repository can be configured to use an in-memory store, allowing faster access to data.

In-memory Rdf4j

Eclipse RDF4J™ is an open source modular Java framework for working with RDF data. This includes parsing, storing, inferencing and querying of/over such data. It offers an easy-to-use API that can be connected to all leading RDF storage solutions. It allows you to connect with SPARQL endpoints and create applications that leverage the power of Linked Data and Semantic Web.

Its in-memory implementation is the default triplestore engine used by PowSyBl for CIM-CGMES import.

1.1.3 Import

The CIM-CGMES importer reads and converts a CIM-CGMES model to the PowSyBl grid model. The import process is performed in two steps:

- Read input files into a triplestore
- Convert CIM-CGMES data retrieved by SPARQL requests from the created triplestore to PowSyBl grid model

The data in input CIM/XML files uses RDF (Resource Description Framework) syntax. In RDF, data is described making statements about resources using triplet expressions: (subject, predicate, object).

The CIM-CGMES importer supports reading CIM/XML profile files from one of the following data sources:

- a folder containing all uncompressed profile files
- a folder containing the zipped profile files, each one being in a separate zip file
- a zipped file containing all profile files

To describe the conversion from CGMES to PowSyBl, we first introduce several ways of using the importer. We then present some generic considerations about the level of detail of the model (node/breaker or bus/branch), the identity of the equipments and equipment containment in substations and voltage levels. After that, the conversion for every CGMES relevant class is explained. Consistency checks and validations performed during the conversion are mentioned in the corresponding sections.

Import / Update

In CGMES, the power system model is described using data organized into different profiles. This structure allows the equipment in the network and its physical characteristics to be defined in one profile (EQ), while operational data is provided in other profiles (SSH, TP and SV). As a result, it is common to have a single EQ file and multiple SSH files, for example, one for each of the 24 hours of the day.

To handle these cases properly, the importer supports subsequent updates to the initially imported model. You can use the `Network.read` method to perform the initial import, and then apply further updates with `network.update`, where network refers to the originally imported model. Next, we present four use cases to illustrate how both methods can be applied:

Import the entire model in a single step You need to use the `Network.read` method, providing all the CGMES profiles.

Import the complete model in two steps In the first step, only the EQ profile is imported using the `Network.read` method. Then, the operational data is imported using the `network.update` method, providing all or part of the following profiles: SSH, TP, and SV. The `network.update` method cannot be used with EQ profiles.

Import one EQ file and 24 partial SSH files In this case, the 00 SSH file is the only complete file and contains data for all equipment. The remaining SSH files are partial, including only the changes relative to the preceding SSH file. In the first step, we can import the EQ and 00 SSH files using the `Network.read` method. Then, we can perform one update for each of the remaining SSH files using the `network.update` method. Before performing the updates, the

property `iidm.import.cgmes.use-previous-values-during-update` must be set to `true` to fill in any missing data in the partial SSH files using the values previously recorded in the model.

PowSyBl uses variants to record operational data. In this case, we are performing the update on the same variant, so at the end, PowSyBl will only contain the operational data corresponding to the last hour.

Import one EQ file and 24 SSH files, using a different variant for each SSH file In this case, all the SSH files contain data for all the equipment, but we want to record each SSH file on a different variant. To do that, we first import the EQ and 00 SSH files using the `Network.read` method, recording the data in the default initial variant. Then, we perform an update for each of the remaining SSH files using the `network.update` method. However, before each update, we must create a new variant by cloning the initial one and set it as the working variant. At the end, PowSyBl will contain the operational data for all 24 hours, with each hour recorded in a different variant.

Levels of detail: node/breaker and bus/branch

CGMES models defined at node/breaker level of detail will be mapped to PowSyBl node/breaker topology level. CGMES models defined at bus/branch level will be mapped to PowSyBl bus/breaker topology level.

For each equipment in the PowSyBl grid model, it is necessary to specify how it should be connected to the network.

If the model is specified at the bus/breaker level, a `Bus` must be specified for the equipment.

If the voltage level is built at node/breaker level, a `Node` must be specified when adding the equipment to PowSyBl. The conversion will create a different `Node` in PowSyBl for each equipment connection.

Using the `Node` or `Bus` information, PowSyBl creates a `Terminal` that will be used to manage the point of connection of the equipment to the network.

Some equipment, like switches, lines or transformers, have more than one point of connection to the Network.

In PowSyBl, a `Node` can have zero or one terminal. In CGMES, the `ConnectivityNode` objects may have more than one associated terminal. To be able to represent this in PowSyBl, the conversion process will automatically create internal connections between the PowSyBl nodes that represent equipment connections and the nodes created to map `ConnectivityNode` objects.

Identity of model equipments

Almost all the equipments of the PowSyBl grid model require a unique identifier `Id` and may optionally have a human-readable `Name`. Whenever possible, these attributes will be directly copied from original CGMES attributes.

Terminals are used by CGMES and PowSyBl to define the points of connection of the equipment to the network. CGMES terminals have unique identifiers. PowSyBl does not allow terminals to have an associated identifier. Information about original CGMES terminal identifiers is stored in each PowSyBl object using aliases.

Equipment containers: substations and voltage levels

The PowSyBl grid model establishes the substation as a required container of voltage levels and transformers (two- and three-winding transformers and phase shifters). Voltage levels are the required container of the rest of the network equipment, except for the AC and DC transmission lines that establish connections between substations and are directly associated with the network model. All buses at the transformer ends should be kept in the same substation.

The CGMES model does not guarantee these hierarchical constraints, so the first step in the conversion process is to identify all the transformers with ends in different substations and all the breakers and switches with ends in different voltage levels. All the voltage levels connected by breakers or switches should be mapped to a single voltage level in the PowSyBl grid model. The first CGMES voltage level, in alphabetical order, will be the representative voltage level associated with the PowSyBl voltage level. The same criterion is used for substations, and the first CGMES substation will be the representative substation associated with the PowSyBl one. The joined voltage level and substation information is used in almost every step of the mapping between CGMES and PowSyBl models, and it is recorded in the `Context` conversion class, which keeps the data throughout the entire conversion process.

Conversion from CGMES to PowSyBl grid model

The following sections describe in detail how each supported CGMES network object is converted to PowSyBl network model objects.

Substation

For each substation (considering only the representative substation if they are connected by transformers) in the CGMES model a new substation is created in the PowSyBl grid model with the following attributes created as such:

- `Country` It is obtained from the `regionName` property as a first option, from `subRegionName` as second option. Otherwise, is assigned to `null`.
- `GeographicalTags` It is obtained from the `SubRegion` property.

VoltageLevel

As for substations, for each voltage level (considering only the representative voltage level if they are connected by switches) in the CGMES model, a new voltage level is created in the PowSyBl grid model with the following attributes created as such:

- `NominalV` It is copied from the `nominalVoltage` property of the CGMES voltage level.
- `TopologyKind` It will be `NODE_BREAKER` or `BUS_BREAKER` depending on the level of detail of the CGMES grid model.
- `LowVoltageLimit` It is copied from the `lowVoltageLimit` property.
- `HighVoltageLimit` It is copied from the `highVoltageLimit` property.

ConnectivityNode

If the CGMES model is a node/breaker model then `ConnectivityNode` objects are present in the CGMES input files, and for each of them a new `Node` is created in the corresponding PowSyBl voltage level. A `Node` in the PowSyBl model is an integer identifier that is unique by voltage level.

If the import option `iidm.import.cgmes.create-busbar-section-for-every-connectivity-node` is `true` an additional busbar section is also created in the same voltage level. This option is used to debug the conversion and facilitate the comparison of the topology present in the CGMES input files and the topology computed by PowSyBl. The attributes of the busbar section are created as such:

- Identity attributes `Id` and `Name` are copied from the `ConnectivityNode`.
- `Node` The same `Node` assigned to the mapped `ConnectivityNode`.

TopologicalNode

If the CGMES model is defined at bus/branch detail, then `TopologicalNode` objects are used in the conversion, and for each of them a `Bus` is created in the PowSyBl grid model inside the corresponding voltage level container, at the PowSyBl bus/breaker topology level. The created `Bus` has the following attributes:

- Identity attributes `Id` and `Name` are copied from the `TopologicalNode`.
- `V` The voltage of the `TopologicalNode` is copied if it is valid (greater than `0`).
- `Angle` The angle the `TopologicalNode` is copied if the previous voltage is valid.

BusbarSection

Busbar sections can be created in PowSyBl grid model only at node/breaker level.

CGMES Busbar sections are mapped to PowSyBl busbar sections only if CGMES is node/breaker and the import option `iidm.import.cgmes.create-busbar-section-for-every-connectivity-node` is set to `false`. In this case, a `BusbarSection` is created in the PowSyBl grid model for each `BusbarSection` of the CGMES model, with the attributes created as such:

- Identity attributes `Id` and `Name` are copied from the CGMES `BusbarSection`.
- `Node` A new `Node` in the corresponding voltage level.

EnergyConsumer

Every `EnergyConsumer` in the CGMES model creates a new `Load` in PowSyBl. The attributes are created as such:

- `P0`, `Q0` are set from CGMES values taken from `SSH`, `SV`, or `EQ` data depending on which are defined.
- `LoadType` It will be `FICTITIOUS` if the `Id` of the `energyConsumer` contains the pattern `fict`. Otherwise `UNDEFINED`.
- `LoadDetail` Additional information about conform and non-conform loads is added as an extension of the `Load` object (for more details about the *extension*).

The `LoadDetail` extension attributes depend on the `type` property of the `EnergyConsumer`. For a conform load:

- `withFixedActivePower` is always `0`.
- `withFixedReactivePower` is always `0`.
- `withVariableActivePower` is set to the Load `P0`.
- `withVariableReactivePower` is set to the Load `Q0`.

When the type is a non-conform load:

- `withFixedActivePower` is set to the Load `P0`.
- `withFixedReactivePower` is set to the Load `Q0`.
- `withVariableActivePower` is set to `0`.
- `withVariableReactivePower` is set to `0`.

EnergySource

An `EnergySource` is a generic equivalent for an energy supplier, with the injection given using load sign convention.

For each `EnergySource` object in the CGMES model a new PowSyBl `Load` is created, with attributes created as such:

- `P0`, `Q0` set from `SSH` or `SV` values depending on which are defined.
- `LoadType` It will be `FICTITIOUS` if the `Id` of the `energySource` contains the pattern `fict`. Otherwise `UNDEFINED`.

SvInjection

CGMES uses `SvInjection` objects to report mismatches on calculated buses: they record the calculated bus injection minus the sum of the terminal flows. According to the documentation, the values will thus follow generator sign convention: positive sign means injection into the bus. Note that all the reference cases used for development follow load sign convention to report these mismatches, so we have decided to follow this load sign convention as a first approach.

For each SvInjection in the CGMES network model a new PowSyBl Load with attributes created as such:

- P0, Q0 are set from SvInjection.pInjection/qInjection.
- LoadType is always set to FICTITIOUS.
- Fictitious is set to true.

EquivalentInjection

The mapping of an EquivalentInjection depends on its location relative to the boundary area.

If the EquivalentInjection is outside the boundary area, it will be mapped to a PowSyBl Generator.

If the EquivalentInjection is at the boundary area, its regulating voltage data will be mapped to the generation data inside the PowSyBl BoundaryLine created at the boundary point and its values for P, Q will be used to define the BoundaryLine P0, Q0. Please note that the said BoundaryLine can be created from an *ACLineSegment*, a *Switch*, an *EquivalentBranch* or a *PowerTransformer*.

Attributes of the PowSyBl generator or of the PowSyBl boundary line generation are created as such:

- MinP/MaxP are copied from CGMES minP/maxP if defined, otherwise they are set to -Double.MAX_VALUE/Double.MAX_VALUE.
- TargetP/TargetQ are set from SSH or SV values depending on which are defined. CGMES values for p/q are given with load sign convention, so a change in sign is applied when copying them to TargetP/TargetQ.
- TargetV The regulationTarget property is copied if it is not equal to zero. Otherwise, the nominal voltage associated to the connected terminal of the equivalentInjection is assigned. For CGMES Equivalent Injections, the voltage regulation is allowed only at the point of connection.
- VoltageRegulatorOn It is assigned to true if both properties, regulationCapability and regulationStatus are true and the terminal is connected.
- EnergySource is set to OTHER.

ACLineSegment

ACLineSegments' mapping depends on its location relative to the boundary area.

If the ACLineSegment is outside the boundary area, it will be mapped to a PowSyBl *Line*.

If the ACLineSegment is completely inside the boundary area, if the boundaries are not imported, it is ignored. Otherwise, it is mapped to a PowSyBl *Line*.

If the ACLineSegment has one side inside the boundary area and one side outside the boundary area, the importer checks if another branch is connected to the same *TopologicalNode* in the boundary area.

- If there is no other branch connected to this TopologicalNode, it will be mapped to a PowSyBl *BoundaryLine*.
- If there are one or more other branches connected to this TopologicalNode and they all are in the same SubGeographicalRegion, they will all be mapped to PowSyBl *BoundaryLines*.
- If there is exactly one other branch connected to this TopologicalNode in another SubGeographicalRegion, they will both be mapped to PowSyBl *BoundaryLines*, which are part of the same PowSyBl *TieLine*.
- If there are two or more other branches connected to this TopologicalNode in different SubGeographicalRegions:
  - If there are only two branches with their boundary terminal connected and in different SubGeographicalRegion, they will both be mapped to PowSyBl *BoundaryLines*, which are part of the same PowSyBl *TieLine* and all other ACLineSegments will be mapped to PowSyBl *BoundaryLines*.
  - Otherwise, they will all be mapped to PowSyBl *BoundaryLines*.

If the ACLineSegment is mapped to a PowSyBl *Line*:

- R is copied from CGMES r
- X is copied from CGMES x
- G1 is calculated as half of CGMES gch if defined, 0.0 otherwise
- G2 is calculated as half of CGMES gch if defined, 0.0 otherwise
- B1 is calculated as half of CGMES bch
- B2 is calculated as half of CGMES bch

If the ACLineSegment is mapped to an unpaired PowSyBl *BoundaryLine*:

- R is copied from CGMES r
- X is copied from CGMES x
- G is copied from CGMES gch if defined, 0.0 otherwise
- B is copied from CGMES bch
- PairingKey is copied from the name of the TopologicalNode or the ConnectivityNode (respectively in NODE-BREAKER or BUS-BRANCH) inside boundaries
- P0 is copied from CGMES P of the terminal at boundary side
- Q0 is copied from CGMES Q of the terminal at boundary side

If the ACLineSegment is mapped to a paired PowSyBl *BoundaryLine*:

- R is copied from CGMES r
- X is copied from CGMES x
- G1 is 0.0 is the boundary line is on side ONE of the Tie Line. If the boundary line is on side TWO of the Tie Line, it is copied from CGMES gch if defined, 0.0 otherwise.
- G2 is 0.0 is the boundary line is on side TWO of the Tie Line. If the boundary line is on side ONE of the Tie Line, it is copied from CGMES gch if defined, 0.0 otherwise.
- B1 is 0.0 is the boundary line is on side ONE of the Tie Line. If the boundary line is on side TWO of the Tie Line, it is copied from CGMES bch.
- B2 is 0.0 is the boundary line is on side TWO of the Tie Line. If the boundary line is on side ONE of the Tie Line, it is copied from CGMES bch.
- PairingKey is copied from the name of the TopologicalNode or the ConnectivityNode (respectively in NODE-BREAKER or BUS-BRANCH) inside boundaries

EquivalentBranch

Equivalent branches mapping depends on its location relative to the boundary area.

If the EquivalentBranch is outside the boundary area, it will be mapped to a PowSyBl *Line*.

If the EquivalentBranch is completely inside the boundary area, if the boundaries are not imported, it is ignored. Otherwise, it is mapped to a PowSyBl *Line*.

If the EquivalentBranch has one side inside the boundary area and one side outside the boundary area, the importer checks if another branch is connected to the same *TopologicalNode* in the boundary area.

- If there is no other branch connected to this TopologicalNode, it will be mapped to a PowSyBl *BoundaryLine*.

- If there are one or more other branches connected to this TopologicalNode and they all are in the same SubGeographicalRegion, they will all be mapped to PowSyBl *BoundaryLines*.
- If there is exactly one other branch connected to this TopologicalNode in another SubGeographicalRegion, they will both be mapped to PowSyBl *BoundaryLines*, which are part of the same PowSyBl *TieLine*.
- If there are two or more other branches connected to this TopologicalNode in different SubGeographicalRegions:
  - If there are only two branches connected with their boundary terminal connected and in different SubGeographicalRegion, they will both be mapped to PowSyBl *BoundaryLines*, which are part of the same PowSyBl *TieLine* and all other EquivalentBranches will be mapped to PowSyBl *BoundaryLines*.
  - Otherwise, they will all be mapped to PowSyBl *BoundaryLines*.

If the EquivalentBranch is mapped to a PowSyBl *Line*:

- R is copied from CGMES r
- X is copied from CGMES x
- G1 is 0.0
- G2 is 0.0
- B1 is 0.0
- B2 is 0.0

If the EquivalentBranch is mapped to a PowSyBl *BoundaryLine*:

- R is copied from CGMES r
- X is copied from CGMES x
- G is 0.0
- B is 0.0
- PairingKey is copied from the name of the TopologicalNode or the ConnectivityNode (respectively in NODE-BREAKER or BUS-BRANCH) inside boundaries
- P0 is copied from CGMES P of the terminal at boundary side
- Q0 is copied from CGMES Q of the terminal at boundary side

AsynchronousMachine

Asynchronous machines represent rotating machines whose shaft rotates asynchronously with the electrical field. It can be motor or generator; no distinction is made for the conversion of these two types.

An AsynchronousMachine is mapped to a PowSyBl *Load* with attributes created as described below:

- P0, Q0 are set from CGMES values taken from SSH or SVdata depending on which are defined. If there is no defined data, it is 0.0.
- LoadType is FICTITIOUS if the CGMES ID contains "fict". Otherwise, it is UNDEFINED.

SynchronousMachine

Synchronous machines represent rotating machines whose shaft rotates synchronously with the electrical field. It can be motor or generator; no distinction is made for the conversion of these two types.

A SynchronousMachine is mapped to a PowSyBl *Generator* with attributes created as described below:

- MinP is set from GeneratingUnit.minOperatingP on the GeneratingUnit associated with the SynchronousMachine. If invalid, MinP is -Double.MAX_VALUE.
- MaxP is set from GeneratingUnit.maxOperatingP on the GeneratingUnit associated with the SynchronousMachine. If invalid, MaxP is Double.MAX_VALUE.
- ratedS is copied from CGMES ratedS. If it is strictly lower than 0, it is considered undefined.
- EnergySource is defined from the GeneratingUnit class of the GeneratingUnit associated with the SynchronousMachine
  - If it is a HydroGeneratingUnit, EnergySource is HYDRO
  - If it is a NuclearGeneratingUnit, EnergySource is NUCLEAR
  - If it is a ThermalGeneratingUnit, EnergySource is THERMAL
  - If it is a WindGeneratingUnit, EnergySource is WIND. Additionally, the WindGeneratingUnit.windGenUnitType value (onshore or offshore) is stored as an iIDM property CGMES.windGenUnitType of the generator.
  - If it is a SolarGeneratingUnit, EnergySource is SOLAR
  - Else, EnergySource is OTHER
- TargetP/TargetQ are set from SSH or SV values depending on which are defined. CGMES values for p/q are given with load sign convention, so a change in sign is applied when copying them to TargetP/TargetQ. If undefined, TargetP is set from CGMES GeneratingUnit.initialP from the GeneratingUnit associated to the SynchronousMachine and TargetQ is set to 0.
- isCondenser is defined from the SynchronousMachine.type. If it contains condenser (condenser, generatorOrCondenser, motorOrCondenser, generatorOrMotorOrCondenser), then the flag is set to true. Otherwise, it is set to false.

TODO reactive limits

TODO regulation

TODO normalPF

EquivalentShunt

An EquivalentShunt is mapped to a PowSyBl linear *ShuntCompensator*. A linear shunt compensator has banks or sections with equal admittance values. Its attributes are created as described below:

- SectionCount is 1 if the EquivalentShunt CGMES Terminal is connected, else it is 0.
- BPerSection is copied from CGMES b
- MaximumSectionCount is set to 1

ExternalNetworkInjection

External network injections are injections representing the flows from an entire external network.

An ExternalNetworkinjection is mapped to a PowSyBl *Generator* with attributes created as described below:

- MinP is copied from CGMES minP
- MaxP is copied from CGMES maxP
- TargetP/TargetQ are set from SSH or SV values depending on which are defined. CGMES values for p/q are given with load sign convention, so a change in sign is applied when copying them to TargetP/TargetQ. If undefined, they are set to 0.

- EnergySource is set as OTHER

TODO reactive limits

TODO regulation

LinearShuntCompensator

Linear shunt compensators represent shunt compensators with banks or sections with equal admittance values.

A LinearShuntCompensator is mapped to a PowSyBl *ShuntCompensator* with SectionCount copied from CGMES SSH sections if present. If not, it is copied from CGMES SvShuntCompensatorSections.sections or normalSections. The SolvedSectionCount is copied from SvShuntCompensatorSections.sections if the SV is imported, and left to null otherwise. The created PowSyBl shunt compensator is linear, and its attributes are defined as described below:

- BPerSection is copied from CGMES bPerSection if defined. Else, it is Float.MIN_VALUE.
- GPerSection is copied from CGMES gPerSection if defined. Else, it is left undefined.
- MaximumSectionCount is copied from CGMES maximumSections.

TODO regulation

NonlinearShuntCompensator

Non-linear shunt compensators represent shunt compensators with banks or section admittance values that differ.

A NonlinearShuntCompensator is mapped to a PowSyBl *ShuntCompensator* with SectionCount copied from CGMES SSH sections if present. If not, it is copied from CGMES SvShuntCompensatorSections.sections or normalSections. The SolvedSectionCount is copied from SvShuntCompensatorSections.sections if the SV is imported, and left to null otherwise. The created PowSyBl shunt compensator is non-linear and has as many Sections as there are NonlinearShuntCompensatorPoint associated with the NonlinearShuntCompensator it is mapped to.

Sections are created from the lowest CGMES sectionNumber to the highest and each section has its attributes created as described below:

- B is calculated as the sum of all CGMES b of NonlinearShuntCompensatorPoints with sectionNumber lower or equal to its sectionNumber
- G is calculated as the sum of all CGMES g of NonlinearShuntCompensatorPoints with sectionNumber lower or equal to its sectionNumber

TODO regulation

OperationalLimits

OperationalLimits model a specification of limits associated with equipments.

OperationalLimitSet

A CGMES OperationalLimitSet is a set of OperationalLimit associated with equipment or terminal. It is mapped to a PowSyBl *OperationalLimitsGroup*.

Just like CGMES allows to attach multiple OperationalLimitSet on the same equipment or terminal, PowSyBl stores a collection of OperationalLimitsGroup for every *Line* side, *BoundaryLine* and *ThreeWindingTransformer.Leg*.

The same way a CGMES OperationalLimitSet may contain OperationalLimit of different subclasses, a PowSyBl OperationalLimitsGroup may have multipe non-null LoadingLimits.

If there is only one OperationalLimitsGroup on an end, it automatically gets to be selected (active). However, if there is multiple groups, none is selected: the user has to choose which set is active.

OperationalLimit

A CGMES OperationalLimit is an abstract class that represent different kinds of limits: current, active power or apparent power. The collection of the same subclass of CGMES OperationalLimit in the set is mapped to a PowSyBl *LoadingLimits* as follows:

- The collection of CGMES CurrentLimit in the OperationalLimitSet is mapped to the currentLimits attribute of the PowSyBl OperationalLimitsGroup corresponding to the set.
- The collection of CGMES ActivePowerLimit in the OperationalLimitSet is mapped to the activePowerLimits attribute of the PowSyBl OperationalLimitsGroup corresponding to the set.
- The collection of CGMES ApparentPowerLimit in the OperationalLimitSet is mapped to the apparentPowerLimits attribute of the PowSyBl OperationalLimitsGroup corresponding to the set.

A particular CGMES OperationalLimit is mapped differently depending on its associated CGMES OperationalLimitType:

- A permanent limit (OperationalLimitType.limitType is LimitTypeKind.patl) is mapped as follows:
  - PowSyBl LoadingLimits.permanentLimit is copied from CGMES OperationalLimit.value
- A temporary limit (OperationalLimitType.limitType is LimitTypeKind.tatl) is mapped as follows:
  - A new entry is created in PowSyBl LoadingLimits.temporaryLimits
  - name is copied from OperationalLimit.name
  - value is copied from OperationalLimit.value
  - acceptableDuration is copied from OperationalType.acceptableDuration

PowerTransformer

Power transformers represent electrical devices consisting of two or more coupled windings, each represented by a PowerTransformerEnd. PowSyBl only supports PowerTransformers with two or three windings.

PowerTransformer with two PowerTransformerEnds

If a PowerTransformer has two PowerTransformerEnds, both outside the boundary area, it is mapped to a PowSyBl *TwoWindingsTransformer*. Please note that in this case, if PowerTransformerEnds are in different substations, the substations are merged into one.

If a PowerTransformer has two PowerTransformerEnds, both completely inside the boundary area, and if the boundary area is not imported, the PowerTransformer is ignored. Otherwise, it is mapped to a PowSyBl *TwoWindingsTransformer*.

If the PowerTransformer has one PowerTransformerEnd inside the boundary area and the other outside the boundary area, the importer checks if another branch is connected to the same *TopologicalNode* in the boundary area.

- If there is no other connected to this TopologicalNode, it is mapped to a PowSyBl *BoundaryLine*.
- If there is one or more other branches connected to this TopologicalNode and they are all in the same SubGeographicalRegion, they will all be mapped to PowSyBl *BoundaryLines*.
- If there is exactly one other branch connected to this TopologicalNode in another SubGeographicalRegion, they will both be mapped to PowSyBl *BoundaryLines*, which are part of the same PowSyBl *TieLine*.
- If there are two or more other branches connected to this TopologicalNode in different SubGeographicalRegions:

  - If there are only two branches with their boundary terminal connected and in different SubGeographicalRegion, they will both be mapped to PowSyBl *BoundaryLines*, which are part of the same PowSyBl *TieLine* and all other EquivalentBranches will be mapped to PowSyBl *BoundaryLines*.
  - Otherwise, they will all be mapped to PowSyBl *BoundaryLines*.

In every case, a PowerTransformer with two PowerTransformerEnds is mapped to an intermediary model that corresponds to a PowSyBl *TwoWindingsTransformer*. For more information about this conversion, please look at the classes InterpretedT2xModel and ConvertedT2xModel.

If the PowerTransformer is finally mapped to a PowSyBl *BoundaryLine*, its structural attributes (R, X, G and B) are calculated from the intermediary model's attributes, and the ratio from its ratio tap changer and/or its phase tap changer. P0 and Q0 are set from CGMES P and Q values at boundary side; PairingKey is copied from the name of the TopologicalNode or the ConnectivityNode (respectively in NODE-BREAKER or BUS-BRANCH) inside boundaries.

If the PowerTransformer is finally mapped to a PowSyBl *BoundaryLine*, its attributes are calculated using a standard π model with distributed parameters.

PowerTransformer with three PowerTransformerEnds

A PowerTransformer with three PowerTransformerEnds is mapped to a PowSyBl *ThreeWindingsTransformer*. Please note that in this case, if PowerTransformerEnds are in different substations, the substations are merged into one.

For more information about this conversion, please look at the classes InterpretedT3xModel and ConvertedT3xModel.

SeriesCompensator

Series compensators represent series capacitors or reactors or AC transmission lines without charging susceptance.

If a SeriesCompensator has both its ends inside the same voltage level, it is mapped to a PowSyBl *Switch*. In this case, all its CGMES electrical attributes are ignored. It is considered as closed, fictitious and, if it is in a node-breaker voltage level, retained. Its SwitchKind is BREAKER.

If a SeriesCompensator has its ends inside different voltage levels, it is mapped to a PowSyBl *Line* with attributes as described below:

- R is copied from CGMES r
- X is copied from CGMES x
- G1, G2, B1 and B2 are set to 0

StaticVarCompensator

Static VAR compensators represent a facility for providing variable and controllable shunt reactive power.

A StaticVarCompensator is mapped to a PowSyBl *StaticVarCompensator* with attributes as described below:

- Bmin is calculated from CGMES inductiveRating: if it is defined and not equals to 0, Bmin is 1 / inductiveRating. Else, it is -Double.MAX_VALUE.
- Bmax is calculated from CGMES capacitiveRating: if it defined and not equals to 0, Bmax is 1 / capacitiveRating. Else, it is Double.MAX_VALUE.

A PowSyBl *VoltagePerReactivePowerControl* extension is also created from the CGMES StaticVarCompensator and linked to the PowSyBl StaticVarCompensator with its slope attribute copied from CGMES slope if the latter is 0 or positive.

TODO regulation

Switch (Switch, Breaker, Disconnector, LoadBreakSwitch, ProtectedSwitch, GroundDisconnector, Jumper)

Switches, breakers, disconnectors, load break switches, protected switches, jumpers and ground disconnectors are all imported in the same manner. For convenience purposes, we will now use `Switch` as a say but keep in mind that this section is valid for all these CGMES classes.

If the `Switch` has its ends both inside the same voltage level, it is mapped to a PowSyBl *Switch* with attributes as described below:

- `SwitchKind` is defined depending on the CGMES class
  - If it is a CGMES `Breaker`, `Switch` or `ProtectedSwitch`, it is `BREAKER`
  - If it is a CGMES `Disconnector`, `GroundDisconnector` or `Jumper` it is `DISCONNECTOR`
  - If it is a CGMES `LoadBreakSwitch`, it is `LOAD_BREAK_SWITCH`
- `Retained` is copied from CGMES `retained` if defined in node-breaker. Else, it is `false`.
- `Open` is copied from CGMES SSH `open` if defined. Else, it is copied from CGMES `normalOpen`. If neither are defined, it is `false`.

If the CGMES `Switch` has its ends in different voltage levels inside the same IGM, it is mapped to a *Switch* but the voltage levels, and potentially the substations, that contain its ends are merged: they are mapped to only one voltage level and/or substation. The created PowSyBl `Switch` has its attributes defined as described above.

If the `Switch` has one side inside the boundary area and the other side outside the boundary area, the importer checks if another branch is connected to the same CGMES *TopologicalNode* in the boundary area.

- If there is no other branch connected to this `TopologicalNode`, it will be mapped to a PowSyBl *BoundaryLine*.
- If there are one or more other branches connected to this `TopologicalNode` and they all are in the same `SubGeographicalRegion`, they will all be mapped to PowSyBl *BoundaryLines*.
- If there is exactly one other branch connected to this `TopologicalNode` in another `SubGeographicalRegion`, they will both be mapped to PowSyBl *BoundaryLines*, which are part of the same PowSyBl *TieLine*.
- If there are two or more other branches connected to this `TopologicalNode` in different `SubGeographicalRegions`:
  - If there are only two branches with their boundary terminal connected and in different `SubGeographicalRegion`, they will both mapped to PowSyBl *BoundaryLines*, which are part of the same PowSyBl *TieLine* and all other `EquivalentBranches` will be mapped to PowSyBl *BoundaryLines*.
  - Otherwise, they will all be mapped to PowSyBl *BoundaryLines*.

If the CGMES `Switch` is mapped to a PowSyBl *BoundaryLine*, its attributes are as described below:

- `R`, `X`, `G`, `B` are `0.0`;
- `PairingKey` is copied from the name of the `TopologicalNode` or the `ConnectivityNode` (respectively in `NODE-BREAKER` or `BUS-BRANCH`) inside boundaries;
- `P0` is copied from CGMES `P` of the terminal at boundary side;
- `Q0` is copied from CGMES `Q` of the terminal at boundary side

Control areas

CGMES control areas (objects of class `ControlArea`) are mapped directly to PowSyBl objects of type `Area`.

The control area CGMES `type` is copied as a string in the `areaType` attribute of the PowSyBl `Area`. The CGMES `netInterchange` is copied to the PowSyBl `interchangeTarget`. If CGMES `pTolerance` is defined, its value is copied to a new property named `pTolerance`. Finally, if an attribute `entsoe:IdentifiedObject.energyIdentCodeEic` is found for the CGMES control area, it is added as an alias with `aliasType == "energyIdentCodeEic"`.

The CGMES control area tie flows (objects of class `TieFlow`) are mapped to PowSyBl `Area` boundary items. Boundary items can be terminals (if the corresponding CGMES point can be mapped to a PowSyBl `Terminal`) or boundaries, when the corresponding CGMES point is the boundary side of a boundary line in PowSyBl.

Reduced DC model

The reduced DC model allows the support of the following simple DC configurations:

- Monopole with ground return.
- Monopole with metallic return.
- Bipole with dedicated metallic return (DMR).
- Bipole without DMR.

In the above point-to-point configurations, each `DCConverterUnit` can contain 1 CGMES `ACDCConverter` (1* 12-pulse bridge) or 2 CGMES `ACDCConverter` (2* 6-pulses bridges).

Other configurations such as back-to-back, multi-terminal or hybrid aren't supported with the reduced DC model. If one of these complex DC configurations is to be imported, it is required to set the optional parameter `iidm.import.cgmes.use-detailed-dc-model` to `true`.

Each valid DC configuration is mapped to PowSyBl as follows:

- 1 CGMES `ACDCConverter` is always mapped to 1 PowSyBl `HvdcConverterStation`:
  - CGMES subclass `CsConverter` is mapped to PowSyBl subclass `LccConverterStation`.
  - CGMES subclass `VsConverter` is mapped to PowSyBl subclass `VscConverterStation`.
- 1 or 2 CGMES `DCLineSegment` are mapped to 1 or 2 `HvdcLine`.
  - See table below that shows when dc lines are merged or split.

Configuration	Number of converters(CGMES or PowSyBl)	Number ofCGMES DCLineSegment	Number ofPowSyBl HvdcLine
Monopole, metallic return	1	2	1
Monopole, ground return,1 bridge per unit	1	1	1
Monopole, ground return,2 bridges per unit	2	1	2
Bipole	2	2 (*)	2
Bipole, 2 bridges per unit	4	2 (*)	4

- (*) The DMR is never considered for the mapping since no flow runs through it.

The merging or splitting of dc lines is necessary to always end up with triplets: `HvdcConverterStation` (side 1) + `HvdcLine` + `HvdcConverterStation` (side 2) in PowSyBl.

The detail mapping of the classes is detailed below.

DCLineSegment

The mapping of CGMES `DCLineSegment` to PowSyBl `HvdcLine` isn't done in isolation, but always in association with the `ACDCConverter` on each side it is connected to.

The PowSyBl `R` value is mapped as follows:

- If the CGMES to PowSyBl cardinality is 1 to 1, the PowSyBl `R` value is copied from CGMES EQ `r`.
- If the CGMES to PowSyBl cardinality is 2 to 1, the PowSyBl `R` value is the sum of the CGMES EQ `r`: $R = r_1 + r_2$
- If the CGMES to PowSyBl cardinality is 1 to 2, each PowSyBl `R` is equal to half the CGMES EQ `r`: $R_1 = R_2 = \frac{r}{2}$

The PowSyBl `NominalV` is copied from side 1 converter CGMES EQ `ACDCConverter.ratedUdc`.

The PowSyBl `ConvertersMode` is determined from the 2 neighbouring `ACDCConverter`:

- If one of the CGMES SSH `ACDCConverter.targetPpcc` is set and has a positive value, or in the case of LCC lines if one of the CGMES SSH `CsConverter.operatingMode` is set to `CsOperatingModeKind.rectifier`, then this converter is the rectifier, and the one on the other side is the inverter.
- If one of the CGMES SSH `ACDCConverter.targetPpcc` is set and has a negative value, or in the case of LCC lines if one of the CGMES SSH `CsConverter.operatingMode` is set to `CsOperatingModeKind.inverter`, then this converter is the inverter, and the one on the other side is the rectifier.
- Based on above results and on which side each converter is located, the PowSyBl `ConvertersMode` is then computed to `SIDE_1_RECTIFIER_SIDE_2_INVERTER` or `SIDE_1_INVERTER_SIDE_2_RECTIFIER`. In case the information couldn't be retrieved, for example in the case of an EQ only import, the default mode is set to `SIDE_1_RECTIFIER_SIDE_2_INVERTER`.

Similarly, the PowSyBl `ActivePowerSetpoint` is determined from the 2 neighbouring `ACDCConverter`:

- If the CGMES SSH rectifier's `ACDCConverter` defines a `ACDCConverter.targetPpcc`, then the PowSyBl `ActivePowerSetpoint` is copied from it.
- If the CGMES SSH inverter's `ACDCConverter` defines a `ACDCConverter.targetPpcc`, this value is brought back to the rectifier's side, by adding losses all along the line. See `ACDCConverter` mapping for the calculation detail.
- In case the information couldn't be retrieved, for example in the case of an EQ only import, the default value is `0.0`.

The PowSyBl `MaxP` is set to 120% of `ActivePowerSetpoint`.

ACDCConverter

The mapping of CGMES `ACDCConverter` to PowSyBl `HvdcConverterStation` isn't done in isolation, but always in association with the `DCLineSegment` it is connected to and the `ACDCConverter` on the other side of the line.

The PowSyBl `LossFactor` is computed from CGMES SSH `ACDCConverter.targetPpcc` values and CGMES SV `poleLossP` values. It is sufficient for one of the converter to define a `targetPpcc` to be able to calculate the AC and DC active powers all along the line:

- If `ACDCConverter.targetPpcc` is defined by CGMES SSH rectifier's `ACDCConverter`, then:
  - $P_{AC,rectifier} = targetPpcc$
  - $P_{DC,rectifier} = P_{AC,rectifier} - poleLossP_{rectifier}$
  - $P_{DC,inverter} = -1 \times (P_{DC,rectifier} - resistiveLosses)$, where $resistiveLosses = R * idc$ and $idc = \frac{P_{DC,rectifier}}{NominalV}$
  - $P_{AC,inverter} = P_{DC,inverter} + poleLossP_{inverter}$

- If `ACDCConverter.targetPpcc` is defined by CGMES SSH inverter's `ACDCConverter`, then:
  - $P_{AC,inverter} = targetPpcc$
  - $P_{DC,inverter} = P_{AC,inverter} - poleLossP_{inverter}$
  - $P_{DC,rectifier} = abs(P_{DC,inverter}) + resistiveLosses$, where $resistiveLosses = R * idc$ and $idc = \frac{NominalV - \sqrt{NominalV^2 - 4 \times R \times abs(P_{DC,inverter})}}{2 \times R}$
  - $P_{AC,rectifier} = P_{DC,rectifier} + poleLossP_{rectifier}$
- Once these active power have been calculated, the PowSyBl `LossFactor` is computed as follows:
  - $LossFactor_{rectifier} = \frac{poleLossP_{rectifier}}{P_{AC,rectifier}}$
  - $LossFactor_{inverter} = \frac{poleLossP_{inverter}}{abs(P_{DC,inverter})}$
  - In case the calculations can't be evaluated, for example in the case of an EQ only import, the default value is `0.0`.

The PowSyBl `LccConverterStation` `PowerFactor` is calculated from the CGMES SSH `ACDCConverter.p` and `ACDCConverter.q` values:

- $PowerFactor = \frac{p}{\sqrt{p+q}}$
- In case the calculations can't be evaluated, for example in the case of an EQ only import, the default value is `0.8`.

Detailed DC model

In order to import CGMES DC objects into the IIDM detailed DC model, it is required to set the optional parameter `iidm.import.cgmes.use-detailed-dc-model` to `true`.

The following CGMES classes are read and imported: `DCNode`, `DCTopologicalNode`, `DCLineSegment`, `DCSwitch`, `DCBreaker`, `DCDisconnector`, `DCGround`, `CsConverter`, `VsConverter`.

DCNode, DCTopologicalNode

If the CGMES model is a node/breaker one, then CGMES `DCNode` are imported into PowSyBl *DC Node*, and CGMES `DCTopologicalNode` are discarded. If the CGMES model is a bus/branch one, then CGMES `DCTopologicalNode` are imported into PowSyBl `DcNode`. Attribute mapping is:

- `NominalV` is copied from EQ `ratedUdc` of a CGMES `ACDCConverter` which is located in the same `DCConverterUnit` as the `DCNode`.

DCLineSegment

CGMES `DCLineSegment` are imported into PowSyBl *DC Line*, with attribute:

- `R` is copied from EQ `resistance`.

DCSwitch, DCBreaker, DCDisconnector

All CGMES `DCSwitch` are imported into PowSyBl *DC Switch*, with attributes described as follow:

- `Kind` is defined depending on the CGMES class: it is `BREAKER` for CGMES `DCBreaker`, and it is `DISCONNECTOR` for CGMES `DCSwitch` and `DCDisconnector`.
- `Open` is defined depending on the associated CGMES `DCTerminal`: if both have SSH `connected` set to `true`, then it is `false`, otherwise it is `true`.

DCGround

CGMES DCGround are imported into PowSyBl *DC Ground*, with attribute:

- R is copied from EQ r.

ACDCConverter (CsConverter, VsConverter)

CGMES CsConverter are imported into PowSyBl *Line Commutated Converter*, and VsConverter into *Voltage Source Converter*. Common *AC/DC Converter* attributes are mapped as follows:

- IdleLoss is copied from EQ idleLoss.
- SwitchingLoss is copied from EQ switchingLoss.
- ResistiveLoss is copied from EQ resistiveLoss.
- PccTerminal is copied from EQ PccTerminal.
- ControlMode is P_PCC if SSH pPccControl is activePower (CSC) or pPcc (VSC), else it is V_DC if pPccControl is dcVoltage (CSC) or udc (VSC).
- TargetP is copied from SSH targetPpcc.
- TargetVdc is copied from SSH targetUdc.

Specific CsConverter attributes are mapped as follows:

- ReactiveModel is always set to FIXED_POWER_FACTOR.
- PowerFactor is calculated from SSH p and q as $\left|\frac{p}{\sqrt{p^2+q^2}}\right|$. If p and q are equal to 0, the power factor is calculated with $Q = 0.5P$, which gives the value 0.89443.

Specific VsConverter attributes are mapped as follows:

- VoltageRegulatorOn is set to true if SSH qPccControl is set to voltagePcc, else it is false.
- VoltageSetpoint is copied from SSH targetUpcc.
- ReactivePowerSetpoint is copied from SSH targetQpcc.

Extensions

The CIM-CGMES format contains more information than what the iidm grid model needs for calculation. The additional data that are needed to export a network in CIM-CGMES format are stored in several extensions.

CGMES boundary line boundary node

This extension is used to add some CIM-CGMES characteristics to boundary lines.

Attribute	Type	Unit	Required	Default value	Description
hvdc status	boolean	-	no	false	Indicates if the boundary line is associated with a DC Xnode or not
Line Energy Identification Code (EIC)	String	-	no	-	The EIC of the boundary line if it exists

This extension is provided by the com.powsybl:powsybl-cgmes-extensions module.

CGMES line boundary node

This extension is used to add some CIM-CGMES characteristics to tie lines.

Attribute	Type	Unit	Required	Default value	Description
hvdc status	boolean	-	no	false	Indicates if the boundary line is associated with a DC Xnode or not
Line Energy Identification Code (EIC)	String	-	no	-	The EIC of the boundary line EIC if it exists

This extension is provided by the `com.powsybl:powsybl-cgmes-extensions` module.

CGMES Tap Changers

TODO

The `TapPosition` of the IIDM `TapChanger` is copied from the CGMES SSH `step` if present. If not, it is copied from CGMES `SVtapStep` or `normalStep` from EQ. The `SolvedTapPosition` is copied from `SVtapStep` if the SV is imported, and left to `null` otherwise.

CGMES metadata models

This extension is attached to a Network and is used to store the metadata information about the imported CGMES dataset. The extension consists of a collection of `CgmesMetadataModel` objects, one per CGMES instance file or *subset*, that hold the metadata information present in the `FullModel` tag in the header of the CGMES instance file.

Attribute	Type	Unit	Required	Default value	Description
id	String	-	no	-	Unique identifier of the model
subset	CgmesSubset	-	yes	-	The imported instance file(EQUIPMENT, TOPOLOGY, STEADY_STATE_HYPOTHESIS, STATE_VARIABLES)
description	String	-	no	<subset> + Model	The description of the model and explanation of the purpose
version	int	-	no	1	The version number of the model
modelingAuthoritySet	String	-	yes	-	The organisation role which is the source of the model
profiles	Set<String>	-	no	-	The profiles included in this subset
dependentOn	Set<String>	-	no	-	References to other models this model depends on
supersedes	Set<String>	-	no	-	References to other models this model supsersedes

Example of code to read the extension and retrieve the modeling authority set assuming the network has been imported from a CGMES datasource:

```
CgmesMetadataModels models = network.getExtension(CgmesMetadataModels.class);
CgmesMetadataModel eqModel = models.getModelForSubset(CgmesSubset.EQUIPMENT).
↪orElseThrow();
String modelingAuthoritySet = eqModel.getModelingAuthoritySet();
```

This extension is provided by the `com.powsybl:powsybl-cgmes-extensions` module.

CIM characteristics

This extension is attached to a network and is used to store characteristics about the imported CGMES dataset.

Attribute	Type	Unit	Required	Default value	Description
cimVersion	int	-	yes	-	Version number of imported dataset: 16 for CIM16/CGMES 2.4.15, 100 for CIM100/CGMES 3.0
topologyKind	CgmesTopologyKind	-	yes	-	Topology kind: NODE_BREAKER or BUS_BRANCH

Please note that the topologyKind attribute reflects how the dataset has been considered: if the `iidm.import.cgmes.import-node-breaker-as-bus-breaker` import parameter has been set to `true`, topologyKind will be `BUS_BRANCH`, even if the network has `NODE_BREAKER` model details.

Example of code to read the extension and retrieve the topology kind assuming the network has been imported from a CGMES datasource:

```
CimCharacteristics cimCharacteristics = network.getExtension(CimCharacteristics.class);
CgmesTopologyKind topologyKind = cimCharacteristics.getTopologyKind();
```

This extension is provided by the `com.powsybl:powsybl-cgmes-extensions` module.

Base voltage mapping

This extension is attached to a network and is used to store information about BaseVoltage of the imported CGMES dataset. The extension consists of a collection of `BaseVoltageSource` objects, indexed by nominal voltage, with the following attributes:

Attribute	Type	Unit	Required	Default value	Description
id	String	-	yes	-	The rdf:ID of the CGMES BaseVoltage
nominalV	double	kV	yes	-	The CGMES BaseVoltage's nominal voltage
source	Source	-	yes	-	The kind of grid model containing the BaseVoltage definition (IGM or BOUNDARY)

Example of code to read the extension and retrieve the source of the 400kV base voltage assuming the network has been imported from a CGMES datasource:

```
BaseVoltageMapping bvMapping = network.getExtension(BaseVoltageMapping.class);
BaseVoltageMapping.BaseVoltageSource bvSource = bvMapping.getBaseVoltages().get(400.0);
Source source = bvSource.getSource();
```

This extension is provided by the `com.powsybl:powsybl-cgmes-extensions` module.

CGMES model

This extension provides access to the PowSyBl CGMES Network Model implemented with a triplestore.

Note that in order for this extension to be present, the corresponding property `iidm.import.cgmes.store-cgmes-model-as-network-extension` must be set to true.

Exemple of code to read the extension and retrieve the substations in the CGMES input files:

```
CgmesModelExtension cgmesModel = network.getExtension(CgmesModelExtension.class);
PropertyBags substationBags = cgmesModel.getCgmesModel().substations();
```

This extension is provided by the `com.powsybl:powsybl-cgmes-conversion` module.

CGMES conversion context

This extension is useful for external validation of the mapping made between CGMES and IIDM.

Note that in order for this extension to be present, the corresponding property `iidm.import.cgmes.store-cgmes-conversion-context-as-network-extension` must be set to true.

Exemple of code to read the extension and retrieve the naming strategy:

```
CgmesConversionContextExtension cgmesConversionContext = network.
↪getExtension(CgmesConversionContextExtension.class);
NamingStrategy namingStrategy = cgmesConversionContext.getContext().namingStrategy();
```

This extension is provided by the `com.powsybl:powsybl-cgmes-conversion` module.

Options

These properties can be defined in the configuration file in the *import-export-parameters-default-value* module.

iidm.import.cgmes.boundary-location Optional property that defines the directory path where the CGMES importer can find the boundary files (`EQBD` and `TPBD` profiles) if they are not present in the imported zip file. By default, its value is `<ITOOLS_CONFIG_DIR>/CGMES/boundary`. This property can also be used at CGMES export if the network was not imported from a CGMES to indicate the boundary files that should be used for reference.

iidm.import.cgmes.convert-boundary Optional property that defines if the equipment located inside the boundary is imported as part of the network. Used for debugging purposes. `false` by default.

iidm.import.cgmes.convert-sv-injections Optional property that defines if `SvInjection` objects are converted to IIDM loads. `true` by default.

iidm.import.cgmes.create-active-power-control-extension Optional property that defines if active power control extensions are created for the converted generators. `true` by default. If `true`, the extension will be created for the CGMES `SynchronousMachines` with the attribute `normalPF` defined. For these generators, the `normalPF` value will be saved as the `participationFactor` and the flag `participate` set to `true`.

iidm.import.cgmes.create-busbar-section-for-every-connectivity-node Optional property that defines if the CGMES importer creates an *IIDM Busbar Section* for each CGMES connectivity node. Used for debugging purposes. `false` by default.

iidm.import.cgmes.create-fictitious-switches-for-disconnected-terminals-mode Optional property that defines if fictitious switches are created when terminals are disconnected in CGMES node-breaker networks. Three modes are available:

- `ALWAYS`: fictitious switches are created at every disconnected terminal.
- `ALWAYS_EXCEPT_SWITCHES`: fictitious switches are created at every disconnected terminal that is not a terminal of a switch.
- `NEVER`: no fictitious switches are created at disconnected terminals.

The default value is `ALWAYS`.

iidm.import.cgmes.decode-escaped-identifiers Optional property that defines if identifiers containing escaped characters are decoded when CGMES files are read. `true` by default.

iidm.import.cgmes.ensure-id-alias-unicity Optional property that defines if IDs' and aliases' unicity is ensured during CGMES import. If it is set to `true`, identical CGMES IDs will be modified to be unique. If it is set to `false`, identical CGMES IDs will throw an exception. `false` by default.

iidm.import.cgmes.import-control-areas Optional property that defines if control areas must be imported or not. `true` by default.

iidm.import.cgmes.naming-strategy Optional property that defines which naming strategy is used to transform CGMES identifiers to IIDM identifiers. Currently, all naming strategies assign CGMES Ids directly to IIDM Ids during import, without any transformation. The default value is `identity`. You can also define a custom naming strategy by implementing the `NamingStrategy` interface on your own project and declare a `NamingStrategyProvider` that can be automatically discovered. Then in this parameter, you can specify the name of the provider.

iidm.import.cgmes.post-processors Optional property that defines all the CGMES post-processors which will be activated after import. By default, it is an empty list. One implementation of such a post-processor is available in PowSyBl in the powsybl-diagram repository, named *CgmesDLImportPostProcessor*.

iidm.import.cgmes.powsybl-triplestore Optional property that defines which Triplestore implementation is used. Currently, PowSyBl only supports RDF4J. `rdf4j` by default.

iidm.import.cgmes.source-for-iidm-id Optional property that defines if IIDM IDs must be obtained from the CGMES `mRID` (master resource identifier) or the CGMES `rdfID` (Resource Description Framework identifier). The default value is `mRID`.

iidm.import.cgmes.store-cgmes-model-as-network-extension Optional property that defines if the whole CGMES model is stored in the imported IIDM network as an *extension* of the IIDM output network. The default value is `false`.

The CGMES model triplestore is not closed after CGMES import when this option is enabled. To reclaim memory, manually close the triplestore via the extension.

iidm.import.cgmes.store-cgmes-conversion-context-as-network-extension Optional property that defines if the CGMES conversion context is stored as an *extension* of the IIDM output network. It is useful for external validation of the mapping made between CGMES and IIDM. Its default value is `false`.

The CGMES model triplestore is not closed after CGMES import when this option is enabled. To reclaim memory, manually close the triplestore via the extension.

iidm.import.cgmes.use-detailed-dc-model Optional property that defines which IIDM DC model should be populated at import. Set to `true` to import DC objects into the detailed DC model, `false` to import into the reduced DC model. The default value is `false`.

iidm.import.cgmes.import-node-breaker-as-bus-breaker Optional property that forces CGMES model to be in topology bus/breaker in IIDM. This is a key feature when some models do not have all the breakers to connect and disconnect equipments in IIDM. In bus/breaker topology, connect and disconnect equipment only rely on terminal statuses and not on breakers. Its default value is `false`.

iidm.import.cgmes.disconnect-boundary-line-if-boundary-side-is-disconnected
Optional property used at CGMES import that disconnects the IIDM boundary line if in the CGMES model the line is open at the boundary side. As IIDM does not have any equivalence for that, this is an approximation. Its default value is `false`.

iidm.import.cgmes.missing-permanent-limit-percentage Optional property used when in operational limits, temporary limits are present and the permanent limit is missing as it is forbidden in IIDM. The missing permanent limit is equal to a percentage of the lowest temporary limit, with the percentage defined by the value of this property if present, `100` by default.

iidm.import.cgmes.cgm-with-subnetworks Optional property to define if subnetworks must be added to the network when importing a Common Grid Model (CGM). Each subnetwork will model an Individual Grid Model (IGM). By

default `true`: subnetworks are added, and the merging is done at IIDM level, with a main IIDM network representing the CGM and containing a set of subnetworks, one for each IGM. If the value is set to `false` all the CGMES data will be flattened in a single network and information about the ownership of each equipment will be lost.

iidm.import.cgmes.cgm-with-subnetworks-defined-by If `iidm.import.cgmes.cgm-with-subnetworks` is set to `true`, use this property to specify how the set of input files should be split by IGM: based on their filenames (use the value `FILENAME`) or by its modeling authority, read from the header (use the value `MODELING_AUTHORITY`). Its default value is `MODELING_AUTHORITY`.

iidm.import.cgmes.create-fictitious-voltage-level-for-every-node Optional property that defines the fictitious voltage levels created by line container. If it is set to `true`, a fictitious voltage level is created for each connectivity node inside the line container. If it is set to `false`, only one fictitious voltage level is created for each line container. `true` by default.

iidm.import.cgmes.use-previous-values-during-update Optional property that defines whether the CGMES importer should use previous values to fill in missing SSH attributes during an update. When EQ and one or more SSH files are imported separately, and this property is set to `true`, the importer will use values from previously imported SSH files to complete missing attributes in the SSH file currently being imported. If set to `false`, missing SSH attributes will be filled using default values. This property does not apply to SV data. SV data is handled as a complete dataset, with no support for partial updates of the SV file. `false` by default.

iidm.import.cgmes.remove-properties-and-aliases-after-import Properties and aliases are generated during the EQ import process and are used both in the initial import and in subsequent network updates. When this option is set to `true`, all generated properties and aliases are removed after the import/update process. If the option is set to `true` during the initial import, then both the EQ and SSH files must be provided to obtain a valid network at the steady-state hypothesis level. Cgmes importer will import the EQ file, create the properties and aliases, perform the update by importing the SSH file, and finally remove the properties and aliases. If only the EQ file is provided, the properties and aliases will be deleted immediately after the import, not allowing any future update. In this case, the imported network will only be valid at the Equipment level. If the option is set to `true` during an update, the update will be performed and then the properties and aliases will be removed. Removing properties and aliases invalidates all subsequent updates but reduces the size of the IIDM network during serialization, thereby improving performance. This option is suitable when the user does not need to preserve CGMES data for persistency purposes or does not intend to perform further network updates. `false` by default.

iidm.import.cgmes.silence-frequent-issues-warnings
Optional property that defines whether a warning log should be issued for the following issues that happen frequently on real cases:

- cim:OperationalLimit-s which could not be imported into IIDM:
  - cim:OperationalLimit-s of type CurrentLimit, ActivePowerLimit and ApparentPowerLimit can be imported in IIDM only if they relate to Branches (Lines, Tie-Lines, Two Windings Transformers), Three Windings Transformers, and Boundary Lines (at network side, limits at boundary side can not be imported).
  - For all other equipment types, no convertion is done. This is the case for example for Switches, Generators, Loads, etc…
- cim:Switch-es not imported because the import is Bus/Breaker and the switch from Bus and to Bus are the same Bus
- missing minQ/maxQ for cim:EquivalentInjection-s and cim:SynchronousMachine-s

If the option is set to `false`, a warning is logged for every occurrence of the above issues, which may lead to excessive logging in real cases.
If the option is set to `true`, no warning is logged for any of the above issues.

`false` by default.

1.1.4 CGMES post-processors

CgmesDLImportPostProcessor

This post-processor loads the diagram layout (DL) profile contained in the CGMES file, if available, into the triplestore. The diagram layout profile contains the data which is necessary to represent a drawing of the diagram corresponding to the CGMES file. For instance, it contains the position of all equipment.

This post-processor is enabled by adding the name `cgmesDLImport` to the list associated to `iidm.import.cgmes.post-processors` property.

CgmesGLImportPostProcessor

TODO

CgmesMeasurementsPostProcessor

TODO

CgmesShortCircuitPostProcessor

TODO

EntsoeCategoryPostProcessor

TODO

PhaseAngleClock

TODO

1.1.5 Export

There are two main use-cases supported:

- Export IGM (Individual Grid Model) instance files. There is a single network and a unique CGMES modeling authority.
- Export CGM (Common Grid Model) instance files. A network composed of multiple subnetworks, where each subnetwork is an IGM.

In both cases, the metadata model information in the exported files is built from metadata information read from the input files and stored in IIDM or received through parameters. Information received through parameters takes precedence over information available from original metadata models.

For a quick CGM export, the user may rely on the parameter **iidm.export.cgmes.cgm_export** to write in a single export multiple updated SSH files (one for each IGM) and a single SV for the whole common grid model. Specifics about this option are explained in the section *below*. If you need complete control over the exported files in a CGM scenario, you may prefer to iterate through the subnetworks and make multiple calls to the export function. This is described in detail in the section *below*.

Please note that when exporting equipment, PowSyBl always uses the CGMES node/breaker level of detail, without considering the topology level of the PowSyBl network.

The user can specify the profiles to be exported using the parameter **iidm.export.cgmes.profiles**. The list of currently supported export instance files are: EQ, SSH, SV, TP.

If the IIDM network has at least one voltage level with node/breaker topology level, and the SSH or SV is requested in the export, and the TP is not requested, an error will be logged, as there could be missing references in the SSH, SV files to Topological Nodes calculated automatically by IIDM that are not present in the output.

If the dependencies have to be updated automatically (see parameter **iidm.export.cgmes.update-dependencies** below), the exported instance files will contain metadata models where:

- TP and SSH depend on EQ.
- SV depends on TP and SSH.
- EQ depends on EQ_BD (if present). EQ_BD is the profile for the boundary equipment definitions.
- SV depends on TP_BD (if present). TP_BD is the profile for the boundary topology. Only for CGMES 2.4.

The output filenames will follow the pattern `<baseName>_<profile>.xml`. The basename is determined from the parameters, or the basename of the export data source or the main network name.

CGM (Common Grid Model) quick export

When exporting a CGM, we need an IIDM network (CGM) that contains multiple subnetworks (one for each IGM). Only the CGMES instance files corresponding to SSH and SV profiles are exported: an updated SSH file for every subnetwork (for every IGM) and a single SV file for the main network that represents the CGM.

When exporting, it is verified that the main network and all subnetworks have the same scenario time (network case date). If they are different, an error is logged.

If a version number is given as a parameter, it is used for the exported files. If not, the versions of the input CGM SV and IGM SSHs are obtained from their metadata, and their maximum value calculated. The output version is then set to 1 + this maximum value.

The quick CGM export will always write updated SSH files for IGMs and a single SV for the CGM. The parameter for selecting which profiles to export is ignored in this kind of export.

If the dependencies have to be updated automatically (see parameter **iidm.export.cgmes.update-dependencies** below), the exported instance files will contain metadata models where:

- Updated SSH for IGMs supersede the original ones, and depend on the original EQ from IGMs.
- Updated SV for the CGM depends on the updated SSH from IGMs and on the original TP and TP_BD from IGMs.

The filenames of the exported instance files will follow the pattern:

- For the CGM SV: `<basename>_SV.xml`.
- For the IGM SSHs: `<basename>_<IGM name>_SSH.xml`. The IGM name is built from the country code of the first substation or the IIDM name if no country is present.

The basename is determined from the parameters, or the basename of the export data source or the main network name.

As an example, you can export one of the test configurations that have been provided by ENTSO-E. It is available in the cgmes-conformity module of the powsybl-core repository. If you run the following code:

```
Network cgmNetwork = Network.read(CgmesConformity1Catalog.microGridBaseCaseAssembled().
↪dataSource());

Properties exportParams = new Properties();
exportParams.put(CgmesExport.EXPORT_BOUNDARY_POWER_FLOWS, true);
exportParams.put(CgmesExport.NAMING_STRATEGY, "cgmes");
exportParams.put(CgmesExport.CGM_EXPORT, true);
exportParams.put(CgmesExport.UPDATE_DEPENDENCIES, true);
exportParams.put(CgmesExport.MODELING_AUTHORITY_SET, "MAS1");

cgmNetwork.write("CGMES", exportParams, new FileDataSource(Path.of("/exampleFolder"),
↪"exampleBase"));
```

You will obtain the following files in your `exampleFolder`:

```
exampleBase_BE_SSH.xml
exampleBase_NL_SSH.xml
exampleBase_SV.xml
```

where the updated SSH files will supersede the original ones and depend on the original EQs, and the SV will contain the correct dependencies of new SSH and original TPs and TP_BD.

CGM (Common Grid Model) manual export

If you want to intervene in how the updated IGM SSH files or the CGM SV are exported, you can make multiple calls to the CGMES export function.

You can use the following code for reference:

```
Network cgmNetwork = Network.read(CgmesConformity1Catalog.microGridBaseCaseAssembled().
↪dataSource());

// We decide which version we want to export
int exportedVersion = 18;

// Common export parameters
Properties exportParams = new Properties();
exportParams.put(CgmesExport.EXPORT_BOUNDARY_POWER_FLOWS, true);
exportParams.put(CgmesExport.NAMING_STRATEGY, "cgmes");
// We do not want a quick CGM export
exportParams.put(CgmesExport.CGM_EXPORT, false);
exportParams.put(CgmesExport.UPDATE_DEPENDENCIES, false);

Path outputPath = Path.of("/manualExampleFolder");
String basename = "manualExampleBasename";

// For each subnetwork, prepare the metadata for SSH and export it
for (Network n : cgmNetwork.getSubnetworks()) {
    String country = n.getSubstations().iterator().next().getCountry().orElseThrow().
↪toString();
    CgmesMetadataModel sshModel = n.getExtension(CgmesMetadataModels.class).
↪getModelForSubset(CgmesSubset.STEADY_STATE_HYPOTHESIS).orElseThrow();
        sshModel.clearDependencies()
                .addDependentOn("myDependency")
                .addSupersedes("mySupersede")
                .setVersion(exportedVersion)
                .setModelingAuthoritySet("myModellingAuthority");
        exportParams.put(CgmesExport.PROFILES, List.of("SSH"));
    n.write("CGMES", exportParams, new FileDataSource(outputPath, basename + "_" +␣
↪country));
}

// In the main network, CREATE the metadata for SV and export it
cgmNetwork.newExtension(CgmesMetadataModelsAdder.class)
    .newModel()
        .setSubset(CgmesSubset.STATE_VARIABLES)
        .addProfile("http://entsoe.eu/CIM/StateVariables/4/1")
```

(continues on next page)

(continued from previous page)

```
        .setId("mySvId")
        .setVersion(exportedVersion)
        .setModelingAuthoritySet("myModellinAuthority")
        .addDependentOn("mySvDependency1")
        .addDependentOn("mySvDependency2")
    .add()
    .add();
exportParams.put(CgmesExport.PROFILES, List.of("SV"));
cgmNetwork.write("CGMES", exportParams, new FileDataSource(outputPath, basename));
```

The file `manualExampleBasename_BE_SSH.xml` inside `/manualExampleFolder` will have the following contents for the metadata:

```
...
<md:Model.description>CGMES Conformity Assessment ...</md:Model.description>
<md:Model.version>18</md:Model.version>
<md:Model.DependentOn rdf:resource="myDependency"/>
<md:Model.Supersedes rdf:resource="mySupersede"/>
<md:Model.profile>http://entsoe.eu/CIM/SteadyStateHypothesis/1/1</md:Model.profile>
<md:Model.modelingAuthoritySet>myModellingAuthority</md:Model.modelingAuthoritySet>
...
```

And the file `manualExampleBasename_SV.xml` will contain:

```
...
<md:Model.version>18</md:Model.version>
<md:Model.DependentOn rdf:resource="mySvDependency1"/>
<md:Model.DependentOn rdf:resource="mySvDependency2"/>
<md:Model.profile>http://entsoe.eu/CIM/StateVariables/4/1</md:Model.profile>
<md:Model.modelingAuthoritySet>myModelingAuthority</md:Model.modelingAuthoritySet>
...
```

Remember that, in addition to setting the info for metadata models in the IIDM extensions, you could also rely on parameters passed to the export methods.

Topology kind

The elements written in the exported files depend on the topology kind of the export and on the CIM version. By default, the export topology kind is computed from the IIDM network's `VoltageLevel` *connectivity level* as follows:

- If all `VoltageLevel` of the network are at `node/breaker` connectivity level, then the export topology kind is `NODE_BREAKER`
- If all `VoltageLevel` of the network are at `bus/breaker` connectivity level, then the export topology kind is `BUS_BRANCH`
- If some `VoltageLevel` of the network are at `node/breaker` and some other at `bus/breaker` connectivity level, then the export's topology kind depends on the CIM version for export: it is `BUS_BRANCH` for CIM 16 and `NODE_BREAKER` for CIM 100

It is however possible to ignore the computed export topology kind and force it to be `NODE_BREAKER` or `BUS_BRANCH` by setting the parameter *`iidm.export.cgmes.topology-kind`*.

The various configurations and the differences in what's written are summarized in the following table:

CIM version	Exporttopology kind	Connectivity elementsare written	CIM 16 Equipment Operationelements are written
16	NODE_BREAKER	Yes (*)	Yes
16	BUS_BRANCH	No	No
100	NODE_BREAKER	Yes (*)	Yes
100	BUS_BRANCH	Yes (**)	Yes

Connectivity elements

- Non-retained Switch are always written in the case of a NODE_BREAKER export, and never written in the case of a BUS_BRANCH export.
  - Having non-retained open switches in a node/breaker network that is exported as bus/branch may result in multiple connectivity components in the exported network.
  - To avoid this, it would be best to close all non-retained switches in the case before exporting it.
  - Then, the maximum amount of connectivity will be preserved in the export, and the bus/branch exported files can more easily be used for later calculations.
- ConnectivityNode are:
  - Never exported in the case of a CIM 16 BUS_BRANCH export
  - (*) Always exported in the case of a NODE_BREAKER export. If the VoltageLevel's connectivity level is node/breaker, they are exported from nodes, and if the VoltageLevel's connectivity level is bus/breaker, they are exported from buses
  - (**) Exported from buses of the BusBreakerView in case of a CIM 100 BUS_BRANCH export

CIM 16 Equipment Operation elements

If the version is CIM 16, a BUS_BRANCH export is intrinsically linked to not writing the *operation* stereotype elements.

This means the following classes are not written:

- ConnectivityNode
- StationSupply
- GroundDisconnector
- ActivePowerLimit
- ApparentPowerLimit
- LoadArea
- SubLoadArea
- SvStatus

As well as the following attributes:

- LoadGroup.SubLoadArea
- ControlArea.EnergyArea

In CIM 100 these elements have been integrated in the core equipment profile and can be written even if the export is BUS_BRANCH.

Conversion from PowSyBl grid model to CGMES

The following sections describe in detail how each supported PowSyBl network model object is converted to CGMES network components.

Battery

PowSyBl *Battery* is exported as SynchronousMachine with HydroGeneratingUnit.

TODO details

BusbarSection

PowSyBl *BusbarSection* is exported as CGMES BusbarSection.

TODO details

BoundaryLine

PowSyBl *BoundaryLine* is exported as several CGMES network objects. Each boundary line will be exported as one EquivalentInjection and one ACLineSegment.

TODO details

Detailed DC model

DC node

PowSyBl *DC Node* is exported as CGMES DCNode, with attribute:

- EQ DCEquipmentContainer is a CGMES DCConverterUnit, which is the container of the closest converter.

DC Line

PowSyBl *DC Line* is exported as CGMES DCLineSegment, with attribute:

- EQ resistance is copied from R.

DC Switch

PowSyBl *DC Switch* is exported as:

- CGMES DCBreaker if attribute Kind is BREAKER.
- CGMES DCDisconnector if attribute Kind is DISCONNECTOR.

DC Ground

PowSyBl *DC Ground* is exported as CGMES DCGround, with attribute:

- EQ r is copied from R.

AC/DC Converter (Line Commutated Converter, Voltage Source Converter)

PowSyBl *Line Commutated Converter* is exported as CGMES CsConverter, and PowSyBl *Voltage Source Converter* as CGMES VsConverter. They share the following attributes:

- EQ idleLoss is copied from IdleLoss.
- EQ switchingLoss is copied from SwitchingLoss.

- EQ `resistiveLoss` is copied from `ResistiveLoss`.
- EQ `ratedUdc` is copied from the `NominalV` of the associated `DC Node`.
- EQ `PccTerminal` is copied from `PccTerminal`.
- SSH `targetPpcc` is copied from `TargetP`.
- SSH `targetUdc` is copied from `TargetVdc`.
- SSH `p` is the PCC terminal's `P` value.
- SSH `q` is the PCC terminal's `Q` value.

Specific `Line Commutated Converter` attributes:

- SSH `pPccControl` is `CsPpccControlKind.activePower` if `ControlMode` is `P_PCC`, else it is `CsPpccControlKind.dcVoltage`.
- SSH `operatingMode` is `CsOperatingModeKind.rectifier` if the `TargetP` is greater than 0, else it is `CsOperatingModeKind.inverter`.
- SSH `targetAlpha` is defaulted to 0.
- SSH `targetGamma` is defaulted to 0.
- SSH `targetIdc` is defaulted to 0.

Specific `Voltage Source Converter` attributes:

- SSH `pPccControl` is `VsPpccControlKind.pPcc` if `ControlMode` is `P_PCC`, else it is `VsPpccControlKind.udc`.
- SSH `qPccControl` is `VsQpccControlKind.voltagePcc` if `VoltageRegulatorOn` is set to `true`, else it is `VsQpccControlKind.reactivePcc`.
- SSH `targetUpcc` is copied from `VoltageSetpoint`.
- SSH `targetQpcc` is copied from `ReactivePowerSetpoint`.
- SSH `droop` is defaulted to 0.
- SSH `droopCompensation` is defaulted to 0.
- SSH `qShare` is defaulted to 0.

Generator

PowSyBl *Generator* is exported as CGMES `SynchronousMachine`.

Regulating control

If the network comes from a CIM-CGMES model and a generator initially has a `RegulatingControl`, it will still have one at export. Otherwise, a `RegulatingControl` is always exported for generators, except if it has no regulating capabilities because $minQ = maxQ$.

A `RegulatingControl` is exported with `RegulatingControl.mode` set to `RegulatingControlModeKind.reactivePower` when a generator has the extension *RemoteReactivePowerControl* with the `enabled` activated and the generator attribute `voltageRegulatorOn` set to `false`. In all other cases, a `RegulatingControl` is exported with `RegulatingControl.mode` set to `RegulatingControlModeKind.voltage`.

SynchronousMachine type (EQ) and operatingMode (SSH)

The SynchronousMachine.type is exported in the EQ profile depending on the *reactive limits* of the generator or battery and its capacity to behave like a condenser (a battery can behave like a condenser but does not have the flag isCondenser so we consider it as true):

- if the flag isCondenser is true:
  - if the minimum and the maximum active power limit are positive, then the generator or battery will be exported as generatorOrCondenser,
  - if the minimum and the maximum active power limit are negative, then the generator or battery will be exported as motorOrCondenser,
  - if the minimum and the maximum active power limit are both equal to zero, then the generator or battery will be exported as condenser,
  - otherwise, the generator or battery will be exported as generatorOrCondenserOrMotor.
- if the flag isCondenser is false:
  - if the minimum active power limit is positive, then the generator or battery will be exported as generator,
  - if the maximum active power limit is negative, then the generator or battery will be exported as motor,
  - otherwise, the generator will be exported as generatorOrMotor.

The SynchronousMachine.operatingMode is exported in the SSH profile depending on the target active power of the generator or battery and on fact that it is regulating or not:

- if the target active power is positive, then the generator or battery will be exported as generator,
- if the target active power is negative, then the generator or battery will be exported as motor,
- if the target active power is zero and the generator or battery is regulating, then the operating mode will be condenser if it is allowed by its SynchronousMachine.type.
- otherwise, the generator or battery will be exported as generator if is allowed by its SynchronousMachine.type, otherwise motor and otherwise condenser. To know if the generator or battery is behaving as a condenser, its targetV, targetQ and voltageRegulatorOn attributes are used.

HVDC line and HVDC converter stations

A PowSyBl *HVDCLine* and its two *HVDCConverterStations* represents a monopole with ground return configuration. As such, it is exported to CGMES EQ as a DCLineSegment with two DCConverterUnits, where each unit contains:

- A specialization of ACDCConverter, depending on the HvdcConverterStation.HvdcType:
  - A CsConverter if the type is LCC.
  - A VsConverter if the type is VSC.
- A DCGround

Both unit operationMode is DCConverterOperatingModeKind.monopolarGroundReturn.

Both converter ratedUdc is NominalV of the HvdcLine.

As for the CGMES SSH export:

The converter pPccControl is:

- Active power control kind at rectifier side:
  - CsPpccControlKind.activePower for CsConverter.
  - VsPpccControlKind.pPcc for VsConverter.

- DC voltage control kind at inverter side:
  - `CsPpccControlKind.dcVoltage` for `CsConverter`.
  - `VsPpccControlKind.udc` for `VsConverter`.

The corresponding targets are exported as follows:

- At rectifier side, `ACDCConverter.targetPpcc` is equal to `HvdcLine`'s `ActivePowerSetpoint`.
- At inverter side, `ACDCConverter.targetUdc` is equal to $NominalV - U_R$, where U_R is the voltage drop caused by the line's resistance and is equal to: $U_R = R \times \frac{ActivePowerSetpoint}{NominalV}$

For VSC lines, the `qPccControl` is the same for both rectifier and inverter, and depends on the regulation mode:

- It is `VsQpccControlKind.voltagePcc` if VSC voltage regulation is on.
- It is `VsQpccControlKind.reactivePcc` if VSC voltage regulation is off.

The corresponding targets are:

- In case of voltage regulation, `targetUpcc` is set to `VscConverterStation.VoltageSetpoint`.
- In case of reactive power regulation, `targetQpcc` is set to `VscConverterStation.ReactivePowerSetpoint`.

Line

PowSyBl *`Line`* is exported as `ACLineSegment`.

TODO details

Load

PowSyBl *`Load`* is exported as `ConformLoad`, `NonConformLoad` or `EnergyConsumer` depending on the extension *`LoadDetail`*.

TODO details

Fictitious injections (fictitiousP0/fictitiousQ0)

The fictitious injections on buses (bus-branch topology) or on nodes (node-breaker topology) are created using:

- Bus topology: `Bus.setFictitiousP0(double)` and `Bus.setFictitiousQ0(double)`
- Node-breaker: `VoltageLevel.getNodeBreakerView().setFictitiousP0(int node, double)` and `setFictitiousQ0(int node, double)`

These fictitious injections are exported in CGMES as either a `NonConformLoad` or an `EnergySource`, depending on the sign of `fictitiousP0`, with values written to SSH and connectivity/topology bindings set according to the network topology and CIM version. A corresponding 'SvPowerFlow' is written for the terminal in the SV. In case of a node-breaker or CIM100 export, the terminal will refer to a `ConnectivityNode`.

If the EQ profile is not exported and the network contains fictitious injections, note that the references to equipment in the SSH and SV will be invalid.

Shunt compensator

PowSyBl *`ShuntCompensator`* is exported as `LinearShuntCompensator` or `NonlinearShuntCompensator` depending on their models. The CGMES SSH `sections` is written from the IIDM `SectionCount`, and the CGMES SV `SvShuntCompensatorSections.sections` is written from the IIDM `SolvedSectionCount` if present, otherwise `SectionCount`.

Regulating control

If the network comes from a CIM-CGMES model and a shunt compensator initially has a `RegulatingControl`, it will still have one at export.

A shunt compensator with local voltage control (i.e., the regulating terminal is the same of the terminal of connection) and no valid voltage target will not have any exported regulating control. In all other cases, a `RegulatingControl` is exported with `RegulatingControl.mode` set to `RegulatingControlModeKind.voltage`.

StaticVarCompensator

PowSyBl *`StaticVarCompensator`* is exported as `StaticVarCompensator`.

Regulating control

If the network comes from a CIM-CGMES model and a static VAR compensator initially has a `RegulatingControl`, it will still have one at export.

A static VAR compensator which voltage control is local (i.e., the regulating terminal is the same of the terminal of connection) and no valid voltage or reactive power target will not have any exported regulating control.

A `RegulatingControl` is exported with `RegulatingControl.mode` set to `RegulatingControlModeKind.reactivePower` when the static VAR compensator mode is `REACTIVE_POWER`. A `RegulatingControl` is exported with `RegulatingControl.mode` set to `RegulatingControlModeKind.voltage` when the static VAR compensator mode is `VOLTAGE`. When the static VAR compensator is `OFF`, the exported regulating control mode will be reactive power only if the voltage target is not valid but the reactive power target is. Otherwise, the exported mode will be voltage.

Substation

PowSyBl *`Substation`* is exported as `Substation`.

TODO details

Switch

PowSyBl *`Switch`* is exported as CGMES `Breaker`, `Disconnector` or `LoadBreakSwitch` depending on its `SwitchKind`.

TODO details

ThreeWindingsTransformer

PowSyBl *`ThreeWindingsTransformer`* is exported as `PowerTransformer` with three `PowerTransformerEnds`. If the transformer has a `TapChanger`, the CGMES SSH `step` is written from the IIDM `TapPosition` and the CGMES SV `SVtapStep` is written from the IIDM `SolvedTapPosition` if it is not null, otherwise `TapPosition`.

Tap changer control

If the network comes from a CIM-CGMES model and the tap changer has initially a `TapChangerControl`, it always has at export too. Otherwise, a `TapChangerControl` is exported for the tap changer if it is considered as defined. A `RatioTapChanger` is considered as defined if it has a valid regulation value, a valid target deadband and a non-null regulating terminal. A `PhaseTapChanger` is considered as defined if it has a valid regulation value, a valid target deadband, and a non-null regulating terminal.

In a `RatioTapChanger`, the `TapChangerControl` is exported with `RegulatingControl.mode` set to `RegulatingControlModeKind.reactivePower` when `RatioTapChanger regulationMode` is set to

REACTIVE_POWER, and with RegulatingControl.mode set to RegulatingControlModeKind.voltage when RatioTapChanger regulationMode is set to VOLTAGE.

In a PhaseTapChanger, the TapChangerControl is always exported with RegulatingControl.mode set to RegulatingControlModeKind.activePower. If the original PhaseTapChanger regulationMode is CURRENT_LIMITER, the TapChangerControl regulation is disabled, the regulation target value and deadband are set to 0, and an OperationalLimitSet with a CurrentLimit is created at the regulated terminal with the regulation value.

TwoWindingsTransformer

PowSyBl *TwoWindingsTransformer* is exported as PowerTransformer with two PowerTransformerEnds.

If the transformer has a TapChanger, the CGMES SSH step is written from the IIDM TapPosition and the CGMES SV SVtapStep is written from the IIDM SolvedTapPosition if it is not null, otherwise TapPosition.

Tap changer controls for two-winding transformers are exported following the same rules explained in the previous section about three-winding transformers. See *tap changer control*.

Operational limits

PowSyBl exports IIDM loading limits to CGMES OperationalLimit elements as follows:

- Permanent limits are exported with type PATL (Permanent Allowable Transmission Limit) and a corresponding OperationalLimit value.
- Temporary limits are exported with type TATL (Temporary Allowable Transmission Limit), parameterized by the acceptable duration in seconds. If a temporary limit name is empty in IIDM, a fallback name is written in EQ as: TATL (for example: TATL 600).

This applies to CurrentLimits, ActivePowerLimits, and ApparentPowerLimits.

Voltage level

PowSyBl *VoltageLevel* is exported as VoltageLevel.

TODO details

Extensions

Control areas

PowSyBl *ControlAreas* are exported as several ControlArea.

TODO details

Options

These properties can be defined in the configuration file in the *import-export-parameters-default-value* module.

Note that if you are exporting a network that does not come from CGMES, you can use the *iidm.import.cgmes.boundary-location* property to define the location of the boundary files to use as reference.

iidm.export.cgmes.base-name Optional property that defines the base name of the exported files. Exported CGMES files' names will look like this:

```
<base_name>_EQ.xml
<base_name>_TP.xml
<base_name>_SSH.xml
<base_name>_SV.xml
```

By default, the base name is the network's name if it exists, or else the network's ID.

iidm.export.cgmes.boundary-eq-id Optional property that defines the ID of the EQ-BD model if there is any. Its default value is `null`: we consider there is no EQ-BD model to consider. If this property is defined, then this ID will be written in the header of the exported EQ file.

iidm.export.cgmes.boundary-tp-id Optional property that defines the ID of the TP-BD model if there is any. Its default value is `null`: we consider there is no TP-BD model to consider. If this property is defined, then this ID will be written in the header of the exported SV file.

iidm.export.cgmes.cim-version Optional property that defines the CIM version number in which the user wants the CGMES files to be exported. CIM versions 16 and 100 are supported i.e. its valid values are `16` and `100`. If not defined, and the network has the extension `CimCharacteristics`, the CIM version will be the one indicated in the extension. If not, its default value is `16`. CIM version 16 corresponds to CGMES 2.4.15. CIM version 100 corresponds to CGMES 3.0.

iidm.export.cgmes.encode-ids Optional property that must be used if IIDM IDs that are not compliant with CGMES requirements are to be used as CGMES IDs. `true` by default. Used for debugging purposes.

iidm.export.cgmes.export-boundary-power-flows Optional property that defines if power flows at boundary nodes are to be exported in the SV file or not. `true` by default.

iidm.export.cgmes.export-power-flows-for-switches Optional property that defines if power flows of switches are exported in the SV file. `true` by default.

iidm.export.cgmes.naming-strategy Optional property that defines which naming strategy is used to transform IIDM identifiers to CGMES identifiers. Available naming strategies are:

- `identity`: For IIDM objects that have an ID (e.g. TwoWindingTransformer), the CGMES ID is identical to the IIDM ID. For IIDM objects that don't have an ID (e.g. TapChanger), either the CGMES ID is contained in a property or an alias (this is typically the case when the network is the result of a CGMES import) and is exported as such, or there is no such property or alias and a combination of IIDM properties and constants is used to generate the CGMES ID.
- `cgmes`: The ID that would be generated by the identity naming strategy serves as basis. When that ID is CGMES compliant (against IEC 61970-552), it is exported as such. Otherwise, that ID is used to generate a compliant CGMES one in a deterministic way.

Its default value is `identity`. You can also define a custom naming strategy by implementing the `NamingStrategy` interface on your own project and declare a `NamingStrategyProvider` that can be automatically discovered. Then in this parameter, you can specify the name of the provider.

iidm.export.cgmes.uuid-namespace Optional property related to the naming strategy specified in `iidm.export.cgmes.naming-strategy`. When new CGMES IDs have to be generated, a mechanism that ensures creation of new, stable identifiers based on IIDM IDs is used (see RFC 4122). These new IDs are guaranteed to be unique inside a namespace given by this UUID. By default, it is the name-based UUID fo the text "powsybl.org" in the empty namespace.

iidm.export.cgmes.profiles Optional property that determines which instance files will be exported. By default, it is a full CGMES export: the instance files for the profiles EQ, TP, SSH and SV are exported.

iidm.export.cgmes.topology-kind Optional property that defines the topology kind of the export. Allowed values are: `NODE_BREAKER` and `BUS_BRANCH`. By default, the export topology kind reflects the network's voltage levels connectivity level detail: node/breaker or bus/breaker. This property is used to bypass the natural export topology kind and force a desired one (e.g. export as bus/branch a node/breaker network).

iidm.export.cgmes.modeling-authority-set Optional property allowing to write a custom modeling authority set in the exported file headers. `powsybl.org` by default. If a Boundary set is given with the property `iidm.import.cgmes.boundary-location` and the network sourcing actor is found inside it, then the modeling authority set will be obtained from the boundary file without the need to set this property. The sourcing actor can be specified using the parameter `iidm.export.cgmes.sourcing-actor`.

iidm.export.cgmes.model-description Optional property allowing to write a custom model description in the file headers. By default, the model description is `EQ model` for the EQ file, `TP model` for the TP file, `SSH model` for the SSH file and `SV model` for the SV file.

iidm.export.cgmes.export-transformers-with-highest-voltage-at-end1 Optional property defining whether the transformers should be exported with the highest voltage at end 1, even if it might not be the case in the IIDM model. This property is set to `false` by default.

iidm.export.cgmes.export-load-flow-status Optional property that indicates whether the load flow status (`converged` or `diverged`) should be written for the `TopologicalIslands` in the SV file. If `true`, the status will be computed by checking, for every bus, if the voltage and angle are valid, and if the bus is respecting Kirchhoff's first law. For the latter, we check that the sums of active power and reactive power at the bus are higher than a threshold defined by the properties `iidm.export.cgmes.max-p-mismatch-converged` and `iidm.export.cgmes.max-q-mismatch-converged`. This property is set to `true` by default.

iidm.export.cgmes.export-all-limits-group Optional property that defines whether all OperationalLimitsGroup should be exported, or only the selected (active) ones. This property is set to `true` by default, which means all groups are exported (not only the active ones).

iidm.export.cgmes.export-generators-in-local-regulation-mode Optional property that allows to export voltage regulating generators in local regulation mode. This doesn't concern reactive power regulating generators. If set to true, the generator's regulating terminal is set to the generator's own terminal and the target voltage is rescaled accordingly. This property is set to `false` by default.

iidm.export.cgmes.max-p-mismatch-converged Optional property that defines the threshold below which a bus is considered to be balanced for the load flow status of the `TopologicalIsland` in active power. If the sum of all the active power of the terminals connected to the bus is greater than this threshold, then the load flow is considered to be divergent. Its default value is `0.1`, and it should be used only if the `iidm.export.cgmes.export-load-flow-status` property is set to `true`.

iidm.export.cgmes.max-q-mismatch-converged Optional property that defines the threshold below which a bus is considered to be balanced for the load flow status of the `TopologicalIsland` in reactive power. If the sum of all the reactive power of the terminals connected to the bus is greater than this threshold, then the load flow is considered to be divergent. Its default value is `0.1`, and it should be used only if the `iidm.export.cgmes.export-load-flow-status` property is set to `true`.

iidm.export.cgmes.export-sv-injections-for-slacks Optional property to specify if the total mismatch left after power flow calculation at IIDM slack buses should be exported as an SvInjection. This property is set to `true` by default.

iidm.export.cgmes.sourcing-actor Optional property allowing to specify a custom sourcing actor. If a Boundary set with reference data is provided for the export through the parameter `iidm.import.cgmes.boundary-location`, the value of this property will be used to look for the modeling authority set and the geographical region to be used in the export. No default value is given. If this property is not given, the export process will still try to determine the sourcing actor from the IIDM network if it only contains one country.

iidm.export.cgmes.model-version Optional property defining the version of the exported CGMES file. It will be used if the version is not already available in the network. The version will be written in the header of each exported file and will also be used to generate a unique UUID for the `FullModel` field. Its default value is 1.

iidm.export.cgmes.business-process The business process in which the export takes place. This is used to generate unique UUIDs for the EQ, TP, SSH and SV file `FullModel`. Its default value is `1D`.

iidm.export.cgmes.cgm_export Optional property to specify the export use-case: IGM (Individual Grid Model) or CGM (Common Grid Model). To export instance files of a CGM, set the value to `True`. The default value is `False` to export network as an IGM.

iidm.export.cgmes.update-dependencies Optional property to determine if dependencies in the exported instance files should be managed automatically. The default value is `True`.

1.1.6 Examples

Have a look at the CGMES sample files from ENTSO-E Test Configurations for Conformity Assessment Scheme v2.0.

The CGMES (**C**ommon **G**rid **M**odel **E**xchange **S**pecification) is an IEC technical specification (TS 61970-600-1, TS 61970-600-2) based on the IEC CIM (**C**ommon **I**nformation **M**odel) family of standards. It was developed to meet the necessary requirements for TSO data exchanges in the areas of system development and system operation. In this scenario the agents (the Modelling Authorities) generate their Individual Grid Models (IGM) that can be assembled to build broader Common Grid Models (CGM). Boundaries between IGMs are well-defined: the boundary data is shared between the modeling agents and contain all boundary points required for a given grid model exchange.

In CGMES, an electric power system model is described by data grouped in different subsets (profiles) and exchanged as CIM/XML files, with each file associated to a given profile. The profiles considered in PowSyBl are:

- `EQ` Equipment. Contains data that describes the equipment present in the network and its physical characteristics.
- `SSH` Steady State Hypothesis. Required input parameters to perform power flow analysis; e.g., energy injections and consumptions and setpoint values for regulating controls.
- `TP` Topology. Describe how the equipment is electrically connected. Contains the definition of power flow buses.
- `SV` State Variables. Contains all the information required to describe a steady-state power flow solution over the network.
- `EQBD` Equipment Boundary. Contains definitions of the equipment in the boundary.
- `TPBD` Topology Boundary. Topology information associated with the boundary.
- `DL` Diagram Layout. Contains information about diagram positions.
- `GL` Geographical Layout. Contains information about geographical positions.

CGMES model connectivity can be defined at two different levels of detail:

`Node/breaker` This is the level of detail required for Operation. The `EQ` contains Connectivity Nodes where the conducting equipment is attached through its Terminals. All switching devices (breakers, disconnectors, ...) are modelled. The contents of the `TP` file must be the result of the topology processing over the graph defined by connectivity nodes and switching devices, taking into account its open/closed status.

`Bus/branch` No Connectivity Nodes are present in the `EQ` file. The association of every equipment to a bus is defined directly in the `TP` file, that must be provided.

1.2 UCTE-DEF

1.2.1 Format specification

The UCTE-DEF (UCTE **D**ata **E**xchange **F**ormat) format is an exchange format specified by the UCTE, for the exchange of grid model among its members. The data refer to load flow and three-phase short-circuit studies and describe the interconnected extra high-voltage network. The data are contained in an unformatted standard US ASCII file. The file is divided into 7 different blocks:

- Comments (C)
- Nodes (N)
- Lines (L)
- Two-winding transformers (T)
- Two-winding transformers regulation (RR)
- Two-winding transformers special description (TT)
- Exchange powers (E)

Each block is introduced by a key line consisting of the two characters ## and of the character given above in brackets. The end of a block is given by the next key line or the end of the file. The information of each block is written in lines, and the contents are separated by a blank (empty space).

The grid is described in Bus/Branch topology, and only a few types of equipment are supported (nodal injections, AC line, two-winding transformer). Fictitious nodes are located at the electric middle of each tie line. The defined X-nodes are binding for all users.

File name convention

The UCTE-DEF format use the following file name convention: `<yyyymmdd>_<HHMM>_<TY><w>_<cc><v>.uct` with:

- `yyyymmdd`: year, month and day
- `HHMM`: hour and minute
- `TY`: the file type
  - `FO`: Day ahead congestion forecast
  - `2D`: 2-days ahead congestion forecast
  - `SN`: Snapshots
  - `RE`: Reference
  - `LT`: Long-term reference
  - `01...23`: Intra-day ahead congestion forecast. The value is the number of hours separating the case date and the generation date.
- `w`: day of the week, starting with 1 for Monday
- `cc`: The ISO country-code for national datasets, `UC` for UCTE-wide merged datasets without X nodes and `UX` for UCTE-wide merged datasets with X nodes
- `v`: version number starting with 0

The specifications of the UCTE-DEF format are available online.

1.2.2 Import

The import of a UCTE file into an IIDM grid model is done in three steps. First, the file is parsed and a UCTE grid model is created in memory. Then, some inconsistency checks are performed on this model. Finally, the UCTE grid model is converted into an IIDM grid model.

The UCTE parser provided by PowSyBl does not support the blocks `##TT` and `##E` providing respectively a special description of the two-winding transformers and the scheduled active power exchange between countries. Those blocks are ignored during the parsing step.

Options

Parameters for the import can be defined in the configuration file in the *import-export-parameters-default-value* module.

`ucte.import.ucte.import.combine-phase-angle-regulation`

The `ucte.import.ucte.import.combine-phase-angle-regulation` property is an optional property that defines if the ratio and phase tap changers of transformers should, when both are present, be combined.

The default value is `false`.

ucte.import.create-areas

The `ucte.import.create-areas` property is an optional property that defines if *areas* should be created in the imported IIDM grid model.

The default value is `true`. For more details see further below about *area conversion*.

ucte.import.areas-dc-xnodes

The `ucte.import.areas-dc-xnodes` property is an optional property that defines the list of X-nodes that should be considered as *area* DC boundaries.

The default value is an empty list. For more details see further below about *area conversion*.

Inconsistency checks

Once the UCTE grid model is created in memory, a consistency check is performed on the different elements (nodes, lines, two-winding transformers and regulations).

In the tables below, we summarize the inconsistency checks performed on the network for each type of equipment. For the sake of clarity, we use notations that are made explicit before each table.

Nodes with active or reactive power generation

Notations:

- P: active power generation, in MW
- Q: reactive power generation, in MVar
- $minP$: minimum permissible active power generation in MW
- $maxP$: maximum permissible active power generation in MW
- $minQ$: minimum permissible reactive power generation in MVar
- $maxQ$: maximum permissible reactive power generation in MVar
- $Vreg$: voltage regulation, which can be enabled or disabled
- $Vref$: voltage reference, in V

Check	Consequence
$Vreg$ enabled and Q undefined	$Q = 0$
P undefined	$P = 0$
Q undefined	$Q = 0$
$Vreg$ enabled and $Vref$ undefined or $\in [0, 0.0001[$	Node type = PQ
$minP$ undefined	$minP = -9999$
$maxP$ undefined	$maxP = 9999$
$minQ$ undefined	$minQ = -9999$
$maxQ$ undefined	$maxQ = 9999$
$minP > maxP$	$minP$ switched with $maxP$
$minQ > maxQ$	$minQ$ switched with $maxQ$
$P > maxP$	$maxP = P$
$P < minP$	$minP = P$
$Q > maxQ$	$maxQ = Q$
$Q < minQ$	$minQ = Q$
$minP = maxP$	$maxP = 9999$ and $minP = -9999$
$minQ = maxQ$	$maxQ = 9999$ and $minQ = -9999$
$maxP > 9999$	$maxP = 9999$
$minP < -9999$	$minP = -9999$
$maxQ > 9999$	$maxQ = 9999$
$minQ < -9999$	$minQ = -9999$

Lines or two-winding transformers

Notations:

- X: reactance in Ω

Check	Consequence
$X \in [-0.05, 0.05]$	$X = \pm 0.05$
Current limit < 0	Current limit value ignored

Regulations

Phase regulation

Check	Consequence
Voltage value $<= 0$	Voltage value set to NaN
Invalid ($n = 0$) or undefined (n, n' or δU) data provided	Regulation ignored

Angle regulation

Check	Consequence
Invalid ($n = 0$) or undefined (n, n' or δU) data provided	Regulation ignored
Undefined type	type set to `ASYM`

From UCTE to IIDM

The UCTE file name is parsed to extract metadata required to initialize the IIDM network, such as its ID, the case date and the forecast distance.

Node conversion

There is no equivalent *voltage level* or *substation* concept in the UCTE-DEF format, so we have to create substations and voltage levels from the node description and the topology. Two nodes are in the same substation if they have the same geographical spot (the 1st-6th character of the node code) or are connected by a two-winding transformer, a coupler or a low-impedance line. Two nodes are in the same voltage level if their code only differs by the eighth character (busbars identifier).
Note: We do not create a substation, a voltage level or a bus for X-nodes. They are converted to *boundary lines*.

For nodes with a valid active or reactive load, a *load* is created. Its ID is equal to the ID of the bus post-fixed by `_load`. The `P0` and `Q0` are equal to the active load and the reactive load of the UCTE node. For those with a valid generator, a *generator* is created. Its ID is equal to the ID of the bus post-fixed by `_generator`. The power plant type is converted to an energy source value (see the mapping table below for the matching).

Mapping table for power plant types:

UCTE Power plant type	IIDM Energy source
Hydro (H)	Hydro
Nuclear (N)	Nuclear
Lignite (L)	Thermal
Hard coal (C)	Thermal
Gas (G)	Thermal
Oil (O)	Thermal
Wind (W)	Wind
Further (F)	Other

The list of the power plant types is more detailed than the list of available energy source types in IIDM, so we add the `powerPlantType` property to each generator to keep the initial value.

Line conversion

The busbar couplers connecting two real nodes are converted into a *switch*. This switch is open or closed depending on the status of the coupler. In the UCTE-DEF format, the coupler can have extra information not supported in IIDM we keep as properties:

- the current limits is stored in the `currentLimit` property
- the order code is stored in the `orderCode` property
- the element name is stored in the `elementName` property

The lines connected between two real nodes are converted into an *AC line*, except if its impedance is too small (e.g. smaller than `0.05`). In that particular case, the line is considered as a busbar coupler, and a *switch* is created. The susceptance of the UCTE line is split in two, to initialize `B1` and `B2` with equal values. If the current limits are defined, a permanent limit is created for both ends of the line. The element name of the UCTE line is stored in the `elementName` property.

The lines connected between a read node and an X-node are converted into a *boundary line*. In IIDM, a boundary line is a line segment connected to a constant load. The sum of the active load and generation (rep. reactive) is computed to initialize the `P0` (resp. `Q0`) of the boundary line. The element name of the UCTE line is stored in the `elementName` property and the geographical name of the X-node is stored in the `geographicalName` property.

Two-winding transformer conversion

The two-winding transformers connected between two real nodes are converted into a *two-winding transformer*. If the current limits are defined, a permanent limit is created only for the second side. The element name of the transformer is stored in the `elementName` property and the nominal power is stored in the `nominalPower` property.

If a two-winding transformer is connected between a real node and an X-node, a fictitious intermediate voltage level is created, with a single bus called a Y-node. This new voltage level is created in the same substation as the real node. The transformer is created between the real node and the new Y-node, and the X-node is converted into a boundary line. The only difference with a classic X-node conversion, is that the electrical characteristic are hold by the transformer and set to `0` for the boundary line, except for the reactance that is set to 0.05Ω.

TODO: insert a schema

Phase regulation

If a phase regulation is defined for a transformer, it is converted into a *ratio tap changer*. If the voltage setpoint is defined, the ratio tap changer will regulate the voltage to this setpoint. The regulating terminal is assigned to the first side of the transformer. The of each step is calculated according to the following formula: $\rho = 1/(1 + i * \delta U/100)$.

Angle regulation

If an angle regulation is defined for a transformer, it is converted into a *phase tap changer*, with a `CURRENT_LIMITER` regulation mode with `regulating=false`. and of each step are calculated according to the following formulas:

- for an asymmetrical regulation (e.g. $\Theta \neq 90$)

$$\begin{aligned} dx &= step_i \times \frac{\delta U}{100} \times \cos(\Theta) \\ dy &= step_i \times \frac{\delta U}{100} \times \sin(\Theta) \\ \rho &= \frac{1}{\sqrt{dy^2 + (1 + dx)^2}} \\ \alpha &= -\text{atan2}(dy, 1 + dx) \end{aligned}$$

- for a symmetrical regulation (e.g. $\Theta = 90$)

$$\begin{aligned} dx &= step_i \times \frac{\delta U}{100} \times \cos(\Theta) \\ dy &= step_i \times \frac{\delta U}{100} \times \sin(\Theta) \\ \rho &= 1 \\ \alpha &= -2 \times \text{atan2}(dy, 2 \times (1 + dx)) \end{aligned}$$

Note: The sign of α is changed because the phase tap changer is on side 2 in UCTE-DEF, and on side 1 in IIDM.

Area conversion

When the *`ucte.import.create-areas` option* is set to `true` (default), the importer will create *areas* in the IIDM Network.

The areas are created on the basis of the countries present in the input UCTE-DEF file. Each country present results in the creation of an Area in the IIDM model with:

- Area **ID**: set to the country alpha-2 code

- Area **Type**: hardcoded to `ControlArea`
- Area **VoltageLevels**: all VoltageLevels created in the Country (see *Node conversion*)
- Area **Boundaries**: all BoundaryLines in the Country

By default, all Area Boundaries are flagged as AC, because the UCTE-DEF format does not have any HVDC explicit description. To specify which boundaries should be considered as DC in the conversion, you may supply a list of X-nodes IDs in the *`ucte.import.areas-dc-xnodes` option*.

Note: Creating areas for German subregions / TSOs is not supported today (D2, D4, D7, D8). Let us know if you have this use case by entering an issue in our GitHub. We also welcome contributions.

1.2.3 Export

TODO

Options

These properties can be defined in the configuration file in the *import-export-parameters-default-value* module.

ucte.export.naming-strategy The `ucte.export.naming-strategy` property is an optional property that defines the naming strategy to be used for UCTE export.

Its default value is `Default`, which corresponds to an implementation that expects the network elements' ID to be totally compatible with UCTE-DEF norm (e.g., a network initially imported from a UCTE-DEF file), and throws an exception if any network element does not respect the norm. It does not do any ID modification.

The **U**nion for the **C**o-ordination of **T**ransmission of **E**lectricity, created in 1951, coordinated the operation and development of the electricity transmission grid for the Continental European synchronously operated transmission grid, thus providing a reliable platform to all participants of the Internal Electricity Market and beyond.

In 1999, UCTE re-defined itself as an association of TSOs in the context of the Internal Energy Market. Building on its experience with recommendations, UCTE turned to make its technical standards. These standards became indispensable for the reliable international operation of the high-voltage grids which are all working at one "heart beat": the 50Hz UCTE frequency related to the nominal balance between generation and the electricity demand of some 500 million people in one of the biggest electrical synchronous interconnections worldwide.

On 1 July 2009, UCTE was wound up. All operational tasks were transferred to ENTSO-E.

1.3 IIDM exchange formats

1.3.1 Import

Options

These properties can be defined in the configuration file in the *import-export-parameters-default-value* module.

iidm.import.xml.throw-exception-if-extension-not-found The `iidm.import.xml.throw-exception-if-extension-not-found` property is an optional property that defines if the XIIDM importer throws an exception while trying to import an unknown or undeserializable extension or if it just ignores it.

By default, the value is `false`.

iidm.import.xml.included.extensions The `iidm.import.xml.included.extensions` property is an optional property that defines the list of extensions that will be imported by the XIIDM importer. By default, all extensions will be imported.

When set to an empty string, all extensions will be ignored during the import.

iidm.import.xml.excluded.extensions The `iidm.import.xml.excluded.extensions` property is an optional property that defines the list of extensions that will not be imported by the XIIDM importer. When both `iidm.import.xml.included.extensions` and `iidm.import.xml.excluded.extensions` are defined, a configuration exception is thrown.

By default, no extension is excluded from the import.

iidm.import.xml.with-automation-systems The `iidm.import.xml.with-automation-systems` property is an optional property that defines whether to import or not the network overload management systems. It should be set as a double value.

By default, the value is `true`, the overload management systems are imported when deserializing a network.

iidm.import.xml.missing-permanent-limit-percentage The `iidm.import.xml.missing-permanent-limit-percentage` property is an optional property that defines the percentage applied to the lowest temporary limit to compute the permanent limit when missing (for IIDM < 1.12 only).

By default, it is set to `100`, the computed permanent limit is then the same as the lowest temporary limit.

iidm.import.minimal-validation-level The `iidm.import.minimal-validation-level` property is an optional property that allows to override the network minimum validation level among the accepted validation levels. The possible validation levels are `EQUIPMENT` and `STEADY_STATE_HYPOTHESIS`.

By default, the value is `null`, the network validation level remains the same.

Deprecated properties

throwExceptionIfExtensionNotFound The `throwExceptionIfExtensionNotFound` property is deprecated since v2.0.0. Use the `iidm.import.xml.throw-exception-if-extension-not-found` property instead.

Removed properties

iidm.import.xml.import-mode The `iidm.import.xml.import-mode` property is an optional property that defines the import mode of the XIIDM importer.

Its possible values are :

- `UNIQUE_FILE`: Imports the network and its extensions from a unique file.
- `EXTENSIONS_IN_ONE_SEPARATED_FILE`: Imports the network from a file and the extensions from another file. In this mode, if the network file name is network.xiidm, the extension file name must be network-ext.xiidm.
- `ONE_SEPARATED_FILE_PER_EXTENSION_TYPE`: Imports the network from a file and each extension type from a separate file. In this mode, if the network file name is `network.xiidm`, each extension file name must be `network-extensionName.xiidm`. Example: if our network has two extensions `loadFoo` and `loadBar`, then the network will be imported from the `network.xiidm` file and `loadFoo` and `loadBar` will be imported respectively from `network-loadFoo.xiidm` and `network-loadBar.xiidm`.

The default value of this parameter is `NO_SEPARATED_FILE_FOR_EXTENSIONS`. This property has been removed in v3.3.0.

1.3.2 Export

Networks can be exported in three IIDM formats:

- XML: “XIIDM”
- JSON: “JIIDM”
- binary: “BIIDM”

Exporting with default values

To export a network using the default parameters, use:

```
Network n = ...;
String FORMAT = "XIIDM"; // or "BIIDM" or "JIIDM"
n.write(FORMAT, new Properties(), Path.of("/path/to/output.format"));
```

Exporting with custom properties

You can configure your export either with `ExportOptions` or with `Properties`.

From ExportOptions

```
Network n = ...;
ExportOptions options = new ExportOptions();
options.setVersion("1.13");
options.setFormat(TreeDataFormat.BIN); // or XML or JSON
NetworkSerDe.write(n, options, Path.of("/tmp/test.biidm"));
```

From config file

```
Network n = ...;
Properties prop = new Properties();
prop.load(new FileInputStream("/path/to/powsybl_config.yaml"));
String FORMAT = "XIIDM"; // or "BIIDM" or "JIIDM"
n.write(FORMAT, prop, Path.of("/path/to/output.format"));
```

With the config file containing at least:

```
import-export-parameters-default-value:
  iidm.export.xml.version: 1.12
```

This configuration allows to export to IIDM v1.12. Note that the parameter includes "xml" in the name, but it can be used for any IIDM format (including JIIDM and BIIDM). For more information about the config file, see *import-export-parameters-default-value* module, or the *options below*.

Options

These properties can be defined in the configuration file in the *import-export-parameters-default-value* module.

iidm.export.xml.indent The `iidm.export.xml.indent` property is an optional property that defines whether the XIIDM file generated by the XIIDM exporter will be indented.

By default, the value is `true`.

iidm.export.xml.with-branch-state-variables The `iidm.export.xml.with-branch-state-variables` property is an optional property that defines whether the network will be exported by the XIIDM exporter with branch state variables.

By default, the value is `true`.

iidm.export.xml.only-main-cc The `iidm.export.xml.only-main-cc` property is an optional property that defines whether the XIIDM exporter should only export the main connected component of the network.

By default, the value is `false`.

iidm.export.xml.anonymised The `iidm.export.xml.anonymised` property is an optional property that defines whether the XIIDM exporter anonymizes all equipment in the generated file.

By default, the value is `false`.

iidm.export.xml.topology-level The `iidm.export.xml.topology-level` property is an optional property that defines the most detailed topology in which the XIIDM exporter can export the network. The topology level can be:

- `NODE_BREAKER`: the voltage levels are exported using the Node/Breaker view. Voltage levels described in Bus/Breaker topology are exported using the Bus/Breaker view.
- `BUS_BREAKER`: all voltage levels are all exported using the Bus/Breaker view
- `BUS_BRANCH`: all voltage levels are exported using the Bus view

The default value is `NODE_BREAKER` to export all voltage levels in the same level of details than the one they are described.

iidm.export.xml.topology-level.voltage-levels.node-breaker The `iidm.export.xml.topology-level.voltage-levels.node-breaker` property is an optional property that defines a list of voltage level IDs to be exported using the Node/Breaker view by the IIDM exporter. Note: if the voltage level is described using a lower detailed topology (Bus/Breaker or Bus/Branch), it will be exported using the Bus/Breaker (or Bus/Branch) view (Node/Breaker is not possible since the voltage level has not enough details).

iidm.export.xml.topology-level.voltage-levels.bus-breaker The `iidm.export.xml.topology-level.voltage-levels.bus-breaker` property is an optional property that defines a list of voltage level IDs to be exported using the Bus/Breaker view by the IIDM exporter.

iidm.export.xml.topology-level.voltage-levels.bus-branch The `iidm.export.xml.topology-level.voltage-levels.bus-branch` property is an optional property that defines a list of voltage level IDs to be exported using the Bus/Branch view by the IIDM exporter.

iidm.export.xml.throw-exception-if-extension-not-found The `iidm.export.xml.throw-exception-if-extension-not-found` property is an optional property that defines whether the XIIDM exporter throws an exception if the network contains an unknown or unserializable extension or if it just ignores it.

By default, the value is `false`.

iidm.export.xml.included.extensions The `iidm.export.xml.included.extensions` property is an optional property that defines the list of extensions that we will be exported by the XIIDM exporter. If set to an empty string, all extensions will be ignored during the export.

By default, all extensions will be exported.

iidm.export.xml.excluded.extensions The `iidm.export.xml.excluded.extensions` property is an optional property that defines the list of extensions that will not be exported by the XIIDM exporter. When both `iidm.export.xml.included.extensions` and `iidm.export.xml.excluded.extensions` are defined a configuration exception is thrown.

By default, no extension is excluded from the export.

iidm.export.xml.sorted The `iidm.export.xml.sorted` property is an optional property that defines whether the XIIDM file generated by the XIIDM exporter will be sorted or not for some objects. Depending on object types the following sorting key has been chosen :

- the id for identifiables
- the name for extensions
- the name for temporary limits
- node1 then node2 for internal connections
- the name for properties of an identifiable

By default, the network components are not sorted.

iidm.export.xml.flatten The `iidm.export.xml.flatten` property is an optional property that defines whether the XIIDM exporter will flatten the network by merging subnetworks into the main network during export. When set to `true`, all subnetworks are integrated into a single flat network structure in the generated XIIDM file. If the network contains no subnetworks, setting this property to `true` has no effect.

Its default value is `false`.

iidm.export.xml.version The `iidm.export.xml.version` property is an optional property that defines the XIIDM version to use for the exported file. If the chosen version is not compatible with the network to write, an error occurs. This is typically the case when an attribute appeared in a version more recent than the target one, and its value is not the default one (importing back the file will lead to a different network).

By default, the export is done in the more recent version that is supported.

iidm.export.xml.with-automation-systems The `iidm.export.xml.with-automation-systems` property is an optional property that defines if the automation systems are exported.

By default, the value is `true`, automation systems are exported.

iidm.export.xml.iidm-version-incompatibility-behavior The `iidm.export.xml.iidm-version-incompatibility-behavior` property is an optional property that defines the behavior of the XIIDM exporter when there is a version incompatibility between the IIDM network and the XIIDM version in which the export is done. There are two possible behaviors:

- `LOG_ERROR`: an error is logged when there is a version incompatibility
- `THROW_EXCEPTION`: an exception is thrown when there is a version incompatibility

By default, this behavior is set as `THROW_EXCEPTION`

iidm.export.xml.bus-branch.voltage-level.incompatibility-behavior The `iidm.export.xml.bus-branch.voltage-level.incompatibility-behavior` property is an optional property that defines the behavior of the XIIDM exporter when exporting a network in BUS_BRANCH topology with a voltage level that will be invalid due to a reference to a non-exported bus. There are two possible behaviors:

- `KEEP_ORIGINAL_TOPOLOGY`: the problematic voltage level is exported keeping the original topology and a warning is logged
- `THROW_EXCEPTION`: an exception is thrown when trying to export a network in BUS_BRANCH topology, and a problematic voltage level is detected.

By default, this behavior is set as `THROW_EXCEPTION`

Removed properties

iidm.export.xml.export-mode The `iidm.export.xml.export-mode` property is an optional property that defines the export mode of the XIIDM exporter. The export mode can be:

- `UNIQUE_FILE`: Exports the network and its extensions in a unique file.
- `EXTENSIONS_IN_ONE_SEPARATED_FILE`: Exports the network to a file and the extensions to another file. In this mode, if the network file name is `network.xiidm`, the extension file name must be `network-ext.xiidm`.
- `ONE_SEPARATED_FILE_PER_EXTENSION_TYPE`: Exports the network to a file and each extension type to a separate file. In this mode, if the network file name is `network.xiidm`, each extension file name must be `network-extensionName.xiidm`. Example: if our network has two extensions `loadFoo` and `loadBar`, then the network will be exported to the `network.xiidm` file and `loadFoo` and `loadBar` will be exported respectively to `network-loadFoo.xiidm` and `network-loadBar.xiidm`.

The default value for this parameter is `IidmImportExportMode.NO_SEPARATED_FILE_FOR_EXTENSIONS`. This property has been removed in v3.3.0.

iidm.export.xml.skip-extensions The `iidm.export.xml.skip-extensions` property is an optional property that defines whether the XIIDM exporter skips exporting the network extensions or not. Its default value is `false`.

This property has been deprecated since v2.4.0 before being removed in v3.3.0. Set the `iidm.export.xml.extensions` to an empty string instead.

The *IIDM (iTesla Internal Data Model)* format was designed during the iTesla project. IIDM is the internal format used in Powsybl because it is designed for running simulations.

Several exchange formats result from this internal format:

- XIIDM, which corresponds to an XML export of IIDM,
- JIIDM, which corresponds to a JSON export of IIDM,
- BIIDM, which corresponds to a binary export (this is still a beta-feature).

Below are two exports from the same network:

- one XML export (XIIDM exchange format)
- one JSON export (JIIDM exchange format)

1.3.3 XIIDM

```
<?xml version="1.0" encoding="UTF-8"?>
<iidm:network xmlns:iidm="http://www.powsybl.org/schema/iidm/1_12" id="sim1" caseDate=
↪"2013-01-15T18:45:00.000+01:00" forecastDistance="0" sourceFormat="test"␣
↪minimumValidationLevel="STEADY_STATE_HYPOTHESIS">
    <iidm:substation id="P1" country="FR" tso="RTE" geographicalTags="A">
        <iidm:voltageLevel id="VLGEN" nominalV="24.0" topologyKind="BUS_BREAKER">
            <iidm:busBreakerTopology>
                <iidm:bus id="NGEN"/>
            </iidm:busBreakerTopology>
            <iidm:generator id="GEN" energySource="OTHER" minP="-9999.99" maxP="9999.99"␣
↪voltageRegulatorOn="true" targetP="607.0" targetV="24.5" targetQ="301.0" bus="NGEN"␣
↪connectableBus="NGEN">
                <iidm:minMaxReactiveLimits minQ="-9999.99" maxQ="9999.99"/>
            </iidm:generator>
        </iidm:voltageLevel>
        <iidm:voltageLevel id="VLHV1" nominalV="380.0" topologyKind="BUS_BREAKER">
            <iidm:busBreakerTopology>
                <iidm:bus id="NHV1"/>
            </iidm:busBreakerTopology>
        </iidm:voltageLevel>
        <iidm:twoWindingsTransformer id="NGEN_NHV1" r="0.26658461538461536" x="11.
↪104492831516762" g="0.0" b="0.0" ratedU1="24.0" ratedU2="400.0" voltageLevelId1="VLGEN
↪" bus1="NGEN" connectableBus1="NGEN" voltageLevelId2="VLHV1" bus2="NHV1"␣
↪connectableBus2="NHV1"/>
    </iidm:substation>
    <iidm:substation id="P2" country="FR" tso="RTE" geographicalTags="B">
        <iidm:voltageLevel id="VLHV2" nominalV="380.0" topologyKind="BUS_BREAKER">
            <iidm:busBreakerTopology>
                <iidm:bus id="NHV2"/>
            </iidm:busBreakerTopology>
        </iidm:voltageLevel>
        <iidm:voltageLevel id="VLLOAD" nominalV="150.0" topologyKind="BUS_BREAKER">
```

(continues on next page)

(continued from previous page)

```
            <iidm:busBreakerTopology>
                <iidm:bus id="NLOAD"/>
            </iidm:busBreakerTopology>
            <iidm:load id="LOAD" loadType="UNDEFINED" p0="600.0" q0="200.0" bus="NLOAD"␣
↪connectableBus="NLOAD"/>
        </iidm:voltageLevel>
        <iidm:twoWindingsTransformer id="NHV2_NLOAD" r="0.04724999999999999" x="4.
↪049724365620455" g="0.0" b="0.0" ratedU1="400.0" ratedU2="158.0" voltageLevelId1="VLHV2
↪" bus1="NHV2" connectableBus1="NHV2" voltageLevelId2="VLLOAD" bus2="NLOAD"␣
↪connectableBus2="NLOAD">
            <iidm:ratioTapChanger regulating="true" lowTapPosition="0" tapPosition="1"␣
↪targetDeadband="0.0" loadTapChangingCapabilities="true" regulationMode="VOLTAGE"␣
↪regulationValue="158.0">
                <iidm:terminalRef id="NHV2_NLOAD" side="TWO"/>
                <iidm:step r="0.0" x="0.0" g="0.0" b="0.0" rho="0.8505666905244191"/>
                <iidm:step r="0.0" x="0.0" g="0.0" b="0.0" rho="1.0006666666666666"/>
                <iidm:step r="0.0" x="0.0" g="0.0" b="0.0" rho="1.150766642808914"/>
            </iidm:ratioTapChanger>
        </iidm:twoWindingsTransformer>
    </iidm:substation>
    <iidm:line id="NHV1_NHV2_1" r="3.0" x="33.0" g1="0.0" b1="1.93E-4" g2="0.0" b2="1.
↪93E-4" voltageLevelId1="VLHV1" bus1="NHV1" connectableBus1="NHV1" voltageLevelId2=
↪"VLHV2" bus2="NHV2" connectableBus2="NHV2"/>
    <iidm:line id="NHV1_NHV2_2" r="3.0" x="33.0" g1="0.0" b1="1.93E-4" g2="0.0" b2="1.
↪93E-4" voltageLevelId1="VLHV1" bus1="NHV1" connectableBus1="NHV1" voltageLevelId2=
↪"VLHV2" bus2="NHV2" connectableBus2="NHV2"/>
</iidm:network>
```

1.3.4 JIIDM

```
{
  "version" : "1.12",
  "id" : "sim1",
  "caseDate" : "2013-01-15T18:45:00.000+01:00",
  "forecastDistance" : 0,
  "sourceFormat" : "test",
  "minimumValidationLevel" : "STEADY_STATE_HYPOTHESIS",
  "substations" : [ {
    "id" : "P1",
    "country" : "FR",
    "tso" : "RTE",
    "geographicalTags" : [ "A" ],
    "voltageLevels" : [ {
      "id" : "VLGEN",
      "nominalV" : 24.0,
      "topologyKind" : "BUS_BREAKER",
      "busBreakerTopology" : {
        "buses" : [ {
          "id" : "NGEN"
        } ]
      },
```

(continues on next page)

(continued from previous page)


```
      "generators" : [ {
        "id" : "GEN",
        "energySource" : "OTHER",
        "minP" : -9999.99,
        "maxP" : 9999.99,
        "voltageRegulatorOn" : true,
        "targetP" : 607.0,
        "targetV" : 24.5,
        "targetQ" : 301.0,
        "bus" : "NGEN",
        "connectableBus" : "NGEN",
        "minMaxReactiveLimits" : {
          "minQ" : -9999.99,
          "maxQ" : 9999.99
        }
      } ]
    }, {
      "id" : "VLHV1",
      "nominalV" : 380.0,
      "topologyKind" : "BUS_BREAKER",
      "busBreakerTopology" : {
        "buses" : [ {
          "id" : "NHV1"
        } ]
      }
    } ],
    "twoWindingsTransformers" : [ {
      "id" : "NGEN_NHV1",
      "r" : 0.26658461538461536,
      "x" : 11.104492831516762,
      "g" : 0.0,
      "b" : 0.0,
      "ratedU1" : 24.0,
      "ratedU2" : 400.0,
      "voltageLevelId1" : "VLGEN",
      "bus1" : "NGEN",
      "connectableBus1" : "NGEN",
      "voltageLevelId2" : "VLHV1",
      "bus2" : "NHV1",
      "connectableBus2" : "NHV1"
    } ]
  }, {
    "id" : "P2",
    "country" : "FR",
    "tso" : "RTE",
    "geographicalTags" : [ "B" ],
    "voltageLevels" : [ {
      "id" : "VLHV2",
      "nominalV" : 380.0,
      "topologyKind" : "BUS_BREAKER",
      "busBreakerTopology" : {
        "buses" : [ {
```


(continues on next page)

(continued from previous page)

```
        "id" : "NHV2"
      } ]
    }
  }, {
    "id" : "VLLOAD",
    "nominalV" : 150.0,
    "topologyKind" : "BUS_BREAKER",
    "busBreakerTopology" : {
      "buses" : [ {
        "id" : "NLOAD"
      } ]
    },
    "loads" : [ {
      "id" : "LOAD",
      "loadType" : "UNDEFINED",
      "p0" : 600.0,
      "q0" : 200.0,
      "bus" : "NLOAD",
      "connectableBus" : "NLOAD"
    } ]
  } ],
  "twoWindingsTransformers" : [ {
    "id" : "NHV2_NLOAD",
    "r" : 0.04724999999999999,
    "x" : 4.049724365620455,
    "g" : 0.0,
    "b" : 0.0,
    "ratedU1" : 400.0,
    "ratedU2" : 158.0,
    "voltageLevelId1" : "VLHV2",
    "bus1" : "NHV2",
    "connectableBus1" : "NHV2",
    "voltageLevelId2" : "VLLOAD",
    "bus2" : "NLOAD",
    "connectableBus2" : "NLOAD",
    "ratioTapChanger" : {
      "regulating" : true,
      "lowTapPosition" : 0,
      "tapPosition" : 1,
      "targetDeadband" : 0.0,
      "loadTapChangingCapabilities" : true,
      "regulationMode" : "VOLTAGE",
      "regulationValue" : 158.0,
      "terminalRef" : {
        "id" : "NHV2_NLOAD",
        "side" : "TWO"
      },
      "steps" : [ {
        "r" : 0.0,
        "x" : 0.0,
        "g" : 0.0,
        "b" : 0.0,
```

(continues on next page)

(continued from previous page)


```
          "rho" : 0.8505666905244191
        }, {
          "r" : 0.0,
          "x" : 0.0,
          "g" : 0.0,
          "b" : 0.0,
          "rho" : 1.0006666666666666
        }, {
          "r" : 0.0,
          "x" : 0.0,
          "g" : 0.0,
          "b" : 0.0,
          "rho" : 1.150766642808914
        } ]
      }
    } ]
  } ],
  "lines" : [ {
    "id" : "NHV1_NHV2_1",
    "r" : 3.0,
    "x" : 33.0,
    "g1" : 0.0,
    "b1" : 1.93E-4,
    "g2" : 0.0,
    "b2" : 1.93E-4,
    "voltageLevelId1" : "VLHV1",
    "bus1" : "NHV1",
    "connectableBus1" : "NHV1",
    "voltageLevelId2" : "VLHV2",
    "bus2" : "NHV2",
    "connectableBus2" : "NHV2"
  }, {
    "id" : "NHV1_NHV2_2",
    "r" : 3.0,
    "x" : 33.0,
    "g1" : 0.0,
    "b1" : 1.93E-4,
    "g2" : 0.0,
    "b2" : 1.93E-4,
    "voltageLevelId1" : "VLHV1",
    "bus1" : "NHV1",
    "connectableBus1" : "NHV1",
    "voltageLevelId2" : "VLHV2",
    "bus2" : "NHV2",
    "connectableBus2" : "NHV2"
  } ]
}
```

1.4 IEEE CDF

TODO

1.5 PowerFactory

1.5.1 Import

PowerFactory files are available through PowerFactory application. The internal format is `.pfd`. To use and generate PowerFactory files you need to own a licence. You can follow instructions on the official PowerFactory website.

Handle PowerFactory exported format cases

PowerFactory files are encrypted (internal format `.pfd`), data are only readable by the PowerFactory application. PowSyBl core only supports PowerFactory DGS V5 and only the ASCII format (`.dgs` files) which are PowerFactory formatted files. They are created from a PowerFactory application project export.

A basic DGS export configuration is provided with the PowerFactory application in \Dgs\V5.X\DGS Export Definitions 5.pfd. This file has to be imported in your database. However, this basic configuration does not provide all necessary objects and attributes to PowSyBl converter to achieve a correct conversion of the initial data model (and get a calculated state when running a load flow close to the one calculated by PowerFactory). So you need to complete the configuration with those following steps.

Two and three winding transformer taps definition attributes need to be added to existing configuration. Optionally, voltage magnitude and angle could be added to terminals to compare the calculated state between PowerFactory and PowSyBl. VSC and common impedance configuration are totally missing and need to be created. Optionally, in order to get the same slack, the attribute `ip_ctrl` can be provided.

```
- 2-Winding Transformer.IntMon (class ElmTr2)
    - e:mTaps

- 3-Winding Transformer.IntMon (class ElmTr3)
    - e:mTaps
    - e:iMeasTap

- Terminal.IntMon (class ElmTerm)
    - m:u
    - m:phiu

- PWM Converter/2 DC-Connections.IntMon (class ElmVsc)
    - e:loc_name
    - e:fold_id
    - e:psetp
    - e:qsetp
    - e:usetp
    - e:Pnold
    - e:Unom
    - e:P_max

- Common Impedance.IntMon (class ElmZpu)
    - e:loc_name
    - e:outserv
    - e:Sn
    - e:nphshift
```

(continues on next page)

(continued from previous page)

```
    - e:ag
    - e:fold_id
    - e:r_pu
    - e:x_pu
    - e:r_pu_ji
    - e:x_pu_ji
    - e:gi_pu
    - e:bi_pu
    - e:gj_pu
    - e:bj_pu

- Synchronous Machine.IntMon (class ElmSym)
    - e:ip_ctrl
```

Handling HVDC data

Selecting reduced or detailed model

By default, the DGS file importer tries to import HVDC components as *reduced* point-to-point representations. Parameter `powerfactory.import.dgs.HVDC-import-detailed` must be set to `true` to activate import of *detailed* DC subnetworks with multi-terminal possibility.

Additional attributes (*detailed*)

For *detailed* network import, the following additional attributes are also requested in the DGS file for the VSCs (`ElmVsc`)

Attribute	Meaning
i_acdc	Control mode selector. Values 3, 4, 5 and 6 are supported.
usetpdc	Control voltage in p.u. for DC voltage control.
Unomdc	Nominal DC voltage for DC voltage control.
swtLossFactor	Swicthing loss factor (otherwise default to zero).
resLossFactor	Resistive loss factor (otherwise default to zero).

Ground elements (`ElmGndswt`) are not exported by default by PowerFactory. Their export to the DGS file must be declared specifically if grounds are to be re-imported by PowSyBl. No additional attribute is mandatory for ground elements (`ElmGndswt`). If `on_off` is present and has value zero, the ground element is considered disconnected and it is not added to the network. `ciEarthed` is disregarded by the importer. The switch itself is not imported to PowSyBl. The ground resistance is assumed to be zero.

`ElmTerm`, `ElmLne` and `TypLne` are used by the importer, but require no additional data than the default attributes.

Control mode (*detailed*)

The following type of control mode is setup for VSCs, depending on the value of `i_acdc`:

i_acdc	Behavior	ControlMode	voltageRegulatorOn
3	Vdc - Q	V_DC	false
4	P - Vac	P_PCC	true
5	P - Q	P_PCC	false
6	Vdc - Vac	V_DC	true

Values 0, 1, 2, 7 and 8 are not supported and will raise a PowerFactoryException during the import.

Limitations (*detailed*)

- The only supported AC-DC converters are VSCs in `ElmVsc`.
- For now PCC control is not taken into account and the VSC is connected to a single terminal.
- Attribute `ciEarthed` of `ElmTerm` is ignored by the importer.

Import PowerFactory internal format

A powsybl-powerfactory-db-native repository has been created to import a proprietary PowerFactory file (`.pfd`) directly via a C++ API but the import can be slower and is not maintained for now.

PowerFactory file is a licenced file from DIgSILENT used in PowerFactory, a leading power system analysis software application.

1.6 Matpower

1.6.1 Import

There are a lot of Matpower open source cases available all over the internet. Most of the time, those cases are provided as a `.m` file. This is Matlab code, and only Matlab can interpret this kind of data. PowSyBl converter can only import `.mat` file which is a binary serialization of Matlab case data structure created from `.m` file.

Matpower cases conversions

To import a Matpower cases, they have to be converted to `.mat` files first. This can be done using Matlab, but it is also possible to use Octave, an open source scientific programming language which is mostly compatible with Matlab. Matpower toolbox can be installed with Octave.

Octave and Matpower installation

To install Octave, please follow the installation guide for your operating system, available on their wiki.

Then, you can download and install Matpower toolbox, following the instructions of the getting started.

Matpower case conversion

In this example, we'll use the `case6515rte.m` case file found on Matpower GitHub.

Run Octable in the `traditional` mode, meaning the mode with the maximal compatibility with Matlab especially for binary serializations.

```
$> octave --traditional
```

To convert the `case6515rte.m` to `case6515rte.mat`, execute the two following lines:

```
mpc = loadcase('case6515rte.m');
savecase("case6515rte.mat", mpc);
```

Note that the `loadcase` and `savecase` are functions provided by the Matpower toolbox.

Importing with PowSyBl

This section assumes that you have a `.mat` file. If you have a `.m` file, please follow the *Matpower cases conversions* section above.

Dependencies

To be able to read `.mat` files, your `pom.xml` should contain the `powsybl-matpower-converter` dependency and an implementation of `powsybl-iidm-api`, for example `powsybl-iidm-impl`:

```
<dependencies>
    <dependency>
        <groupId>com.powsybl</groupId>
        <artifactId>powsybl-iidm-impl</artifactId>
    </dependency>
    <dependency>
        <groupId>com.powsybl</groupId>
        <artifactId>powsybl-matpower-converter</artifactId>
    </dependency>
</dependencies>
```

Note: This is not necessary if you are using `powsybl-starter` as it is already included. For version reference, either use PowSyBl dependencies or use the correct version following the table here

Your `pom.xml` should also contain an implementation of `powsybl-iidm-api`, for example `powsybl-iidm-impl`.

Reading the file

```
import java.nio.file.Path;
import com.powsybl.iidm.network.Network;

Path filePath = Path.of("/path/to/file.mat");
Network n = Network.read(filePath);
```

1.6.2 Export

The export of PowSyBl networks to Matpower format is not supported.

Matpower is a free and open-source Matlab toolbox for power system simulation and optimization. It is widely used by academics for research purposes.

1.7 PSS®E

1.7.1 Import

The import module reads and converts a PSS®E power flow data file to the PowSyBl grid model. The current implementation supports RAW format for versions 33 and 35 and RAWX format for version 35. The import process is performed in three steps:

- Read the input file.
- Validate input data.
- Convert input data into PowSyBl grid model.

First, input data is obtained by reading and parsing the input file, and as a result, a PSS®E model is created in memory. This model can be viewed as a set of Java classes where each data block of the PSS®E model is associated with a

specific Java class that describes all their attributes or data items. Then, some inconsistency checks are performed on this model. If the validation succeeds, the PSS®E model is converted to a PowSyBl grid model.

Options

Parameters for the import can be defined in the configuration file in the *import-export-parameters-default-value* module.

psse.import.ignore-base-voltage The `psse.import.ignore-base-voltage` property is an optional property that defines if the importer should ignore the base voltage information present in the PSS®E file. The default value is `false`.

psse.import.ignore-node-breaker-topology The `psse.import.ignore-node-breaker-topology` property is an optional property that defines if the importer should ignore the node breaker information present in the PSS®E file. The default value is `false`.

Inconsistency checks

-TODO

Conversion

A PSS®E file specifies a Bus/Branch network model where typically there is a bus for each voltage level inside a substation and where substation objects are not explicitly defined. Breakers and switches were traditionally modeled as zero impedance lines with special identifiers. Since version 35 PSS®E supports explicit definition of substation data and switching devices at both system and substation level. However, this information is optional and may not be present in the case.

The PowSyBl grid model establishes the substation as a required container of voltage levels and equipment. The first step in the conversion process assigns a substation for each PSS®E bus, ensuring that all buses at transformer ends are kept in the same substation.

The current conversion uses explicit PSS®E substation information to group buses, considering zero-impedance branches, transformers, and substation membership as connectors. A new substation is created for each identified component. Within the substation, a new voltage level is created for each group of buses connected by zero-impedance branches or sharing the same nominal voltage. The topology of each voltage level can be defined at either the bus-branch level or the node-breaker level. It is only defined at the node-breaker level when all the buses included in the voltage level belong to the same PSS®E substation. In this case, all the detailed connectivity defined in PSS®E is imported into the PowSyBl model.

In the PowSyBl grid model, all the network components are identified through a global and unique alphanumeric identifier (**Id**). Optionally, they may receive a name (**Name**).

For each substation, the following attributes are defined:

- **Id** following one of the two patterns: `S<n>`, when the substation corresponds to a PSS®E substation (where `n` is the identifier of the PSS®E substation), and `S<n1-n2 ... -ni>`, when the substation represents a bus or a group of buses, where `n1...ni` are the identifiers of the buses sorted in ascending order.

Every voltage level is assigned to its corresponding substation, with attributes:

- **Id** following the pattern `VL<n1-n2 ... -ni>` where `n1 .. ni` represents the PSS®E bus numbers included inside the voltage level sorted in ascending order.
- **NominalV** Nominal voltage of the voltage level. Equal to 1 if `psse.import.ignore-base-voltage` property is `true`. Otherwise, it is assigned to the base voltage of one representative bus inside the voltage level, read from PSS®E field `BASKV`.
- **TopologyKind** Topology level assigned to the network model, `NODE_BREAKER` or `BUS_BREAKER`.

The following sections describe in detail how each supported PSS®E data block is converted to PowSyBl network model objects.

Bus Data

There is a one-to-one correspondence between the records of the PSS®E *Bus Data* block and the buses of the PowSyBl network model. For each record in the *Bus Data* block a PowSyBl bus is created and assigned to its corresponding voltage level with the following attributes:

- **Id** according to the pattern `B<n>` where `n` represents the PSS®E bus number (field `I` in the *Bus Data* record).
- **Name** is copied from PSS®E field `NAME`.
- **V** is obtained from the PSS®E bus voltage magnitude, `VM`, multiplied by the nominal voltage of the corresponding voltage level.
- **Angle** is copied from the PSS®E bus voltage phase angle, `VA`.

Load Data

Every *Load Data* record represents one load. Multiple loads are allowed at the same bus by specifying one *Load Data* record with the same bus and different load identifiers. Each record defines a new load in the PowSyBl grid model associated with its corresponding voltage level and with the following attributes:

- **Id** according to the pattern `<n>-L<m>` where `n` represents the PSS®E bus number (field `I` in the *Load Data* record) and `m` is the PSS®E alphanumeric load identifier (field `ID` in the *Load Data* record).
- **ConnectableBus** PowSyBl bus identifier assigned to the PSS®E bus number (field `I` in the *Load Data* record).
- **P0** Active power. It is copied from PSS®E field `PL`.
- **Q0** Reactive power. It is copied from PSS®E field `QL`.

The load is connected to the ConnectableBus if load status (field `STATUS` in the *Load Data* record) is `1` (In-service).

PSS®E supports loads with three different characteristics: Constant Power, Constant Current and Constant Admittance. The current version only takes into consideration the Constant Power component, discarding the Constant Current component (fields `IP` and `IQ` in the *Load Data* record) and the Constant Admittance component (fields `YP` and `YQ` in the *Load Data* record).

Fixed Bus Shunt Data

Each *Fixed Bus Shunt Data* record defines a PowSyBl shunt compensator with a linear model and a single section. It is possible to define multiple fixed shunts at the same bus. The PowSyBl shunt compensator is associated with its corresponding voltage level and has the following attributes:

- **Id** according to the pattern `<n>-SH<m>` where `n` represents the PSS®E bus number (field `I` in the *Fixed Bus Shunt Data* record) and `m` is the PSS®E alphanumeric shunt identifier (field `ID` in the *Fixed Bus Shunt Data* record).
- **ConnectableBus** PowSyBl bus identifier assigned to the PSS®E bus number (field `I` in the *Fixed Bus Shunt Data* record).
- **SectionCount** Always 1.
- **gPerSection** Positive sequence shunt (charging) conductance per section. It is defined as `GL / (vnom *vnom)`, where `GL` is the active component of shunt admittance to ground, entered in MW at one per unit voltage (field `GL` in the *Fixed Bus Shunt Data* record) and `vnom` is the nominal voltage of the corresponding voltage level.
- **bPerSection** Positive sequence shunt (charging) susceptance per section. It is defined as `BL / (vnom *vnom)`, where `BL` is the reactive component of shunt admittance to ground, entered in MVAR at one per-unit voltage (field `BL` in the *Fixed Bus Shunt Data* record).
- **MaximumSectionCount** Always 1.

The shunt compensator is connected to the ConnectableBus if fixed shunt status (field `STATUS` in the *Fixed Bus Shunt Data* record) is 1 (In-service).

Switched Shunt Data

In the PSS®E version 33, only one switched shunt element can be defined on each bus. Version 35 allows multiple switched shunts on the same bus, adding an alphanumeric switched shunt identifier.

A switched shunt device may be a mix of reactors and capacitors; it is divided in blocks and steps: a block contains `n` steps of the same admittance. The steps and blocks can be adjusted to regulate a given magnitude: a voltage at a bus, a reactive power output or an admittance. Only voltage regulation is considered when mapping this equipment to PowSyBl.

There are two methods for defining how a switched shunt steps and blocks should be adjusted to keep the controlled magnitude between limits; the method chosen is determined by the field `ADJM` in the *Switched Shunt Data* record. If `ADJM` is `0` steps and blocks are switched on in input order, and switched off in reverse output order. If `ADJM` is `1`, steps and blocks are switched on and off such that the next highest (or lowest, as appropriate) total admittance is achieved.

Each switched shunt record defines a PowSyBl shunt compensator with a non-linear model with the following attributes:

- **Id** according to the pattern `<n>-SwSH<m>` where `n` represents the PSS®E bus number (field `I` in the *Switched Shunt Data* record) and `m` is the PSS®E alphanumeric shunt identifier (field `ID` in version 35 of the *Switched Shunt Data* record, forced to `"1"` when importing version 33 data).
- **ConnectableBus** PowSyBl bus identifier assigned to the PSS®E bus number (field `I` in the *Switched Shunt Data* record).
- **SectionCount** Defined as the section count (section index + 1) where section `B` is closer to the initial switched shunt admittance (field `BINIT` in the *Switched Shunt Data* record).
- **TargetV** Voltage setpoint defined as `0.5` * (`VSWLO` + `VSWHI`) * `vnom`, where `VSWLO` is the controlled voltage lower limit (field `VSWLO` in the *Switched Shunt Data* record) and `VSWHI` is the controlled voltage upper limit (field `VSWHI` in the *Switched Shunt Data* record).
- **TargetDeadband** defined as (`VSWHI` - `VSWLO`) * `vnom`.
- **RegulatingTerminal** Regulating terminal assigned to the bus where voltage is controlled by this switched shunt (field `SWREM` in version 33 or field `SWREG` in version 35, both in the *Switched Shunt Data* record if they are not `0`. Otherwise, field `I` in the *Switched Shunt Data* record).
- **VoltageRegulatorOn** defined as `true` if the control mode is not `0` (field `MODSW` in the *Switched Shunt Data* record) and `TargetV` is greater than `0.0`.

The shunt compensator is connected to the ConnectableBus if switched shunt status (field `STAT` in the *Switched Shunt Data* record) is `1` (In-service).

The sections of the PowSyBl shunt compensator non-linear model are defined according to the adjustment method of the PSS®E switched shunt. In the PowSyBl model, the susceptance at each section is the accumulated susceptance obtained if the section and all previous ones are connected.

The attributes of each section in the PowSyBl shunt compensator non-linear model are defined as:

- **Section G** Positive sequence shunt (charging) conductance of this section. Always `0.0`.
- **Section B** Positive sequence shunt (charging) susceptance of this section. It is defined as `B` / (`vnom` *`vnom`), where `B` is the reactive component of shunt admittance to ground, entered in MVAR at one per-unit voltage assigned to this section and `vnom` is the nominal voltage of the corresponding voltage level.

When the adjustment method `ADJM` is `0`, the behaviour of the switched shunt can be mapped directly to the shunt compensator non-linear model with sections based on the switched shunt blocks/steps and its order in the PSS@E input record. A section is assigned to each step of the reactor and capacitor shunt blocks by accumulating the admittance of the corresponding steps that are in-service. Only the in-service switched shunt blocks are considered (field `SI` in version 35 of the *Switched Shunt Data* record, always in-service in version 33). A section with 0.0 susceptance is added between sections assigned to reactor and capacitor blocks.

If the adjustment method ADJM is 1, the reactor and capacitor blocks can be specified at any order, and all the switching combinations are considered in PSS®E. Current conversion does not support building a separate section for each switching combination. To map the PSS@E shunt blocks/steps into PowSyBl sections, first the reactor and capacitor blocks are increasingly ordered by susceptance (field BI in the *Switched Shunt Data* record) and then sections are created like in the previous adjustment considering that blocks are switched on following the sorted order.

Generator Data

Every *Generator Data* single line record represents one generator. Multiple generators are allowed at a PSS®E bus by specifying the same bus and a different identifier. Each record defines a new generator in the PowSyBl grid model associated with its corresponding voltage level and with the following attributes:

- **Id** according to the pattern <n>-G<m> where n represents the PSS®E bus number (field I in the *Generator Data* record) and m is the PSS®E alphanumeric load identifier (field ID in the *Generator Data* record).
- **ConnectableBus** PowSyBl bus identifier assigned to the PSS®E bus number (field I in the *Generator Data* record).
- **TargetP** Active power. It is copied from PSS®E field PG.
- **MinP** Minimum generator active power. It is copied from PSS®E field PB.
- **MaxP** Maximum generator active power. It is copied from PSS®E field PT.
- **TargetQ** Reactive power. It is copied from PSS®E field QG.
- **MinQ** Minimum generator reactive power. It is copied from PSS®E field QB.
- **MaxQ** Maximum generator reactive power. It is copied from PSS®E field QT.
- **TargetV** Voltage setpoint defined as VS * vnom, where VS is the regulated voltage (field VS in the *Generator Data* record) and vnom is the nominal voltage of the corresponding voltage level.
- **RegulatingTerminal** Regulating terminal assigned to the bus where voltage is controlled by this generator (field IREG in the *Generator Data* record if it is not 0. Otherwise, field I in the *Generator Data* record).
- **VoltageRegulatorOn** defined as true if the type code of the associated bus is 2 or 3 (field IDE in the *Bus Data* record) and TargetV is greater than 0.0 and MaxQ is greater than MinQ.

The generator is connected to the ConnectableBus if generator status (field STAT in the *Generator Data* record) is 1 (In-service).

Non-Transformer Branch Data

In PSS®E each AC transmission line is represented as a non-transformer branch record and defines a new line in the PowSyBl grid model with the following attributes:

- **Id** according to the pattern L-<n>-<m>-<p> where n represents the PSS®E bus 1 number (field I in the *Non-Transformer Branch Data* record), m represents the bus 2 number (field J in the *Non-Transformer Branch Data* record) and p is the circuit identifier (field CKT in the *Non-Transformer Branch Data* record).
- **ConnectableBus1** PowSyBl bus identifier assigned to the PSS®E bus 1 number (field I in the *Non-Transformer Branch Data* record).
- **VoltageLevel1** PowSyBl voltage level assigned to the bus 1.
- **ConnectableBus2** PowSyBl bus identifier assigned to the PSS®E bus 2 number (field J in the *Non-Transformer Branch Data* record).
- **VoltageLevel2** PowSyBl voltage level assigned to the bus 2.

- **R** Resistance defined as R * vnom2 * vnom2 / sbase where R is the resistance of the branch (field R in the *Non-Transformer Branch Data* record), vnom2 is the nominal voltage of the voltage level assigned to the bus 2 and sbase is the system MVA base (field SBASE in the *Case Identification Data* record).
- **X** Reactance defined as X * vnom2 * vnom2 / sbase where X is the reactance of the branch (field X in the *Non-Transformer Branch Data* record).
- **G1** Conductance of the line shunt at the bus 1 defined as (GI * sbase) / (vnom2 * vnom2) where GI is the conductance at the bus 1 (field GI in the *Non-Transformer Branch Data* record).
- **B1** Susceptance defined as ((0.5 * B + BI) * sbase) / (vnom2 * vnom2) where B is the total branch charging susceptance (field B in the *Non-Transformer Branch Data* record) and BI is the susceptance at the bus 1 (field BI in the *Non-Transformer Branch Data* record).
- **G2** Conductance of the line shunt at the bus 2 defined as (GJ * sbase) / (vnom2 * vnom2) where GJ is the conductance at the bus 2 (field GJ in the *Non-Transformer Branch Data* record).
- **B2** Susceptance defined as ((0.5 * B + BJ) * sbase) / (vnom2 * vnom2) where B is the total branch charging susceptance (field B in the *Non-Transformer Branch Data* record) and BJ is the susceptance at the bus 2 (field BJ in the *Non-Transformer Branch Data* record).

The line is connected at both ends if the branch status (field ST in the *Non-Transformer Branch Data* record) is 1 (In-service).

A set of current permanent limits is defined as 1000.0 * rateMva / (sqrt(3.0) * vnom1) at the end 1 and 1000.0 * rateMva / (sqrt(3.0) * vnom2) at the end 2 where rateMva is the first rating of the branch (field RATEA in version 33 of the *Non-Transformer Branch Data* record and field RATE1 in version 35) and vnom1 and vnom2 are the nominal voltages of the associated voltage levels.

Transformer Data

The *Transformer Data* block defines two- and three-winding transformers. Two-winding transformers have four line records, while three-winding transformers have five line records. A 0 value in the field K of the *Transformer Data* record first line is used to indicate that is a two-winding transformer, otherwise is considered a three-winding transformer.

PSS@E two-winding transformer records are mapped to two-winding transformers in the PowSyBl grid model. They are associated with corresponding voltage levels inside the same substation and defined with the following attributes:

- **Id** according to the pattern T-<n>-<m>-<p> where n represents the PSS®E bus 1 number (field I in the *Transformer Data* record), m represents the bus 2 number (field J in the *Transformer Data* record) and p is the circuit identifier (field CKT in the *Transformer Data* record).
- **ConnectableBus1** PowSyBl bus identifier assigned to the PSS®E bus 1 number (field I in the *Transformer Data* record).
- **VoltageLevel1** PowSyBl voltage level assigned to the bus 1.
- **ConnectableBus2** PowSyBl bus identifier assigned to the PSS®E bus 2 number (field J in the *Transformer Data* record).
- **VoltageLevel2** PowSyBl voltage level assigned to the bus 2.
- **RatedU1** Rated voltage at the end 1. The nominal voltage of the associated voltageLevel is assigned.
- **RatedU2** Rated voltage at the end 2. The nominal voltage of the associated voltageLevel is assigned.
- **R** Transmission resistance.
- **X** Transmission reactance.
- **G** Shunt conductance.
- **B** Shunt susceptance.

- **TapChanger** Could be a ratio tap changer or a phase tap changer.
- **OperationalLimits** Current limits for both ends.

The transformer is connected at both ends if the branch status (field STAT in the *Transformer Data* record) is 1 (In-service)

In PSS®E the transformer model allows to define a ratio and angle at the end 1 and only a fixed ratio at the end 2. The transformer magnetizing admittance is modeled between the bus and the ratio of the end 1. The PowSyBl grid model supports a ratioTapChanger and a phaseTapChanger at the end 1 and the magnetizing admittance is between the ratio and the transmission impedance.

To express the PSS®E electric attributes of the transformer in the PowSyBl grid model, the following conversions are performed:

- The first step is to define the complex impedance between windings (Z) by using the resistance and reactance (fields R1-2 and X1-2 in the *Transformer Data* record), the winding base MVA (field SBASE1-2 in the *Transformer Data* record) and the system MVA base (field SBASE in the *Case Identification Data* record) according to the code that defines the units in which the winding impedances R1-2, X1-2 are specified (field CZ in the *Transformer Data* record). Then the complex impedance (Z) is converted to engineering units using the nominal voltage of the voltage level at end 2 and the system MVA base. Finally, it should be adjusted after fixing an ideal ratio at end 2 and moving the configured ratio to the end 1. The obtained result is assigned to the transmission resistance and reactance of the PowSyBl transformer.
- The complex shunt admittance Ysh is calculated using the transformer magnetizing admittance connected to ground at bus 1 (fields MAG1 and MAG2 in the *Transformer Data* record), the winding base MVA (field SBASE1-2 in the *Transformer Data* record), the system MVA base, the bus base voltage (field BASKV in the *Bus Data* record) of the transformer bus I and the nominal (rated) winding 1 voltage base (field NOMV1 in the *Transformer Data* record) according to the magnetizing admittance code that defines the units in which MAG1 and MAG2 are specified (filed CM in the *Transformer Data* record). The next step is to convert the complex Ysh to engineering units using the nominal voltage of the voltage level at end 2 and the system MVA base and finally the obtained value is assigned to the shunt conductance and susceptance of the PowSyBl transformer. The shunt conductance in the PowSyBl grid model is located after the ratio, so is necessary to add a step correction by each different ratio in the tabular tapChanger.

To define the tapChanger the first step is to calculate the complex ratio at end 1 and the ratio at end 2. The ratio at end 1 is calculated using the winding ratio (field WINDV1 in the *Transformer Data* record), the nominal (rated) winding voltage base (field NOMV1 in the *Transformer Data* record) and the bus base voltage (field BASKV in the *Bus Data* record) according to the code that defines the units in which the turns ratios are specified (field CZ in the *Transformer Data* record). The angle at end 1 is copied from the winding phase shift angle (field ANG1 in the *Transformer Data* record). The ratio at end 2 is calculated in the same way as at end 1 but using the following fields (fields WINDV2, NOMV1 in the *Transformer Data* record) and the corresponding bus base voltage at bus J (field BASKV in the *Bus Data* record). Then a tapChanger at end 1 is defined by fixing one of the components of the complex ratio at end 1 and moving the other in each step using the number of tap positions available (field NTP1 in the *Transformer Data* record) and the upper and lower limits (fields RMA1, RMI1 in the *Transformer Data* record). Finally, the tapChanger is adjusted twice, after fixing an ideal ratio at end 2 by moving the current ratio to end 1 and after moving the shunt admittance adding in this last case a step correction for each step of the tapChanger. The tap for which the (ratio, angle) is closer to the complex ratio at end 1 is assigned as tapPosition. This current version does not consider the impedance correction table if this transformer impedance is to be a function of either off-nominal turns ratio or phase shift angle.

If the tapChanger is a ratioTapChanger and the transformer control mode (field COD in the *Transformer Data* record) is 1 a voltage control with the following attributes is defined:

- **TargetV** Voltage setpoint defined as 0.5 * (VMI + VMA) * vnom, where VMA and VMI are the voltage upper and lower limits (fields VMA1, VMA1 in the *Transformer Data* record) and vnom is the nominal voltage of the voltageLevel at end 1.

- **TargetDeadband** defined as (`VMA` - `VMI`) * `vnom`.
- **RegulatingTerminal** Regulating terminal assigned to the bus where voltage is controlled (field `CONT1` in the *Transformer Data* record).
- **RegulatingOn** defined as `true` if `TargetV` and `TargetDeadBand` are greater than `0.0`.

If the PSS®E transformer is controlling the reactive power (field `COD = 2` in the *Transformer Data* record) the control is discarded as the current version of PowSyBl does not support reactive control for transformers.

When the `tapChanger` is a `phaseTapChanger` and the transformer control mode (field `COD` in the *Transformer Data* record) is `3` an active power control with the following attributes is defined:

- **TargetValue** Active power flow setpoint defined as `0.5` * (`APFI` + `APFA`), where `APFA` and `APFI` are the active power flow upper and lower limits (fields `VMA1`, `VMA1` in the *Transformer Data* record, same fields as voltage control).
- **TargetDeadband** defined as (`APLA` - `APFI`).
- **RegulatingTerminal** Regulating terminal assigned to the bus where active power flow is controlled (field `CONT1` in the *Transformer Data* record).
- **RegulatingOn** defined as `true` if `TargetV` is greater than `0.0`.

A set of current operational limits is defined for the two-winding transformer as `1000.0` * `rateMva` / (`sqrt(3.0)` * `vnom1`) at the end `1` and `1000.0` * `rateMva` / (`sqrt(3.0)` * `vnom2`) at the end `2` where `rateMva` is the first rating of the transformer (field `RATA1` in version 33 of the *Transformer Data* record and field `RATE11` in version 35) and `vnom1` and `vnom2` are the nominal voltage of the associated voltage levels.

When a three-winding transformer is modeled, the two-winding transformer steps should be followed for each leg of the transformer taking into account the following considerations:

- The **Id** is defined according to the pattern `T-<n>-<m>-<o>-<p>` where `n` represents the PSS®E bus `1` number (field `I` in the *Transformer Data* record), `m` represents the bus `2` number (field `J` in the *Transformer Data* record), `o` represents the bus `3` number (field `K` in the *Transformer Data* record) and `p` is the circuit identifier (field `CKT` in the *Transformer Data* record).
- The three-winding transformers are modeled in PowSyBl as three two-winding transformers connected to a fictitious bus defined with a nominal base voltage and rated voltage of `1.0 kV` (star configuration).
- In PSS®E the between windings transmission impedances `Z1-2`, `Z2-3` and `Z3-1` are specified in the input file. These impedances are generally supplied on a transformer data sheet or test report. The transmission impedances `Z1`, `Z2` and `Z3` of the star network equivalent model are related to them according to the following expressions (see Modeling of Three-Winding Voltage Regulating Transformers for Positive Sequence Load Flow Analysis in PSS®E):

  ```
  Z1-2 = Z1 + Z2

  Z2-3 = Z2 + Z3

  Z3-1 = Z3 + Z1
  ```

  So:

  ```
  Z1 = 0.5 * (Z1-2 + Z3-1 - Z2-3)

  Z2 = 0.5 * (Z1-2 + Z2-3 - Z3-1)

  Z3 = 0.5 * (Z2-3 + Z3-1 - Z1-2)
  ```

- All the shunt admittances of the three-winding transformers in the PSS®E model are assigned to the winding `1`. There is no shunt admittance at windings `2` and `3`.
- Each winding can have a complex ratio and a `ratioTapChanger` or `phaseTapChanger` with its corresponding control, always at end `1`. The current PowSyBl version only supports one enabled control by three-winding

transformers so if there is more than one enabled only the first (winding 1, winding 2, winding 3) is kept enabled, the rest are automatically disabled.

- In three-winding transformers the status attribute (field STAT in the *Transformer Data* record) could be 0 that means all the windings disconnected, 1 for all windings connected, 2 for only the second winding disconnected, 3 for the third winding disconnected and 4 for the first winding disconnected.

Slack bus

The buses defined as slack terminal are the buses with type code 3 (field IDE in the *Bus Data* record).

Area data

Every *Area interchange* single line record represents one area in PowSyBl grid model with the following attributes:

- **AreaType** - always set to "ControlArea"
- **InterchangeTarget** - target active power interchange. It is copied from PSS®E field PDES
- **Name** - area name. It is copied from PSS®E field ARNAME
- **VoltageLevels** - a set of voltage levels of the area. The set is created from VoltageLevels objects that correspond to PSS®E buses within the given area.
- **AreaBoundaries** – a list of area boundaries. Boundary terminals are determined from the boundary (AC) lines, HVDC lines and transformers. The boundary lines or boundary transformers are always identified such that one of their terminals belongs to the area. This terminal is marked as a boundary terminal and included to area boundaries. Similarly, three-winding transformers are considered boundary transformers only if some, but not all, of their terminals belong to the area.

1.7.2 Export

There are two main use-cases supported:

- Update.
- Full export.

The update option is only valid if the psse.export.update option is set to true and the case was previously imported using the same format. This option preserves the version of the initial case and the format if the case was imported using PSS®E version 35. The update creates a copy of the initial case and modifies it by updating the relevant values in each PSS®E data block.

The full export option creates a new PSS®E version 35 case from scratch, using only the information available in the PowSyBl model. By default, the case is exported in raw format, but the rawx format can be selected by setting the psse.export.raw-format option to false. The detailed connectivity is also exported, with each substation in the PowSyBl model corresponding to a substation in PSS®E. A voltage level is exported as node-breaker when the topologyKind is NODE_BREAKER, there is at least one switch, and the number of buses within the voltage level is less than 999.

Options

Parameters for the export can be defined in the configuration file in the *import-export-parameters-default-value* module.

psse.export.update The psse.export.update property is optional and defines whether the exporter should perform an update. The default value is true.

psse.export.raw-format The psse.export.raw-format property is optional and defines whether the exporter should use the raw format. The default value is true.

1.7.3 Examples

A minimum network model is included as an example in version 35 of both formats RAW and RAWX.

```
 0,      100.0, 35, 0, 0, 60.00       / October 27, 2020 18:37:53
 PSS(R)E Minimum RAW Case

0 / END OF SYSTEM-WIDE DATA, BEGIN BUS DATA
    1,'Slack-Bus   ', 138.0000,3
    2,'Load-Bus    ', 138.0000,1
0 / END OF BUS DATA, BEGIN LOAD DATA
    2,'1 ',1,,,   40.000,    15.000
0 / END OF LOAD DATA, BEGIN FIXED SHUNT DATA
0 / END OF FIXED SHUNT DATA, BEGIN GENERATOR DATA
    1,'1 ',   40.350,   10.870
0 / END OF GENERATOR DATA, BEGIN BRANCH DATA
    1,     2,'1 ', 0.01938, 0.05917,0.05280
0 / END OF BRANCH DATA, BEGIN SYSTEM SWITCHING DEVICE DATA
0 / END OF SYSTEM SWITCHING DEVICE DATA, BEGIN TRANSFORMER DATA
0 / END OF TRANSFORMER DATA, BEGIN AREA DATA
0 / END OF AREA DATA, BEGIN TWO-TERMINAL DC DATA
0 / END OF TWO-TERMINAL DC DATA, BEGIN VOLTAGE SOURCE CONVERTER DATA
0 / END OF VOLTAGE SOURCE CONVERTER DATA, BEGIN IMPEDANCE CORRECTION DATA
0 / END OF IMPEDANCE CORRECTION DATA, BEGIN MULTI-TERMINAL DC DATA
0 / END OF MULTI-TERMINAL DC DATA, BEGIN MULTI-SECTION LINE DATA
0 / END OF MULTI-SECTION LINE DATA, BEGIN ZONE DATA
0 / END OF ZONE DATA, BEGIN INTER-AREA TRANSFER DATA
0 / END OF INTER-AREA TRANSFER DATA, BEGIN OWNER DATA
0 / END OF OWNER DATA, BEGIN FACTS CONTROL DEVICE DATA
0 / END OF FACTS CONTROL DEVICE DATA, BEGIN SWITCHED SHUNT DATA
0 / END OF SWITCHED SHUNT DATA, BEGIN GNE DEVICE DATA
0 / END OF GNE DEVICE DATA, BEGIN INDUCTION MACHINE DATA
0 / END OF INDUCTION MACHINE DATA, BEGIN SUBSTATION DATA
0 / END OF SUBSTATION DATA
Q
```

```
{
     "network":{
         "caseid":{
             "fields":["ic", "sbase", "rev", "xfrrat", "nxfrat", "basfrq", "title1"],
             "data":[0, 100.00, 35, 0, 0, 60.00, "PSS(R)E Minimum RAWX Case"]
         },
         "bus":{
             "fields":["ibus", "name", "baskv", "ide"],
             "data":[
                 [1, "Slack-Bus", 138.0, 3],
                 [2, "Load-Bus", 138.0 1]
             ]
         },
         "load":{
             "fields":["ibus", "loadid", "stat", "pl", "ql"],
             "data":[
                 [2, "1", 1, 40.0, 15.0]
             ]
```

(continues on next page)

(continued from previous page)

```
        },
        "generator":{
            "fields":["ibus", "machid", "pg", "qg"],
            "data":[
                [1, "1", "40.35", "10.87"]
            ]
        },
        "acline":{
            "fields":["ibus", "jbus", "ckt", "rpu", "xpu", "bpu"],
            "data":[
                [1, 2, "1", 0.01938, 0.05917, 0.05280]
            ]
        }
    }
}
```

PSS®E software from Siemens provides analysis functions for power system networks in steady-state and dynamic conditions. PSS®E uses different types of files to exchange data about the network. One of them is the RAW file (power flow data file). A PSS®E RAW file contains a collection of unprocessed data that specifies a Bus/Branch network model for the establishment of a power flow working case.

The RAW file has multiple groups of records (data blocks), with each group containing a particular type of data needed in power flow. The last record of each data block is a record specifying a value of zero to indicate the end of the category.

Each record in a data block contains a set of data items separated by a comma or one or more blanks where alphanumeric attributes must be enclosed in single quotes. As many of the data items specified in the RAW file have a default value, only the specific information needed should be defined in the record.

In PSS®E version 35, a new RAWX file format (Extensible Power Flow Data File) based on JSON has been introduced. It will be the standard text-based data format for PSS®E power flow data exchange. The RAWX files contain two types of data objects: Parameter Sets and Data Tables. A Parameter Set has an array with field names and a single array with field values. A Data Table has an array with field names an and array of records, each record being an array of field values. The field names array indicates the order and subset of fields for which data is provided in the data arrays.

Since PSS®E version 34, bus-branch data can be integrated with node-breaker data extensions, enabling a more detailed representation of substation connectivity.

1.8 AMPL

1.8.1 Import

The import of PowSyBl networks to AMPL format is not supported.

1.8.2 Export

Available exporters

According to one's needs, there are several AMPL exporters available, each with a version number.

At the moment, there are:

- The `BasicAmplExporter` (associated with the `AmplExportVersion` `V1_0`);
- The `ExtendedAmplExporter` (associated with the `AmplExportVersion` `V1_1`) that inherits from the `BasicAmplExporter`.

- The `ExtendedAmplExporterV2` (associated with the `AmplExportVersion V1_2`) that inherits from the `ExtendedAmplExporter`.

The default version is the `V1_2`.

Exporters define the information written in text files and fed to AMPL regarding:

- Buses;
- Tap changers;
- Branches;
- Current limits;
- Generators;
- Batteries;
- Loads;
- Shunts;
- Static VAR Compensators;
- Substations;
- VSC Converter stations;
- LCC Converter stations;
- HVDC lines.

The `BasicAmplExporter`

This exporter is the "historical" version, the first that has been designed.

The `ExtendedAmplExporter`

This exporter adds the following information to the `BasicAmplExporter`:

- In the bus tables, a boolean indicating if the bus is a slack one and an integer identifying the synchronous component;
- `r`, `g` and `b` in tap tables as it is already done for `x`;
- The regulating bus id for generators and static VAR compensators that are in voltage regulation mode.

The `ExtendedAmplExporterV2`

This exporter adds the following information to the `ExtendedAmplExporter`:

- In the generator tables, a boolean indicating if the generator is a condenser;
- In LCC converter station tables, the load target Q of the converter station;
- In HVDC line tables, the AC emulation parameters, along with a boolean to indicate whether AC emulation is active.

This exporter also corrects the unit of the load target Q (MVar) in the battery tables.

Options

These properties can be defined in the configuration file in the *import-export-parameters-default-value* module.

iidm.export.ampl.export-ratio-tap-changer-voltage-target The `iidm.export.ampl.export-ratio-tap-changer-voltage-target` property is an optional property that defines whether the AMPL exporter exports the ratio tap changer voltage setpoint or not. Its default value is `false`.

Deprecated properties

iidm.export.ampl.exportRatioTapChangerVoltageTarget The `iidm.export.ampl.exportRatioTapChangerVoltageTarget` property is deprecated since V2.4.0. Use the `iidm.export.ampl.export-ratio-tap-changer-voltage-target` property instead.

1.8.3 Example

```
#Branches
#"variant" "num" "bus1" "bus2" "p1 (MW)" "p2 (MW)" "q1 (MVar)" "q2 (MVar)"
1 1 -1 -1 -100.000 -200.000 -110.000 -120.000
1 4 -1 -1 -100.000 -200.000 -110.000 -120.000
```

The AMPL (**A** **M**athematical **P**rogramming **L**anguage) format is an algebraic modeling language to describe and solve high-complexity problems for large-scale mathematical computing (i.e. large-scale optimization and scheduling-type problems).

IIDM networks can be converted in flat text files easy to read in AMPL, each of which describing one type of the network equipment (batteries, generators, loads, branches, buses, etc.).

1.9 Going further

In this section, you'll discover some advanced features related to grid exchange formats and how to use them.

1.9.1 Datasources

Principles

Datasources are Java-objects used for I/O operations around PowSyBl. Datasources allow users to read and write files. It is for example used under the hood by Importers to access the filesystem during Network imports when using `Network.read()` methods.

For importers and exporters, datasources are used to access files corresponding to a single network.

Types of datasources

Multiple types of datasources exist, depending on whether it shall be writable or not, the kind of storage used, data location, data compression, etc.

Read-Only DataSource

`ReadOnlyDataSource` is the most basic datasource interface available. As you can tell by the name, it only provides reading features. It has two parameters:

- a base name, which is a prefix that can be used to consider only files with names starting with this prefix (while reading) or as a prefix for the output file name (while writing),
- (optionally) a data extension, mainly used to disambiguate identically named data of different type. Note: this does not apply to compression extensions.

Example: *For a file named* `europe.west.xiidm`*, the base name could be* `europe.west` *for instance (or* `europe` *or* `europe.w` *or . . .), while the data extension would be* `xiidm`*.*

The main methods `ReadOnlyDataSource` provides are:

- `exists(String fileName)` and `exists(String suffix, String ext)` to check if a file exists in the datasource
- `newInputStream(String fileName)` and `newInputStream(String suffix, String ext)` to read a file from the datasource
- `listNames(String regex)` to list the files in the datasource whose names match the regex

The methods with `String suffix, String ext` as parameters look for a file which name will be constructed as `<basename><suffix>.<ext>`.

The classes inheriting directly `ReadOnlyDataSource` are:

- `ResourceDataSource`: datasource based on a list of java classpath resources
- `ReadOnlyMemDataSource`: datasource where data is stored in a `Map<filename, data as bytes>` in memory
- `MultipleReadOnlyDataSource`: datasource grouping multiple user-defined datasources
- `GenericReadOnlyDataSource`: datasource used to read data from any known compressed format

DataSource

The `DataSource` interface extends `ReadOnlyDataSource` by adding writing features through the methods `newOutputStream(String fileName, boolean append)` and `newOutputStream(String suffix, String ext, boolean append)`. Those methods allow the user to write in a new file (if `append==false`) or at the end of an existing one (if `append==true`).

This interface also provides two static convenience methods (`fromPath(Path file)` and `fromPath(Path directory, String fileNameOrBaseName)`) for the different use cases like reading data from the local filesystem, and ensuring that the target file exists. These methods have their opposite in the class `Exporters` named `createDataSource(Path file)` and used to write data on the local filesystem, while ensuring that the target file given as parameter is not a directory. All those methods then make use of `DataSourceUtil.createDataSource` to build the datasource.

Two classes implement the `DataSource` interface:

- `MemDataSource`: extension of `ReadOnlyMemDataSource` implementing the writing features of `DataSource`
- `AbstractFileSystemDataSource`: abstract class used to define datasources based on files present in the file system, either directly (see below the DirectoryDataSource class and its children) or in an archive (see below the AbstractArchiveDataSource class and its children).

Directory DataSource

`DirectoryDataSource` are datasources based on files located in a specific directory directly in the file system.

Files stored and used via this type of datasource may be all compressed or not at all. Compression formats available are defined in the class `CompressionFormat`. As of today, the following single-file compressions are available: BZIP2, GZIP, XZ and ZSTD. Each one of those compression format has a corresponding datasource class inheriting `DirectoryDataSource`: `Bzip2DirectoryDataSource`, `GzDirectoryDataSource`, `XZDirectoryDataSource`, `ZstdDirectoryDataSource`.

`DirectoryDataSource` integrates the notions of base name and data extension:

- The base name is used to access files that all start with the same prefix. For example, `network` would be a good base name if your files are `network.xiidm`, `network_mapping.csv`, etc.
- The data extension is the last extension of your main files, excluding the compression extension if they have one. It usually corresponds to the data format extension: `csv`, `xml`, `json`, `xiidm`, etc. This extension is used to disambiguate the files to use in the datasource: just like you can create two different datasources selecting a different subset of files in a folder based on a different base name (e.g. `france.xiidm` and `europe.xiidm`), you can use the data extension to select either `france.xiidm` or `france.uct`.

Even if `DirectoryDataSource` integrates the notions of base name and data extension in the methods with `(String suffix, String ext)` as parameters, you still have the possibility to use files that do not correspond to the base name and data extension by using the methods with `(String filename)` as parameter, excluding the compression extension if there is one.

Note that in directory datasources, there are two behaviours for the method `listNames(String regex)`, which is used to filter the files within the datasource. In addition to the filtering using the regex parameter, it can also filter filenames to only keep those starting with the base name. This behaviour basename-filtering only occurs if `allFiles` is set to `false` in the datasource. Setting `allFiles` to `true` is for example very useful for the CGMES use case, in which a network is defined by several files (TP, EQ, SV, SSH, GL, ...), often without a common prefix (base name). This makes the `DirectoryDataSource` behave just like an `ArchiveDataSource`. Such a directory datasource is created if the user gives an existing folder as path in either one of the `Network::read` or one of the `Network::write` methods.

Archive DataSource

`AbstractArchiveDataSource` are datasources based on files located in a specific archive, in the file system. As of today, two classes implements `AbstractArchiveDataSource`: `ZipArchiveDataSource` and `TarArchiveDataSource`

While the files located in the archive **have to be uncompressed**, the archive file itself can be compressed, depending on the archive format:

- A Zip archive is also already compressed so the compression format for `ZipArchiveDataSource` is always ZIP.
- A Tar archive can be compressed by: BZIP2, GZIP, XZ or ZSTD. It can also not be compressed.

Just like `DirectoryDataSource`, the archive datasources integrate the notions of base name and data extension. If not given as a parameter in the datasource constructor, the archive file name is even defined using the base name and the data extension, as `<directory>/<basename>.<dataExtension>.<archiveExtension>.<compressionExtension>` with the compression extension being optional depending on the archive format. For example `network.xiidm.zip` contains `network.xiidm`.

Unlike in directory datasources, in archive datasources the method `listNames(String regex)` filters filenames only by the regex and not by the base name.

Example

Let's consider a directory containing the following files:

```
directory
├── network
├── network.south
├── network.xiidm.gz
├── network.v3.xiidm.gz
├── network_mapping.csv.gz
├── network.gz
└── toto.xiidm.gz
```

A datasource on this directory could be used this way:

```
// Creation of a directory datasource with compression
GzDirectoryDataSource datasource = new GzDirectoryDataSource(testDir, "network", "xiidm",
↪ observer);

// Check if some files exist in the datasource by using the `exists(String fileName)`␣
↪method
// Since the datasource uses Gzip compression, ".gz" is added to the provided fileName␣
↪parameter
datasource.exists("test.toto"); // Returns false: the file "test.toto.gz" does not exist␣
↪in the directory
datasource.exists("network.south"); // Returns false: the file "network.south.gz" does␣
↪not exist
datasource.exists("network.xiidm"); // Returns true: the file "network.xiidm.gz" exists
datasource.exists("toto.xiidm"); // Returns true: the file "toto.xiidm.gz" exists

// Check if some files exist in the datasource by using the `exists(String suffix,␣
↪String ext)` method
datasource.exists("_south", "reduced"); // Returns false: the file "network_south.
↪reduced.gz" does not exist in the directory
datasource.exists(null, "xiidm"); // Returns true: the file "network.xiidm.gz" exists in␣
↪the directory
datasource.exists("_mapping", "csv"); // Returns true: the file "network_mapping.csv.gz"␣
↪exists in the directory

// We can create some a new file "network_test.txt.gz" and write "line1" inside
try (OutputStream os = dataSource.newOutputStream("_test", "txt", false)) {
    os.write("line1".getBytes(StandardCharsets.UTF_8));
}

// Another line can be added to the same file by setting the `append` boolean parameter␣
↪to true
try (OutputStream os = dataSource.newOutputStream("_test", "txt", true)) {
    os.write("line2".getBytes(StandardCharsets.UTF_8));
}

// We can read the file
try (InputStream is = dataSource.newInputStream("_test", "txt")) {
    System.out.println(ByteStreams.toByteArray(is)); // Displays "line1" then "line2"
}

// List the files in the datasource
Set<String> files = datasource.listNames(".*");
// returns a set containing: "network", "network.south", "network.xiidm", "network.v3.
↪xiidm", "network_test.txt", "network_mapping.csv.gz"
// The file "toto.xiidm.gz" is not listed due to the base name filtering

// Using a datasource with different parameters allows to use other files, even on the␣
↪same directory
GzDirectoryDataSource totoDatasource = new GzDirectoryDataSource(testDir, "toto", "xiidm
↪", observer);
totoDatasource.exists(null, "xiidm"); // Returns true: the file "toto.xiidm.gz" exists␣
```

(continues on next page)

(continued from previous page)

```
↪in the directory
Set<String> files = totoDatasource.listNames(".*");
// returns a set containing: "toto.xiidm.gz"
```

CHAPTER

TWO

GRID MODEL

2.1 Network and subnetwork

In the following sections, the different network components are described in terms of their main attributes and electrotechnical representation. The attributes shared by all the network components are described in the next table:

Attribute	Description
Id	Unique Id assigned to each network component
$Name$	Human readable identifier (not necessary unique)
$Fictitious$	To identify non-physical network components
$Aliases$	Additional unique identifiers associated with each network component
$Properties$	To add additional data items to network components

All equipment and the network itself are identified by a unique identifier which is the only required attribute. They can have a human-readable name. Offers the possibility of adding additional unique identifiers to each component. An alias can be qualified to indicate what it corresponds to.

Properties allow associating additional arbitrary data items under the general schema of pairs <`Key, Value`>.

To identify non-physical network components, one can use the fictitious property that is set to `false` by default.

A network can contain several subnetworks.

2.1.1 Validation level

The validation level can be set to `EQUIPMENT` or `STEADY_STATE_HYPOTHESIS`. A network at equipment level is a network with missing steady-state hypotheses. This occurs just after SCADA systems, before any state estimation. Once all steady-state hypotheses are filled, meaning that a load flow engine has all the data needed to perform a computation, the validation level switches to `STEADY_STATE_HYPOTHESIS`. For some processes, a minimal validation level of the network is required.

2.1.2 Network

In the PowSyBl grid model, the Network contains *substations*, which themselves contain *voltage levels*.

Characteristics

Attribute	Description
$SourceFormat$	Source format of the imported network model
$CaseDate$	Date and time of the target network that is being modeled
$ForecastDistance$	Number of minutes between the network generation date and the case date

The `SourceFormat` attribute is a required attribute that indicates the origin of the network model automatically set by the *importers*. If the case date and the forecast distance cannot be found in the case file, the network is considered as a snapshot: the case date is set to the current date, and the forecast distance is set to `0`.

Available extensions

2.1.3 Substation

A substation represents a specific geographical location with equipment grouped in one or several *voltage levels*.

Characteristics

Attribute	Description
$Country$	To specify in which country the substation is located
$GeographicalTags$	They make it possible to accurately locate the substation
TSO	To track to which Transmission System Operator the substation belongs

All three attributes are optional.

Available extensions

- *ENTSO-E Area*

2.1.4 Voltage level

A voltage level contains equipment with the same nominal voltage. Two voltage levels may be connected through lines (when they belong to different substations) or through transformers (they must be located within the same substation).

Characteristics

Attribute	Unit	Description
$NominalVoltage$	kV	Nominal base voltage
$LowVoltageLimit$	kV	Low voltage limit magnitude
$HighVoltageLimit$	kV	High voltage limit magnitude
$TopologyKind$		Level of connectivity detail

Specifications

Only `NominalVoltage` and `TopologyKind` are required.

The connectivity in each voltage level of the network can be defined at one of two levels: `node/breaker` or `bus/breaker`. The connectivity level can be different in each voltage level of the model.

In `node/breaker` the connectivity is described with the finest level of detail and can provide an exact field representation. This level could be described as a graph structure where the vertices are `Nodes` and the edges are `Switches` (breakers, disconnectors) or internal connections. Each equipment is associated to one `Node` (busbar sections, loads,

generators, ..), two `Nodes` (transmission lines, two-winding transformers, ...) or three `Nodes` (three-winding transformers). Each `Node` can only have one associated equipment. `Nodes` do not have an alphanumeric `Id` or `Name`, they are identified by an integer.

Using `bus/breaker` the voltage level connectivity is described with a coarser level of detail. In this case the vertices of the graph are `Buses`, defined explicitly by the user. A `Bus` has an `Id`, and may have a `Name`. Each equipment defines the bus or buses to which it is connected. `Switches` can be defined between buses.

PowSyBl provides an integrated topology processor that allows to automatically obtain a bus/breaker view from a node/breaker definition, and a bus/branch view from a bus/breaker view or definition. It builds the topology views from the open/close status of `Switches`. `Switches` marked as `retained` in the node/breaker level are preserved in the bus/breaker view.

The following diagram represents an example voltage level with two busbars separated by a circuit breaker, a transformer connected to one of them and three generators that can connect to any of the two busbars. The three topology levels are shown.

When defining the model, the user has to specify how the different pieces of equipment connect to the network. If the voltage level is built at node/breaker level, the user has to specify a `Node` when adding equipment to the model. If the user is building using bus/breaker level, the `Bus` of the equipment must be specified. Using this information, the model creates a `Terminal` that will be used to manage the point of connection of the equipment to the network.

Available extensions

- *Discrete Measurements*
- *Identifiable Short-Circuit*
- *Slack Terminal*

2.1.5 Area

An Area is a geographical zone of a given type.

An Area is composed of a collection of *voltage levels*, and a collection of area boundaries. Area boundaries can be terminals of equipments or `Boundary` objects from *boundary lines*.

The area type is used to distinguish between various area concepts of different granularity. For instance: control areas, bidding zones, countries...

A *voltage level* can belong to several areas, as long as all areas are of a different type.

The area boundaries define how interchange is to be calculated for the area. Area interchange is calculated by summing the active power flows across the area boundaries and can be obtained for AC part only (considering only AC boundaries), for DC part only (considering only DC boundaries) and in total (AC+DC). Note that if the Area has no boundary explicitly defined, the interchange is considered 0MW.

For area types that are meant to be used for area interchange control, e.g., in Load Flow simulations, the interchange target of the area can be specified as an input for the simulation.

All area interchange values use the load sign convention: positive values indicate that the area is importing, negative values that the area is exporting.

Characteristics of an Area

Attribute	Unit	Description
$AreaType$		To specify the type of Area (eg. ControlArea, BiddingZone ...)
$interchangeTarget$	MW	Target active power interchange
$VoltageLevels$		List of voltage levels of the area
$AreaBoundaries$		List of area boundaries of the area

Characteristics of an AreaBoundary

An area boundary is modeled by an `AreaBoundary` instance. It is composed of either BoundaryLine Boundary or a Terminal, and boolean telling if the area boundary is to be considered as AC or DC.

The `Ac` flag is informative and is present to support the use case where boundaries are defined on AC components even though the boundary is related to an HVDC link. An example for this is a BoundaryLine (which is an AC equipment) that may actually represent an HVDC interconnection that is not explicitly described in the network model. This information is used when computing area interchanges, which are then separated for AC and DC parts.

Attribute	Unit	Description
$Area$		The area of this boundary
$Boundary$		Boundary of a BoundaryLine (mutually exclusive with the Terminal attribute)
$Terminal$		Terminal of an equipment (mutually exclusive with the Boundary attribute)
Ac		True if AreaBoundary is to be considered AC, false otherwise

2.1.6 Generator

A generator is a piece of equipment that injects or consumes active power, and injects or consumes reactive power. It may be used as a controller to hold a voltage or reactive target somewhere in the network, not necessarily directly where it is connected. In that specific case, the voltage or reactive power control is remote.

Characteristics

Attribute	Unit	Description
$MinP$	MW	Minimum generator active power output
$MaxP$	MW	Maximum generator active power output
$ReactiveLimits$	MVar	Operational limits of the generator (P/Q/V diagram)
$RatedS$	MVA	The rated nominal power
$TargetP$	MW	The active power target
$TargetQ$	MVAr	The reactive power target at local terminal
$TargetV$	kV	The voltage target at regulating terminal which can be remote or local
$EquivalentLocalTargetV$	kV	The local voltage target consistent with the remote voltage target
$RegulatingTerminal$		Associated node or bus for which voltage is to be regulated, can be remote or local
$VoltageRegulatorOn$		True if the generator regulates voltage
$EnergySource$		The energy source harnessed to turn the generator
$IsCondenser$		True if the generator may behave as a condenser

Specifications

The values `MinP`, `MaxP` and `TargetP` are required. The minimum active power output cannot be greater than the maximum active power output. `TargetP` must be inside this active power limits. `RatedS` specifies the nameplate

apparent power rating for the unit, it is optional and should be a positive value if it is defined. The *reactive limits* of the generator are optional, if they are not given the generator is considered with unlimited reactive power. Reactive limits can be given as a pair of *min/max values* or as a *reactive capability curve*.

The `VoltageRegulatorOn` attribute is required. It voltage regulation is enabled, then `TargetV` and `RegulatingTerminal` must also be defined. If the voltage regulation is disabled, then `TargetQ` is required. `EnergySource` is optional, it can be: `HYDRO`, `NUCLEAR`, `WIND`, `THERMAL`, `SOLAR` or `OTHER`.

Target values for generators (`TargetP` and `TargetQ`) follow the generator sign convention: a positive value means an injection into the bus. Positive values for `TargetP` and `TargetQ` mean negative values at the flow observed at the generator `Terminal`, as `Terminal` flow always follows load sign convention. The following diagram shows the sign convention of these quantities with an example.

The `isCondenser` value corresponds for instance to generators which can control voltage even if their targetP is equal to zero.

The optional `EquivalentLocalTargetV` value can be used by simulators that deactivate the remote voltage algorithms, or by dynamic simulators that use this voltage as a starting value.

Available extensions

- *Active Power Control*
- *Coordinated Reactive Control*
- *Discrete Measurements*
- *Generator ENTSO-E Category*
- *Generator Short-Circuit*
- *Generator Startup*
- *Injection Observability*
- *Measurements*
- *Remote Reactive Power Control*
- *Manual Frequency Restoration Reserve*

2.1.7 Load

A load is a passive equipment representing a delivery point that consumes or produces active and reactive power.

Characteristics

Attribute	Unit	Description
$P0$	MW	The active power setpoint
$Q0$	MVar	The reactive power setpoint

Specifications

- Initial values for loads P0 and Q0 follow the passive-sign convention:
  - Flow out from the bus has a positive sign.
  - Consumptions are positive.

Metadata In the grid model, loads comprise the following metadata:

- The load type, which can be:

  - `UNDEFINED`
  - `AUXILIARY`
  - `FICTITIOUS`
- The load model, which can be:
  - `ZIP` (or polynomial), following equations:

$$P = P0 * (c0p + c1p \times (v/v_0) + c2p \times (v/v_0)^2)$$

$$Q = Q0 * (c0q + c1q \times (v/v_0) + c2q \times (v/v_0)^2)$$

    with v_0 the nominal voltage. Sum of $c0p$, $c1p$ and $c2p$ must be equal to 1. Sum of $c0q$, $c1q$ and $c2q$ must be equal to 1.
  - `EXPONENTIAL`, following equations:

$$P = P0 \times (v/v_0)^{n_p}$$

$$Q = Q0 \times (v/v_0)^{n_q}$$

    with v_0 the nominal voltage. n_p and n_q are expected to be positive.

Available extensions

- *Connectable position*
- *Discrete Measurements*
- *Identifiable Short-Circuit*
- *Injection Observability*
- *Load Asymmetrical*
- *Load Detail*
- *Measurements*

2.1.8 Battery

A battery on the electric grid is an energy storage device that is either capable of capturing energy from the grid or of injecting it into the grid. The electric energy on the grid side is thus transformed into chemical energy on the battery side and vice versa. The power flow is bidirectional, and it is controlled via a power electronic converter.

Characteristics

Attribute	Unit	Description
$TargetP$	MW	The active power target
$TargetQ$	MVar	The reactive power target
$MinP$	MW	The Minimal active power (charging limit)
$MaxP$	MW	The Maximum active power (discharging limit)
$ReactiveLimits$	MVar	Operational limits of the battery (P/Q/V diagram)

The values `TargetP`, `TargetQ`, `MinP`, `MaxP`, are required.

All attributes follow the generator sign convention: a positive value means an injection into the bus. Positive values for `TargetP` and `TargetQ` mean negative values at the flow observed at the battery `Terminal`, as `Terminal` flow always follows load sign convention.

The minimum active power output `MinP` cannot be greater than the maximum active power output `MaxP`. `MinP` represents the battery charging active power limit, and is typically negative. `MaxP` represents discharge active power limit, and is typically positive.

The *reactive limits* of the battery are optional, if they are not given the battery is considered with unlimited reactive power. Reactive limits can be given as a pair of *min/max values* or as a *reactive capability curve*.

Available extensions

- *Active Power Control*
- *Connectable position*
- *Discrete Measurements*
- *Identifiable Short-Circuit*
- *Injection Observability*
- *Measurements*

2.1.9 Boundary line

A network may be connected to other networks for which a full description is not available or unwanted. In this case, a boundary line exists between the two networks. In the network of interest, that connection could be represented through a boundary line, which represents the part of that boundary line which is located in it. A boundary line is thus a passive or active component that aggregates a line chunk and a constant power injection in passive-sign convention. The active and reactive power set points are fixed: the injection represents the power flow that would occur through the connection, were the other network fully described.

A generation part, at boundary side can also be modeled with a constant active power injection and a constant reactive power injection if the generation part of the boundary line is out of voltage regulation or a voltage target if the regulation is enabled. This fictitious generator can only regulate voltage locally: the regulating terminal cannot be set, it is necessary for the boundary side of the boundary line. Limits are modeled through $MinP$ and $MaxP$ for active power limits and through *reactive limits*. This generation part is optional. The generation part of the boundary line follows the classical generator sign convention.

Resulting flows at the boundary line terminal all follow the same passive-sign convention, either for the injection part or for the generation part.

Boundary lines are key objects for merging networks. Merging will be described soon *here*.

Characteristics

Attribute	Unit	Description
$P0$	MW	The active power setpoint
$Q0$	MVar	The reactive power setpoint
R	Ω	The series resistance
X	Ω	The series reactance
G	S	The shunt conductance
B	S	The shunt susceptance

Optional:

Attribute	Unit	Description
$MinP$	MW	Minimum generation part active power output
$MaxP$	MW	Maximum generation part active power output
$ReactiveLimits$	MVar	Operational limits of the generation part (P/Q/V diagram)
$TargetP$	MW	The active power target
$TargetQ$	MVAr	The reactive power target
$TargetV$	kV	The voltage target
$VoltageRegulatorOn$		True if the generation part regulates voltage

Specifications

- $P0$ and $Q0$ are the active and reactive power setpoints
- R, X, G and B correspond to a fraction of the original line and have to be consistent with the declared length of the boundary line (see example below).

Example: Considering a line D of length L, composed of two boundary lines D_1 and D_2 of lengths $L_1 = k_1 L$ and $L_2 = k_2 L$ such as $L = L_1 + L_2$. Then the characteristics of the boundary lines are:

- $R_1 = k_1 R$ and $R_2 = k_2 R$,
- $X_1 = k_1 X$ and $X_2 = k_2 X$,
- $G_1 = k_1 G$ and $G_2 = k_2 G$,
- $B_1 = k_1 B$ and $B_2 = k_2 B$,

Note that if we had only D_1 without D_2, the previous relation between D and D_1 would still hold.

In case the line is a boundary, a pairing key $pairingKey$ (in previous network versions $UcteXnodeCode$) is defined beside the characteristics of the table. It is a key to match two boundary lines and reconstruct the full boundary line, as a *TieLine*, for both UCTE or CIM-CGMES formats.

A boundary line has a `Boundary` object that emulates a terminal located at boundary side. A boundary line is a connectable with a single terminal located on the network side, but sometimes we need state variables such as active or reactive powers on the other side, voltage angle and voltage magnitude at fictitious boundary bus. Note that P, Q, V and $Angle$ at boundary are automatically computed using information from the terminal of the boundary line.

Available extensions

- *CGMES Boundary Line Boundary Node*
- *Connectable position*
- *Discrete Measurements*
- *Identifiable Short-Circuit*
- *Injection Observability*
- *Measurements*

2.1.10 Shunt compensator

A shunt compensator represents a shunt capacitor or reactor or a set of switchable banks of shunt capacitors or reactors in the network. A section of a shunt compensator is an individual capacitor or reactor: if its reactive power (Q) is negative, it is a capacitor; if it is positive, it is a reactor.

There are two supported models of shunt compensators: linear shunt compensators and non-linear shunt compensators.

A linear shunt compensator has banks or sections with equal admittance values. A non-linear shunt compensator has banks or sections with different admittance values.

Shunt compensators follow a passive-sign convention:

- Flow out from bus has positive sign.
- Consumptions are positive.

Characteristics

Attribute	Unit	Description
$MaximumSectionCount$	-	The maximum number of sections that may be switched on
$SectionCount$	-	The current number of sections that are switched on (input of the calculation)
$SolvedSectionCount$	-	The calculated number of sections that are switched on (after a load flow)
B	S	The susceptance of the shunt compensator in its current state
G	S	The conductance of the shunt compensator in its current state
$TargetV$	kV	The voltage target
$TargetDeadband$	kV	The deadband used to avoid excessive update of controls
$RegulatingTerminal$	-	Associated node or bus for which voltage is to be regulated
$VoltageRegulatorOn$	-	True if the shunt compensator regulates voltage

- For Linear Shunt Compensators

Attribute	Unit	Description
$bPerSection$	S	The Positive sequence shunt (charging) susceptance per section
$gPerSection$	S	The Positive sequence shunt (charging) conductance per section

We expect $bPerSection$ to be a non-zero value. The disconnected status of the linear shunt compensator can be modeled by setting the $SectionCount$ attribute to zero.

- For Non-Linear Shunt Compensators

Attribute	Unit	Description
$Sections$	*Section*	The Partition of all the shunt compensator's sections

Section

Attribute	Unit	Description
B	S	The accumulated positive sequence shunt (charging) susceptance of the section if this section and all the previous ones are activated
G	S	The accumulated positive sequence shunt (charging) conductance of the section if this section and all the previous ones are activated

B and G attributes can be equal zero, but the disconnected status of the non-linear shunt compensator can be modeled by setting the $SectionCount$ attribute to zero. The section which $SectionCount$ equal to 1 is the first effective section, and it would be more efficient to affect it a non-zero susceptance.

Specifications

- A section of a shunt compensator is an individual capacitor or reactor. A positive value of bPerSection means that it models a capacitor, a device that injects reactive power into the bus. A negative value of bPerSection means a reactor, a device that can absorb excess reactive power from the network.
- The current section count is expected to be greater than one and lesser or equal to the maximum section count.
- Regulation for shunt compensators does not necessarily model automation, it can represent human actions on the network e.g. an operator activating or deactivating a shunt compensator). However, it can be integrated on a power flow calculation or not, depending on what is wanted to be shown.
- In the case of a capacitor, the value for its Q will be negative.
- In the case of a reactor, the value for its Q will be positive.

Available extensions

- *Connectable position*
- *Discrete Measurements*
- *Identifiable Short-Circuit*
- *Injection Observability*
- *Measurements*

2.1.11 Static VAR compensator

It may be controlled to hold a voltage or reactive setpoint somewhere in the network (not necessarily directly where it is connected). Static VAR compensators follow a passive-sign convention:

- Flow out from bus has positive sign.
- Consumptions are positive.

Characteristics

Attribute	Unit	Description
$Bmin$	S	The minimum susceptance
$Bmax$	S	The maximum susceptance
$VoltageSetpoint$	kV	The voltage setpoint
$ReactivePowerSetpoint$	MVar	The reactive power setpoint

Specifications

- $Bmin$ and $Bmax$ are the susceptance bounds of the static VAR compensator. Reactive power output of a static VAR compensator is limited by the maximum and the minimum susceptance values. The min/max reactive power of a static VAR compensator is determined by:

$$Qmin = -Bmin \times V^2$$

$$Qmax = -Bmax \times V^2$$

  where V is the voltage of the bus that connects the static VAR compensator to the network. Even if the regulating terminal is remote, only the local voltage has to be considered to retrieve the minimum and the maximum amount of reactive power. Reactive limits can be handled in an approximate way using the nominal voltage of the connected bus.

- The voltage setpoint is required when the regulation mode is set to `VOLTAGE`.

- The reactive power setpoint is required when the regulation mode is set to `REACTIVE_POWER`.

Metadata In IIDM the static VAR compensator also comprises some metadata:

- The regulation mode, which can be:
  - `VOLTAGE`
  - `REACTIVE_POWER`
- The participation in regulation (through a boolean)
- The regulating terminal, which can be local or remote: it is the specific connection point on the network where the setpoint is measured.

Available extensions

- *Connectable position*
- *Discrete Measurements*
- *Identifiable Short-Circuit*
- *Injection Observability*
- *Measurements*
- *VoltagePerReactivePowerControl*

2.1.12 Line

AC transmission lines are modeled using a standard π model with distributed parameters. A `Line` is a `Branch`, that models equipment with two terminals (or two sides). For the time being, a branch is an AC equipment.

With series impedance z and the shunt admittance on each side y_1 and y_2:

$$\begin{aligned} z &= r + j.x \\ y_1 &= g_1 + j.b_1 \\ y_2 &= g_2 + j.b_2 \end{aligned}$$

The equations of the line, in complex notations, are as follows:

$$\begin{pmatrix} I_1 \\ I_2 \end{pmatrix} = \begin{pmatrix} y_1 + \frac{1}{z} & -\frac{1}{z} \\ -\frac{1}{z} & y_2 + \frac{1}{z} \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \end{pmatrix}$$

Characteristics

Attribute	Unit	Description
R	Ω	The series resistance
X	Ω	The series reactance
$G1$	S	The first side shunt conductance
$B1$	S	The first side shunt susceptance
$G2$	S	The second side shunt conductance
$B2$	S	The second side shunt susceptance

Metadata

- Lines can have *loading limits*.

Available extensions

- *Connectable position*
- *Branch Observability*
- *Operating Status*
- *CGMES Line Boundary Node*
- *Discrete Measurements*
- *Identifiable Short-Circuit*
- *Measurements*

2.1.13 Tie line

A tie line is an AC line sharing power between two neighbouring regional grids. It is created by pairing two *boundary lines* with the same pairing key. It has line characteristics, with R (resp. X) being the sum of the series resistances (resp. reactances) of the two boundary lines. $G1$ (resp. $B1$) is equal to the first boundary line's $G1$ (resp. $B1$). $G2$ (resp. $B2$) is equal to the second boundary line's $G2$ (resp. $B2$).

Characteristics

Attribute	Unit	Description
R	Ω	The series resistance
X	Ω	The series reactance
$G1$	S	The first side shunt conductance
$B1$	S	The first side shunt susceptance
$G2$	S	The second side shunt conductance
$B2$	S	The second side shunt susceptance

A tie line is not a connectable. It is just a container of two underlying boundary lines with the same pairing key. When connected together, each boundary line `P0` and `Q0` (and generation part if present) is ignored: only global tie line characteristics are used to compute flow. Removing a tie line leads to two free boundary lines, with an optional update of `P0` and `Q0` to match the flows in the global network context.

2.1.14 Transformers

Two-winding transformer

A two-winding power transformer is connected to two voltage levels (side 1 and side 2) that belong to the same substation. Two winding transformers are modeled with the following equivalent Π model:

With the series impedance z and the shunt admittance y and the voltage ratio ρ and the angle difference α and potential

parameters from the current step of a *ratio tap changer* and/or a *phase tap changer*, we have:

$$
\begin{aligned}
r &= r_{nom}.\left(1+\frac{r_{r,tap}+r_{\phi,tap}}{100}\right)\\
x &= x_{nom}.\left(1+\frac{x_{r,tap}+x_{\phi,tap}}{100}\right)\\
g &= g_{nom}.\left(1+\frac{g_{r,tap}+g_{\phi,tap}}{100}\right)\\
b &= b_{nom}.\left(1+\frac{b_{r,tap}+b_{\phi,tap}}{100}\right)\\
\rho &= \frac{V_{2nom}}{V_{1nom}}.\rho_{r,tap}.\rho_{\phi,tap}\\
\alpha &= \alpha_{\phi,tap}\\
z &= r+j.x\\
y &= g+j.b\\
V_0 &= V_1.\rho e^{j\alpha}\\
I_0 &= \frac{I_1}{\rho e^{-j\alpha}}
\end{aligned}
$$

Using the above notation, the equations of the two-winding transformers, in complex notations, are as follows:

$$
\begin{pmatrix} I_1 \\ I_2 \end{pmatrix} = \begin{pmatrix} \rho^2(y+\frac{1}{z}) & -\frac{1}{z}\rho e^{-j\alpha} \\ -\rho\frac{1}{z}e^{j\alpha} & \frac{1}{z} \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \end{pmatrix}
$$

Characteristics

Attribute	Unit	Description
R_{nom}	Ω	The nominal series resistance at the side 2 of the transformer
X_{nom}	Ω	The nominal series reactance at the side 2 of the transformer
G_{nom}	S	The nominal magnetizing conductance at the side 2 of the transformer
B_{nom}	S	The nominal magnetizing susceptance at the side 2 of the transformer
$V_{1\ nom}$	kV	The rated voltage at side 1
$V_{2\ nom}$	kV	The rated voltage at side 2
$RatedS$	MVA	The normal apparent power

Specifications

- A *ratio tap changer* and/or a *phase tap changer* can be associated with a two-winding power transformer.
- For a two-winding transformer, the normal apparent power shall be identical at both sides 1 and 2.

Available extensions

- *Branch Observability*
- *Operating Status*
- *Connectable position*
- *Discrete Measurements*
- *Identifiable Short-Circuit*
- *Measurements*
- *Two-windings Transformer Phase Angle Clock*
- *Two-windings Transformer To Be Estimated*

Three-winding transformer

A three-winding power transformer is connected to three voltage levels (side 1, side 2 and side 3) that belong to the same substation. We usually have:

- Side 1 as the primary side (side with the highest rated voltage)
- Side 2 as the secondary side (side with the medium rated voltage)
- Side 3 as the tertiary side (side with the lowest rated voltage)

A three-winding transformer is modeled with three legs, where every leg model is electrically equivalent to a two-winding transformer. For each leg, the network bus is at side 1 and the star bus is at side 2.

Characteristics

Attribute	Unit	Description
$RatedU0$	kV	The rated voltage at the star bus

Specifications

- A *ratio tap changer* and/or a *phase tap changer* can be associated to all three sides of a three-winding power transformer. Only one tap changer (either ratio or phase tap changer) is allowed to be regulating on the equipment at a given time.

Available extensions

- *Operating Status*
- *Connectable position*
- *Discrete Measurements*
- *Identifiable Short-Circuit*
- *Measurements*
- *Three-windings Transformer Phase Angle Clock*
- *Three-windings Transformer To Be Estimated*

Three-winding transformer leg

Characteristics

Attribute	Unit	Description
R	Ω	The nominal series resistance specified at the voltage of the leg
X	Ω	The nominal series reactance specified at the voltage of the leg
G	S	The nominal magnetizing conductance specified at the voltage of the leg
B	S	The nominal magnetizing susceptance specified at the voltage of the leg
$RatedU$	kV	The rated voltage
$RatedS$	MVA	The normal apparent power

Specifications

- A leg can have *loading limits*.

2.1.15 DC Equipments

Reduced vs. Detailed DC Models

Direct Current (DC) equipment can be modeled in two distinct ways: using a ***reduced*** model or a ***detailed*** model.

The ***reduced*** model abstracts away internal DC components. A DC link is represented as a single DC line rigidly connected to two AC/DC converters. In this representation, each converter connects to the AC network through a single connection.

In contrast, the ***detailed*** model includes a broader set of DC components, such as DC nodes, DC grounds, DC switches, DC lines, and AC/DC converters. This approach enables a more accurate representation of the wide range of HVDC configurations. It allows modeling of features such as:

- Metallic return cables, bipole systems, symmetrical and asymmetrical configurations, bypass DC switches, multi-terminal DC grids (radial or meshed)
- AC/DC converters with dual connections to the AC network (e.g., for the two transformers connections of a 12-pulse converter)

The same network may contain multiple HVDC links with either representation.

Both the *reduced* and *detailed* models support the representation of Line-Commutated Converters (LCCs) and Voltage Source Converters (VSCs).

Further details on these two modeling approaches are provided in the following sections.

Reduced DC model

HVDC line

An HVDC line is connected to the DC side of two HVDC converter stations, either an *LCC station* or a *VSC station*.

Characteristics

Attribute	Unit	Description
R	Ω	The resistance of the HVDC line
$NominalV$	kV	The nominal voltage
$ActivePowerSetpoint$	MW	The active power setpoint
$MaxP$	MW	The maximum active power

Specifications

- The HVDC line operation depends on a converter mode, which indicates the flow direction. In the specification it is thus mandatory to define `ConvertersMode`, which can be:
  - `SIDE_1_RECTIFIER_SIDE_2_INVERTER`: the flow goes from side 1 to side 2
  - `SIDE_1_INVERTER_SIDE_2_RECTIFIER`: the flow goes from side 2 to side 1

The flow sign is thus given by the type of the converter station: the power always flows from the rectifier converter station to the inverter converter station. At a terminal on the AC side, `P` and `Q` follow the passive sign convention. `P` is positive on the rectifier side. `P` is negative at the inverter side.

- The active power setpoint and the maximum active power should always be positive values.

Available extensions

- *HVDC Angle Droop Active Power Control*
- *HVDC Operator Active Power Range*

HVDC converter station

An HVDC converter station converts electric power from high voltage alternating current (AC) to high-voltage direct current (HVDC), or vice versa. Electronic converters for HVDC are divided into two main categories: line-commutated converters (LCC) and voltage-sourced converters (VSC).

Characteristics

Attribute	Type	Unit	Required	Default value	Description
HvdcType	`HvdcType`	-	yes	-	The HVDC type
LossFactor	float	%	yes	-	The loss factor

The LossFactor should be greater than 0.

Specifications

The HVDC type, `LCC` or `VSC`, determines if the Converter Station is an LCC Converter Station or a VSC Converter Station.

The positive loss factor `LossFactor` is used to model the losses during the conversion. In case of:

- A rectifier operation (conversion from AC to DC), we have

$$\frac{P_{DC}}{P_{AC}} = 1 - \frac{LossFactor}{100}$$

- An inverter operation (conversion from DC to AC), we have

$$\frac{P_{AC}}{P_{DC}} = 1 - \frac{LossFactor}{100}$$

Note that at the terminal on the AC side, Q is always positive: the converter station always consumes reactive power.

LCC converter station

An LCC converter station is made with electronic switches that can only be turned on (thyristors). Below are some characteristics:

- Use semiconductors which can withstand voltage in either polarity
- Output voltage can be either polarity to change the power direction
- Current direction does not change
- Store energy inductively

- Use semiconductors which can turn on by control action
- Turn-off and commutation rely on the external circuit

Characteristics

Attribute	Unit	Description
$PowerFactor$	%	Ratio between the active power P and the apparent power S.

Available extensions

- *Connectable position*

VSC converter station

A VSC converter station is made with switching devices that can be turned both on and off (transistors). Below are some characteristics:

- Use semiconductors which can pass current in either direction
- Output voltage polarity does not change
- Current direction changes to change the power direction
- Store energy capacitively
- Use semiconductors which can turn on or off by control action
- Turn-off is independent of external circuit

Characteristics

Attribute	Unit	Description
$VoltageSetpoint$	kV	The voltage setpoint for regulation
$ReactivePowerSetpoint$	MVar	The reactive power setpoint for regulation

Specifications

- The voltage setpoint (in kV) is required if the voltage regulator is on for the VSC station.
- The reactive power setpoint (in MVar) is required if the voltage regulator is off for the VSC station. A positive value of $ReactivePowerSetpoint$ means an injection into the bus, thus a negative value at the corresponding terminal (which is in passive-sign convention).
- A set of reactive limits can be associated to a VSC converter station. All the reactive limits modeling available in the library are described *here*.

Metadata

- The participation in regulation (through a boolean)

Available extensions

- *Connectable position*

Detailed DC model

> **Note**
>
> Currently, this model is only available in the IIDM representation. Support in exchange formats (CGMES, ...) as well as in downstream projects (e.g., `powsybl-diagram`, `powsybl-open-loadflow`, etc.) may vary. Please consult the documentation of each project to verify support. In general, lack of explicit mention means no support.
>
> If you're unsure, feel free to reach out to the PowSyBl community here.

DC Node

DC nodes are points where DC terminals of DC conducting equipment are connected together with zero impedance.

Characteristics

Attribute	Unit	Description
$NominalV$	kV	The nominal voltage, always positive

Although the nominal voltage of DC nodes must always be specified as a positive value, the solved voltages can be negative - for example, in the case of an LCC monopole operating in reverse polarity.

DC Line

A DC Line connects two DC Nodes with a series resistance.

A DC Line has two DC Terminals.

Characteristics

Attribute	Unit	Description
R	Ω	The series resistance, always positive

DC Switch

A DC Switch connects two DC Nodes and can be opened or closed.

Characteristics

Attribute	Unit	Description
$Kind$	`DcSwitchKind`	Either DISCONNECTOR or BREAKER
$Open$		True if the switch is opened

DC Ground

DC Grounds represent grounding electrodes and are modeled as having zero voltage potential with a grounding resistance taken into account.

A DC Ground has a single DC Terminal.

Characteristics

Attribute	Unit	Description
R	Ω	The grounding resistance, always positive

AC/DC Converter

AC/DC Converter transfers power between AC and DC grids. Its connectivity is modeled with:

- either one or two AC Terminals,
- exactly two DC Terminals

AC/DC Converter can be either a Line Commutated Converter (LCC) or Voltage Source Converter (VSC). LCC and VSC share the following characteristics.

Characteristics

Attribute	Unit	Description
$IdleLoss$	MW	Losses at no load
$SwitchingLoss$	MW / A	Switching losses
$ResistiveLoss$	Ω	Resistive losses
$PccTerminal$		Point of common coupling (PCC) AC terminal
$ControlMode$		The converter's control mode: P_PCC, V_DC or V_DC_DROOP
$TargetP$	MW	Active power target at point of common coupling, load sign convention
$TargetVdc$	kV	DC voltage target
$DroopCurve$		Droop curve for droop control mode

Converter losses are modeled using the `IdleLoss`, `SwitchingLoss` and `ResistiveLoss` parameters, all positive values. With `i` being the DC current through the converter, the Converter losses are computed as follows:

$$ConverterLosses = IdleLoss + SwitchingLoss.|i| + ResistiveLoss.i$$

The converter losses is the active power difference between the converter AC Terminal(s) and DC Terminals.

The Point of Common Coupling (PCC) Terminal defines where the AC/DC converter interfaces with the AC grid. The control mode defines whether the converter:

- controls active power at Point of Common Coupling
- or, controls DC voltage at its DC terminals

When the `ControlMode` of the converter is set to `P_PCC`, the converter controls active power flow at the (AC) Point of common coupling terminal. `TargetP` is the desired active power flow at PCC, in passive sign convention, i.e.:

- positive `TargetP` means power flows from AC to DC, the converter is sending AC power to the DC system
- negative `TargetP` means power flows from DC to AC, the converter is sending DC power to the AC system

For LCC 12-pulse converters, the PCC terminal is typically located on the station side of the transformers (not on the converter side). For VSC converters having a single AC Terminal, it is valid for the PCC Terminal to be the converter terminal itself. Because the PCC Terminal is used by the converter to control active power flow, it must be a branch terminal (a line or a transformer). It cannot be a Busbar Section Terminal since no active power can be measured on Busbar Sections.

For AC/DC converters modeled with two AC terminals, it is acceptable for the converter's PCC terminal to be designated as one of the converter terminals. In this case, when the converter's `ControlMode` is set to `P_PCC`, simulators shall interpret `TargetP` as the total active power flow to be achieved across both AC terminals, not just through the PCC Terminal.

When the `ControlMode` of the converter is set to `V_DC`, the converter controls DC voltage at its DC Terminals. `TargetVdc` is the desired target DC voltage, and is the voltage difference between DC Node 1 and DC Node 2. `TargetVdc` may be either positive or negative. Negative value may be used to model reverse polarity operation in case of LCCs. No explicit attribute specifies whether the DC is a symmetrical or asymmetrical scheme. The scheme symmetrical or asymmetrical is derived implicitly by either the presence or absence of a DC Ground connected to the DC system:

- If a DC Ground is connected, the configuration is asymmetrical, and the converter imposes the target voltage difference between the converter DC Node 1 and the DC Node 2 to be equal to `TargetVdc`
- If no DC Ground is present, the configuration is symmetrical, in this case the converter provides internally an implicit DC Ground and imposes:
  - `+TargetVdc / 2` at the converter DC Node 1
  - `-TargetVdc / 2` at the converter DC Node 2

When the `ControlMode` of the converter is set to `P_PCC_DROOP`, the converter controls active power as in the `P_PCC` control mode for normal load flow, but when a security analysis in run, the converter controls the relation between DC Voltage and DC Power: $P_{DC} - P_{REF} = -k * (V_{DC} - V_{REF})$ Where:

- k is the droop coefficient of the actual droop segment.
- P_{REF} is the power which was calculated during the base loadflow, at DC side, so it is not equal to targetP which is the AC setpoint. It represents the operating point before the security analysis starts.
- V_{REF} is the DC voltage which was calculated during the base loadflow. The droop control is only used for P controlled converters, so they should not have a targetVdc.
- P_{DC} is the actual power at DC side during the security analysis, which is determined by Newton Raphson.
- V_{DC} is the actual DC voltage during the security analysis, which is determined by Newton Raphson.

Each droop segment in the `DroopCurve` is defined with minimal and maximal voltage, and a droop coefficient. The actual droop segment should be the one which verifies: $V_{DC} \in [V_{min}, V_{max}]$ where V_{DC} is the DC Voltage at converter's Terminals.

Line Commutated Converter

Characteristics

Attribute	Unit	Description
$ReactiveModel$		FIXED_POWER_FACTOR or CALCULATED_POWER_FACTOR
$PowerFactor$	%	Ratio between the active power P and the apparent power S.

Line Commutated Converters always consume reactive power, the `PowerFactor` attribute specifies how much is consumed when the reactive model is set to `FIXED_POWER_FACTOR`. Typical characteristic for LCCs is $Q = 0.5P$ hence a PowerFactor of 0.89443.

Voltage Source Converter

Characteristics

Attribute	Unit	Description
$VoltageRegulatorOn$		True if the converter regulates voltage
$VoltageSetpoint$	kV	The voltage setpoint for regulation
$ReactivePowerSetpoint$	MVar	The reactive power setpoint for regulation

Specifications

- The terminal used for regulation is the Point of Common Coupling terminal, for both voltage and reactive power control modes.
- The voltage setpoint (in kV) is required if the voltage regulator is on for the converter.
- The reactive power setpoint (in MVar) is required if the voltage regulator is off for the converter. The setpoint is in passive sign convention: a positive value of $ReactivePowerSetpoint$ means withdrawal from the bus.
- A set of reactive limits can be associated to a VSC converter. All the reactive limits modeling available in the library are described *here*.

DC Topology Processing

DcTerminal DcBus

DC equipment connectivity may be modified in two ways:

- By changing the `connected` attribute of a `DcTerminal` of a DC Line, or a DC Ground, or an AC/DC Converter:
  - When `connected = true`, the DC terminal is connected to its associated DC Node.
  - When `connected = false` the DC terminal is disconnected from its associated DC Node.
- By changing the `open` attribute of a `DcSwitch`.

PowSyBl's IIDM topology processor computes DC Buses as follows:

- A DC Bus is formed when there is at least one DC Terminal connected to a DC Node.
- DC Nodes linked by a closed DC switch are considered part of the same DC Bus.
- A DC Node with no switch connected but with at least a DC Terminal connected will form a DC Bus.
- DC Nodes without any connected DC Terminal do not form a DC Bus. A DC Bus is guaranteed to contain at least one connected DC Terminal.

DC Buses linked together via DC Lines and/or AC/DC Converters are part of the same *DC Component* (also called *DC Island*). *Synchronous Components* (also called *AC Islands*) connected together via a DC island through AC/DC converters will form a *Connected Component*.

The IIDM API provides methods for navigating the network topology, for example:

- getting the DC Bus of a DC Terminal,
- getting the DC Nodes part of a DC Bus,
- getting the DC Component/Island of a DC Bus,
- getting all DC Buses of a network or a subnetwork
- getting all DC Components/Islands of a network or a subnetwork

- getting all DC Buses part of a DC Component or a Connected Component
- etc.

Please refer to the javadoc for an exhaustive list of the available methods.

DC Equipment containment in main network and subnetworks

Modeling a DC link involves creating multiple objects in the model such as DC nodes, lines, converters, switches, grounds which relate to each other. When creating DC equipment, associations can be made only within the same network, which can be either the main network or the same subnetwork. For example, the following associations are rejected by the model:

- creating a DC line connecting a DC Node A in subnetwork A with DC Node B in subnetwork B
- creating a DC Ground in subnetwork A connecting a DC Node contained in subnetwork B
- creating an AC/DC converter in a voltage level in subnetwork A connecting to DC nodes contained in main network
- setting the Point of Common Coupling of an AC/DC converter in a voltage level in subnetwork A to be a line terminal in subnetwork B
- etc.

For more details about working with subnetworks, see *Working with subnetworks*.

2.1.16 Busbar section

A busbar section is a non impedant element used in a node/breaker substation topology to connect equipment.

Available extensions

- *Busbar Section Position*
- *Discrete Measurements*
- *Identifiable Short-Circuit*
- *Injection Observability*
- *Measurements*

2.1.17 Breaker/switch

A switch is a device designed to close or open one or more electric circuits. There are several types of switches:

- breakers are capable of breaking currents under abnormal operating conditions (e.g. short-circuit);
- load break switches are capable of breaking currents under normal operating conditions;
- and disconnectors can only make or break negligible current.

A switch has an attribute to say if it is open or closed.

Available extensions

- *Discrete Measurements*

2.1.18 Internal connection

An internal connection is a zero-impedance connection between two elements in a voltage level.

Contrary to the switch, the internal connection does not have any attribute to say of it is open or closed.

2.1.19 Overload management systems

An overload management system is an automation system that monitors current on a terminal, defined by an equipment and an optional side. Based on the measured values, various strategies (referred as 'trippings' in the model) can be implemented to reduce the current. In a given strategy, if the current exceeds a threshold, a switch can be opened or closed to resolve the violation.

The switch open or close operation could be modeled using a switch ID, an association of a branch ID and a side, or an association of a three-windings transformer ID and a side.

Overload management system Characteristics

Attribute	Unit	Description
$Substation$		The substation associated where the system is installed
$MonitoredElementId$		The network element on which the limit will be monitored
$MonitoredSide$		The side of the element that is monitored

Tripping Characteristics

The supported trippings are:

- Branch tripping,
- Switch tripping,
- and three-windings transformer tripping.

Attribute	Unit	Description
$Type$		The type of tripping (e.g. BranchTripping, SwitchTripping, ...)
$CurrentLimit$	A	The current limit for which the action will be triggered
Key		The tripping key
$OpenAction$	bool	Whether the tripping should be opened or closed

2.2 Advanced

Network elements can be described in an advanced way with reactive limits, loading limits, phase and ratio tap changers.

2.2.1 Reactive limits

The reactive limits may be used to model limitations of the reactive power of *generators*, *VSC converter stations* and *batteries*.

Min-Max reactive limits

With the min-max reactive limits, the reactive power does not depend on the active power. For any active power value, the reactive power value is in the [minQ, maxQ] interval.

Reactive capability curve

With the reactive capability curve limits, the reactive power limitation depends on the active power value. This dependency is based on a curve provided by the user. The curve is defined as a set of points that associate, to each active power value, a minimum and maximum reactive power value. In between the defined points of the curve, the reactive power limits are computed through a linear interpolation.

Examples

This example shows how to use the `MinMaxReactiveLimits` and `ReactiveCapabilityCurve` classes:

```
Generator generator = network.getGenerator("G");
if (generator.getReactiveLimits().getKind() == ReactiveLimitsKind.MIN_MAX) {
    MinMaxReactiveLimits limits = generator.getReactiveLimits(MinMaxReactiveLimits.
↪class);
    System.out.println("MinMaxReactiveLimits: [" + limits.getMinQ() + ", " + limits.
↪getMaxQ() + "]");
} else {
    ReactiveCapabilityCurve limits = generator.getReactiveLimits(ReactiveCapabilityCurve.
↪class);
    System.out.println("ReactiveCapabilityCurve:");
    limits.getPoints().forEach(p -> System.out.println("\t" + p.getP() + " -> [" + p.
↪getMinQ() + ", " + p.getMaxQ() + "]"));
}
```

This example shows how to create a new `MinMaxReactiveLimits` object:

```
Generator generator = network.getGenerator("G");
generator.newMinMaxReactiveLimits()
    .setMinQ(-100.0)
    .setMaxQ(100.0)
    .add();
```

This example shows how to create a new `ReactiveCapabilityCurve` object:

```
Generator generator = network.getGenerator("G");
generator.newReactiveCapabilityCurve()
    .beginPoint()
        .setP(-10)
        .setMinQ(-10)
        .setMaxQ(10)
    .endPoint()
    .beginPoint()
        .setP(0)
        .setMinQ(-20)
        .setMaxQ(20)
    .endPoint()
    .beginPoint()
        .setP(10)
        .setMinQ(-15)
        .setMaxQ(-15)
    .endPoint()
    .add();
```

2.2.2 Loading Limits

Some equipment has operational limits regarding the current, active power or apparent power value, corresponding to the equipment's physical limitations (related to heating).

Loading limits can be declined into active power limits (in MW), apparent power limits (in kVA) and current limits (in A). They may be set for *lines*, *boundary lines*, *tie lines* (via their boundary lines), *two-winding transformers* and

three-winding transformers. The active power limits are in absolute value.

Loading limits are defined by one permanent limit and any number of temporary limits (zero or more). The permanent limit sets the current, active power or apparent power absolute value under which the equipment can safely be operated for any duration. The temporary limits can be used to define higher current, active power or apparent power limitations corresponding to specific operational durations. A temporary limit thus has an **acceptable duration**.

The component on which the current limits are applied can safely remain between the preceding limit (it could be another temporary limit or a permanent limit) and this limit for a duration up to the acceptable duration. Please look at this scheme to fully understand the modeling (the following example shows current limits, but this modeling is valid for all loading limits):

Note that, following this modeling, in general, the last temporary limit (the higher one in value) should be infinite with an acceptable duration different from zero, except for tripping current modeling where the last temporary limit is infinite with an acceptable duration equal to zero. If temporary limits are modeled, the permanent limit becomes mandatory. If no temporary limit is present, then the acceptable duration above the permanent limit will be infinite.

Limit group collection

In network development studies or in an operational context (CGMES), we can have a set of operational limits according to the season (winter vs summer, for example), the time of the day (day vs night) etc. In PowSyBl, users can store a collection of limits:

- Active power limits, apparent power limits and current limits are gathered into an `OperationalLimitsGroup` object.
- Lines, transformers (and other mentioned above in *Loading limits*) are associated with a collection of `OperationalLimitsGroup` (one collection per side/leg). Users can then choose one or more `OperationalLimitsGroup` to activate according to their needs.

`OperationalLimitsGroup` objects have an `id`, and may have properties — which allow associating additional arbitrary data items under the general schema of pairs <Key, Value>. Note that unlike the properties on the network components, no notification is emitted when a property is added, changed or removed.

Examples

Four examples are provided below, with their corresponding limits scheme, to show clearly how to create new `CurrentLimits` instances.

First example

This first example creates a `CurrentLimits` instance containing one permanent limit and two temporary limits.

```
CurrentLimits currentLimits = network.getBoundaryLine("DL").newCurrentLimits()
    .setPermanentLimit(100.0)
    .beginTemporaryLimit()
        .setName("TL1")
        .setValue(120.0)
        .setAcceptableDuration(20 * 60)
    .endTemporaryLimit()
    .beginTemporaryLimit()
        .setName("TL2")
        .setValue(140.0)
        .setAcceptableDuration(10 * 60)
    .endTemporaryLimit()
    .add();
```

Second example

This second example creates a `CurrentLimits` instance containing one permanent limit and three temporary limits, one of them having an infinite limit value.

```
CurrentLimits currentLimits = network.getBoundaryLine("DL").newCurrentLimits()
    .setPermanentLimit(700.0)
    .beginTemporaryLimit()
        .setName("IT20")
        .setValue(800.0)
        .setAcceptableDuration(20 * 60)
    .endTemporaryLimit()
    .beginTemporaryLimit()
        .setName("IT10")
        .setValue(900.0)
        .setAcceptableDuration(10 * 60)
    .endTemporaryLimit()
    .beginTemporaryLimit()
        .setName("IT1")
        .setValue(Double.POSITIVE_INFINITY)
        .setAcceptableDuration(60)
    .endTemporaryLimit()
    .add();
```

Third example

This third example shows how to create multiple OperationalLimitsGroup on the same end, and set one of these as the selected (active) one.

```
Line line = network.getLine("Line");

line.newOperationalLimitsGroup1("SUMMER")
        .newCurrentLimits()
        .setPermanentLimit(60)
        .beginTemporaryLimit()
        .setName("TATL-10")
        .setValue(80)
        .setAcceptableDuration(10 * 60)
        .endTemporaryLimit()
        .add();

line.newOperationalLimitsGroup1("WINTER")
        .newCurrentLimits()
        .setPermanentLimit(100)
        .beginTemporaryLimit()
        .setName("TATL-10")
        .setValue(120)
        .setAcceptableDuration(10 * 60)
        .endTemporaryLimit()
        .add();
```

(continues on next page)

(continued from previous page)

```
line.setSelectedOperationalLimitsGroup1("WINTER");
```

In this example, there is two sets of limits on the same line side (1). The selected set is the winter one: the limits violations will be tested against this set. Note that all `setSelected` methods will only keep a single set of limit activated at a time. Selecting "WINTER" then "SUMMER" this way will result in the only active group being the last selected (aka "SUMMER"). To have multiple `OperationalLimitsGroup` selected at the same time, see below the fourth example.

Fourth example

This fourth example creates three OperationalLimitsGroup on the same end, and sets all of them as selected (active).

```
final String activatedOneOne = "activated_1_1";
final String activatedOneTwo = "activated_1_2";

Line line = network.getLine("Line");

// create the default operational limits group and select it
line.getOrCreateSelectedOperationalLimitsGroup1().newCurrentLimits().
↪setPermanentLimit(500).add();
// create two other operational limits group
line.newOperationalLimitsGroup1(activatedOneOne).newCurrentLimits()
        .setPermanentLimit(1100)
        .beginTemporaryLimit()
        .setName("10'")
        .setAcceptableDuration(10 * 60)
        .setValue(1200)
        .endTemporaryLimit()
        .beginTemporaryLimit()
        .setName("1'")
        .setAcceptableDuration(60)
        .setValue(1500)
        .endTemporaryLimit()
        .beginTemporaryLimit()
        .add();

line.newOperationalLimitsGroup1(activatedOneTwo).newCurrentLimits()
        .setPermanentLimit(300)
        .beginTemporaryLimit()
        .setName("40'")
        .setAcceptableDuration(40 * 60)
        .setValue(700)
        .endTemporaryLimit()
        .beginTemporaryLimit()
        .setName("0.5'")
        .setAcceptableDuration(30)
        .setValue(1600)
        .endTemporaryLimit()
        .beginTemporaryLimit()
        .setName("N/A") //there is no need to explicitly create this limit for the␣
↪acceptable duration above the last temporary limit to be 0, this is just an example
```

(continues on next page)

(continued from previous page)

```
        .setAcceptableDuration(0)
        .setValue(Double.MAX_VALUE)
        .endTemporaryLimit()
        .add();

// select the two other limits group on the side 1, no need to select the first created␣
↪group, the function we used already selected it
line.addSelectedOperationalLimitsGroups(TwoSides.ONE, activatedOneOne, activatedOneTwo);
```

We thus have 3 `OperationalLimitsGroup` selected on the side 1 of the line. Note that not all created groups need to be activated, here all are activated, but we could have activated only 2 out of 3 (or 1 out of 3). Each activated group can return information about the acceptable duration (which can be different depending on the group).

The spacing on the y axis (branch current) is not to scale.

Permanent limit without any temporary limit

In the case of a group containing only a permanent limit (without any temporary above), there is no information available for the acceptable duration. The default acceptable duration has been chosen as infinite.

If you wish to have an acceptable duration of 0 instead, create a temporary limit with an acceptable duration of 0 and a value of Double.MAX_VALUE.

2.2.3 Phase tap changer

A phase tap changer can be added to either *two-winding transformers* or *three-winding transformers' legs*.

Specifications

A phase tap changer is described by a set of tap positions (or steps) within which the transformer or transformer leg can operate. Additionally, to that set of steps, it is necessary to specify:

- the lowest tap position
- the highest tap position
- the position index of the current tap (which has to be within the highest and lowest tap position bounds)
- the solved position index of the tap that represents the index after a calculation
- whether the phase tap changer can change tap positions onload or only offload

If the phase tap changer can change tap positions onload, regulation is specified as follows:

- whether the tap changer is regulating or not
- the regulation mode, which can be `CURRENT_LIMITER`, `ACTIVE_POWER_CONTROL`: the tap changer either regulates the current or the active power.
- the regulation value (either a current value in `A` or an active power value in `MW`)
- the regulating terminal, which can be local or remote: it is the specific connection point on the network where the setpoint is measured.
- the target deadband, which defines a margin on the regulation so as to avoid an excessive update of controls

Each step of a phase tap changer has the following attributes:

Attribute	Unit	Description
$r_{\phi,tap}$	%	The resistance deviation in percent of nominal value
$x_{\phi,tap}$	%	The reactance deviation in percent of nominal value
$g_{\phi,tap}$	%	The conductance deviation in percent of nominal value
$b_{\phi,tap}$	%	The susceptance deviation in percent of nominal value
$\rho_{\phi,tap}$	p.u.	The voltage ratio in per unit of the rated voltages
$\alpha_{\phi,tap}$	°	Angle difference

Example

This example shows how to add a phase tap changer to a two-winding transformer:

```
twoWindingsTransformer.newPhaseTapChanger()
    .setLowTapPosition(-1)
    .setTapPosition(0)
    .setLoadTapChangingCapabilities(true)
    .setRegulating(true)
    .setRegulationMode(PhaseTapChanger.RegulationMode.CURRENT_LIMITER)
    .setRegulationValue(25)
    .setRegulationTerminal(twoWindingsTransformer.getTerminal2())
    .beginStep()
        .setAlpha(-10)
        .setRho(0.99)
        .setR(1.)
        .setX(2.)
        .setG(0.5)
        .setB(0.5)
        .endStep()
    .beginStep()
        .setAlpha(0)
        .setRho(1)
        .setR(1.)
        .setX(2.)
        .setG(0.5)
        .setB(0.5)
        .endStep()
    .beginStep()
        .setAlpha(10)
        .setRho(1.01)
        .setR(1.)
        .setX(2.)
        .setG(0.5)
        .setB(0.5)
        .endStep()
    .add()
```

2.2.4 Ratio tap changer

A ratio tap changer can be added to either *two-winding transformers* or *three-winding transformers' legs*.

Specifications

A ratio tap changer is described by a set of tap positions (or steps) within which the transformer or transformer leg can operate (or be operated offload). Additionally, to that set of steps, it is necessary to specify:

- the lowest tap position
- the highest tap position
- the position index of the current tap (which has to be within the highest and lowest tap position bounds)
- the solved position index of the tap that represents the index after a calculation
- whether the ratio tap changer can change tap positions onload or only offload

If the ratio tap changer can change tap positions onload, regulation is specified as follows:

- whether the tap changer is regulating or not
- the regulation mode, which can be `VOLTAGE` or `REACTIVE_POWER`: the tap changer either regulates the voltage or the reactive power
- the regulation value (either a voltage value in `kV` or a reactive power value in `MVar`)
- the regulating terminal, which can be local or remote: it is the specific connection point on the network where the setpoint is measured.
- the target deadband, which defines a margin on the regulation so as to avoid an excessive update of controls

Each step of a ratio tap changer has the following attributes:

Attribute	Unit	Description
$r_{r,tap}$	%	The resistance deviation in percent of nominal value
$x_{r,tap}$	%	The reactance deviation in percent of nominal value
$g_{r,tap}$	%	The conductance deviation in percent of nominal value
$b_{r,tap}$	%	The susceptance deviation in percent of nominal value
$\rho_{r,tap}$	p.u.	The voltage ratio in per unit of the rated voltages

Example

This example shows how to add a ratio tap changer to a two-winding transformer:

```
twoWindingsTransformer.newRatioTapChanger()
    .setLowTapPosition(-1)
    .setTapPosition(0)
    .setLoadTapChangingCapabilities(true)
    .setRegulating(true)
    .setRegulationMode(RatioTapChanger.RegulationMode.VOLTAGE)
    .setRegulationValue(25)
    .setRegulationTerminal(twoWindingsTransformer.getTerminal1())
    .beginStep()
        .setRho(0.95)
        .setR(1.)
        .setX(2.)
```

(continues on next page)

(continued from previous page)

```
        .setG(0.5)
        .setB(0.5)
        .endStep()
    .beginStep()
        .setRho(1)
        .setR(1.)
        .setX(2.)
        .setG(0.5)
        .setB(0.5)
        .endStep()
    .beginStep()
        .setRho(1.05)
        .setR(1.)
        .setX(2.)
        .setG(0.5)
        .setB(0.5)
        .endStep()
    .add()
```

2.3 Grid model extensions

The grid model contains enough data to basically describe supported components and run power flow computations, but it may not be sufficient for more complex studies. The extensions are a way to add additional structured data to a piece of equipment to extend its features. The extensions can be attached to any objects of a network or to the network itself.

Some extensions are mono-variant, meaning the data are identical for all the variants of the network. However, some of them are multi-variants to allow a different value for each variant of the network. It's typically the case for the *LoadDetail* extension that give the distribution of the constant part and the thermosensitive part of the consumption.

Note that some extensions provided by PowSyBl aren't supported in the persistent implementation of IIDM.

Every extension is considered as serializable unless explicitly specified as non-serializable in XML-IIDM.

2.3.1 Active power control

This extension is used to configure the participation factor of the generator, typically in the case of a load flow computation with distributed slack enabled (with *balance type* on generator). This extension is attached to a *generator* or a *battery*.

Attribute	Type	Unit	Required	Default value	Description
participate	boolea	-	yes	-	The participation status
droop	double	None (repartition key)	no	-	The participation factor equals maxP / droop
participation factor	double	None (repartition key)	no	-	Defines the participation factor explicitly
max targetP	double	MW	no	-	If defined, this limit is used for slack distribution instead of the generator's maxP
min targetP	double	MW	no	-	if defined, this limit is used for slack distribution instead of the generator's minP

Here is how to add an active power control extension to a generator:

```
generator.newExtension(ActivePowerControlAdder.class)
    .withParticipate(true)
    .withDroop(4)
    .withParticipationFactor(1.5)
    .add();
```

If defined, min targetP and max targetP must be in the [pMin, pMax] interval of the Generator or Battery. The participation status, the participation factor, the max targetP and the min targetP are multi-variants: they can vary from one variant to another.

This extension is provided by the `com.powsybl:powsybl-iidm-extensions` module.

##(manual-frequency-restoration-reserve)=

2.3.2 Manual frequency restoration reserve

This extension models whether an injection is part of the manual frequency restoration reserve.

Attribute	Type	Unit	Required	Default value	Description
participate	boolean	-	yes	-	indicates if the injection participates in the manual frequency restoration reserve

This extension is provided in the `com.powsybl:powsybl-iidm-extensions` module.

Here is how to add a manual frequency restoration reserve extension to an injection:

```
generator.newExtension(ManualFrequencyRestorationReserve.class)
    .withParticipate(true)
    .add();
```

2.3.3 Branch observability

This extension models branch flow observability on both sides, obtained after a state estimation.

Attribute	Type	Unit	Required	Default value	Description
qualityP1	ObservabilityQuality	MW	no	-	The observability quality of active power on side ONE
qualityP2	ObservabilityQuality	MW	no	-	The observability quality of active power on side TWO
qualityQ1	ObservabilityQuality	MVar	no	-	The observability quality of reactive power on side ONE
qualityQ2	ObservabilityQuality	MVar	no	-	The observability quality of reactive power on side TWO

Observability quality

This extension contains the sub-object `ObservabilityQuality`.

Attribute	Type	Unit	Required	Default value	Description
standard deviation	double	MW or MVar	yes	-	The standard deviation
redundant	boolean	-	yes	-	Indicates if this value is confirmed by redundancy

This extension is provided by the `com.powsybl:powsybl-iidm-extensions` module.

2.3.4 Busbar section position

This extension gives positions information about a busbar section. The `busbarIndex` gives the position of the busbar section relative to other busbar sections. The `sectionIndex` gives the position of the busbar section within the corresponding busbar. Note that a busbar is a set of busbar sections. Hence, the sections of the same busbar should have the same busbar index. The busbar indices induce an order of busbars within the voltage level, which usually reflects the busbars physical relative positions. Similarly, the section indices induce an order of sections of the same busbar, which usually reflects their physical relative position.

2.3.5 Connectable position

This extension gives information about the relative position of connectables to each other in a voltage level. It also indicates the direction of the connectable relative to the busbar section to which it is connected. This is mainly used for visualization.

This extension is created by connectable, this information is available for each side of branches and three-winding transformers.

The attributes of the extension are given below:

Attribute	Type	Unit	Required	Default value	Description
feeder	Feeder	-	no	-	The information for an injection with only one side
feeder1	Feeder	-	no	-	The information on side 1 for a Branch or three-winding transformer
feeder2	Feeder	-	no	-	The information on side 2 for a Branch or three-winding transformer
feeder3	Feeder	-	no	-	The information on side 3 for a three-winding transformer

Depending on the type of connectable, the associated attribute should be filled out: only feeder for injections, feeder1 and feeder2 for branches and feeder1, feeder2 and feeder3 for three-winding transformers.

The attributes of the `Feeder` class, containing all the position information of the connectable relative to one voltage level, are:

Attribute	Type	Unit	Required	Default value	Description
name	String	-	yes	-	The name associated to the feeder. It is the name that will be displayed on diagrams.
order	Integer	-	no	-	The position of the connectable relative to the others
direction	`ConnectablePositic Direction`	-	no	-	The direction of the connectable relative to its connected busbar section. Can be `TOP` or `BOTTOM`

2.3.6 Coordinated reactive control

Some generators can be coordinated to control reactive power in a point of the network. This extension is used to configure the percentage of reactive-coordinated control that comes from a generator. This extension is attached to a *generator*.

Attribute	Type	Unit	Required	Default value	Description
QPercent	percent [0-100]	-	yes	-	The reactive control percent of participation

Here is how to add a coordinated reactive control extension to a generator:

```
generator.newExtension(CoordinatedReactiveControlAdder.class)
    .withQPercent(40)
    .add();
```

Please note that the sum of the $qPercent$ values of the generators coordinating the same point of the network must be 100.

This extension is provided by the `com.powsybl:powsybl-iidm-extensions` module.

2.3.7 Discrete measurements

This extension is used to store discrete measurements (such as tap positions, switch positions, etc.) collected in substations.

Attribute	Type	Unit	Required	Default value	Description
discreteMeasurements	Collection	-	no	-	Contains a collection of DiscreteMeasurement objects

The DiscreteMeasurement class characteristics are the following:

Attribute	Type	Unit	Required	Default value	Description
id	String	-	no	-	The ID of the discrete measurement if it exists
type	DiscreteMeasurement.Type	-	no	-	The type of discrete measurement (TAP_POSITION, SWITCH_POSITION, SHUNT_COMPENSATOR_SECTION, OTHER)
tapCha	DiscreteMeasurement.TapChanger	-	no	-	The tap changer the discrete measurement is applied on (null if the measurement is not applied to a tap changer)
properties	Map<String, String>	-	no	-	The properties (name and value) associated with the discrete measurement
valueType	DiscreteMeasurement.ValueType	-	no	-	The discrete measurement value type (BOOLEAN, INT or STRING)
value	Object	-	no	-	The discrete measurement value
valid	boolean	-	no	-	The validity status (if true, the discrete measured value cannot be null)

2.3.8 ENTSO-E area

TODO

2.3.9 HVDC angle droop active power control

This is an extension dedicated to DC line in order to model AC emulation. For a VSC converter station operating in AC emulation, its active power setpoint is given by

$$P = P0 + k\ (ph1 - ph2)$$

Attribute	Type	Unit	Required	Default value	Description
P0	float	MW	yes	-	P0 in the equation
droop	float	MW by degree	yes	-	k in the equation
enabled	boolean	-	yes	-	if the AC emulation is active or not

2.3.10 HVDC operator active power range

This extension enables to replace the operational limits of a DC line in AC emulation. In that case, the VSC converter stations min active power and max active power are not used.

2.3.11 Generator ENTSO-E category

This extension enables to store the entso-e category code of a given generator, which represents a combination of fuel and type. For instance category 26 is for hydro run-of-river generating units. It is mainly used in TYNDP works. This extension is attached to a *generator*.

Attribute	Type	Unit	Required	Default value	Description
code	int	-	yes	-	The entso-e category

Here is how to add a generator entsoe-e category extension to a generator:

```
generator.newExtension(GeneratorEntsoeCategoryAdder.class)
    .withCode(4)
    .add();
```

2.3.12 Generator startup

This extension contains the information related to the startup of a generator.

Attribute	Type	Unit	Required	Default value	Description
forcedOutageRate	double	-	no	-	Rate of forced unavailability
marginalCost	double	-	no	-	Cost to increase the production of one unit (in general one MW)
plannedActivePowerSetpoint	double	MW	no	-	Active power target planned by the market
plannedOutageRate	double	-	no	-	Rate of planned unavailability
startupCost	double	-	no	-	Cost to start the generator

This extension is provided in the `com.powsybl:powsybl-iidm-extensions` module.

To add this extension to a generator, the code to be used is:

```
generator.newExtension(GeneratorStartupAdder.class)
        .withPlannedActivePowerSetpoint(600.0)
        .withStartupCost(5.0)
        .withMarginalCost(10.0)
        .withPlannedOutageRate(0.8)
        .withForcedOutageRate(0.7)
        .add();
```

2.3.13 Generator short-circuit

This extension models the generator data used for short-circuit calculations. Depending on the type of short-circuit study to be performed, either the transient or the sub-transient reactance should be filled. The reactance of the step-up transformer should be filled if the generator has a transformer that is not directly modeled in the network. This extension should be used in addition to the *GeneratorFortescue extension* for asymmetrical short-circuit calculations.

Attribute	Type	Unit	Required	Default value	Description
directTransX (X'd)	double		yes	-	Direct transient reactance of the generator
directSubtransX (X"d)	double		no	-	Direct sub-transient reactance of the generator
stepUpTransformerX	double		no	-	Reactance of the step-up transformer

This extension is provided in the `com.powsybl:powsybl-iidm-extensions` module.

To add this extension to a generator, the code to be used is:

```
generator.newExtension(GeneratorShortCircuitAdder.class)
    .withDirectTransX(20)
    .withDirectSubtransX(14)
    .withStepUpTransformerX(10)
    .add();
```

2.3.14 Battery short-circuit

This extension models the battery data used for short-circuit calculations. Depending on the type of short-circuit study to be performed, either the transient or the sub-transient reactance should be filled. The step-up transformer reactance should be filled if the battery has a transformer that is not directly modeled on the network.

Attribute	Type	Unit	Required	Default value	Description
directTransX (X'd)	double		yes	-	Direct transient reactance of the battery
directSubtransX (X"d)	double		no	-	Direct sub-transient reactance of the battery
stepUpTransformerX	double		no	-	Reactance of the step-up transformer

This extension is provided in the `com.powsybl:powsybl-iidm-extensions` module.

To add this extension to a battery, the code to be used is:

```
battery.newExtension(BatteryShortCircuitAdder.class)
    .withDirectTransX(20)
    .withDirectSubtransX(14)
    .withStepUpTransformerX(10)
    .add();
```

2.3.15 Identifiable short-circuit

This extension models the maximum and minimum short-circuit current admissible for any identifiable.

Attribute	Type	Unit	Required	Default value	Description
ipMin	double	A	no	-	The minimum admissible current
ipMax	double	A	yes	-	The maximum admissible current

This extension is provided in the `com.powsybl:powsybl-iidm-extensions` module.

To add this extension to a bus, for example, the code to be used is:

```
bus.newExtension(IdentifiableShortCircuitAdder.class)
    .withIpMin(3000)
    .withIpMax(10000)
    .add();
```

The code is similar for every identifiable.

2.3.16 Generator Fortescue

This extension models the homopolar generator data to be used for asymmetrical short-circuit calculations.

Attribute	Type	Unit	Required	Default value	Description
rz	double		no	-	The zero-sequence resistance of the generator
xz	double		no	-	The zero-sequence reactance of the generator
rn	double		no	-	The negative-sequence resistance of the generator
xn	double		no	-	The negative-sequence reactance of the generator
grounded	boolean	-	no	false	Indicates if the generator is earthed
groundingR	double		no	0	If the generator is earthed, the resistance part of the impedance to the ground
groundingX	double		no	0	If the generator is earthed, the reactance part of the impedance to the ground

This extension is provided in the `com.powsybl:powsybl-iidm-extensions` module.

2.3.17 Line Fortescue

This extension models the homopolar line data to be used for asymmetrical short-circuit calculations.

Attribute	Type	Unit	Required	Default value	Description
rz	double		no	-	The zero-sequence resistance of the line
xz	double		no	-	The zero-sequence reactance of the line
openPhaseA	boolean	-	no	false	Indicates if the phase A of the line is open
openPhaseB	boolean	-	no	false	Indicates if the phase B of the line is open
openPhaseC	boolean	-	no	false	Indicates if the phase C of the line is open

This extension is provided in the `com.powsybl:powsybl-iidm-extensions` module.

2.3.18 Two-winding Transformer Fortescue

This extension models the homopolar two-winding transformer data to be used for asymmetrical short-circuit calculations.

Attribute	Type	Unit	Required	Default value	Description
rz	double		no	-	The zero-sequence resistance of the two-winding transformer
xz	double		no	-	The zero-sequence reactance of the two-winding transformer
freeFluxes	boolean	-	no	true	If set to true, then the magnetizing impedance is considered as infinite
connectionType1	WindingConnectionType	-	no	WindingConnectionType.DELTA	The connection type of the winding 1
connectionType2	WindingConnectionType	-	no	WindingConnectionType.Y_GROUNDED	The connection type of the winding 2
groundingR1	double		no	0	If the winding on side 1 is connected to the earth, the resistance part of the impedance to the ground
groundingX1	double		no	0	If the winding on side 1 is connected to the earth, the reactance part of the impedance to the ground
groundingR2	double		no	0	If the winding on side 2 is connected to the earth, the resistance part of the impedance to the ground
groundingX2	double		no	0	If the winding on side 2 is connected to the earth, the reactance part of the impedance to the ground

This extension is provided in the `com.powsybl:powsybl-iidm-extensions` module.

2.3.19 Three-winding Transformer Fortescue

This extension models the homopolar three-winding transformer data to be used for asymmetrical short-circuit calculations. The data is described by leg.

The attributes of the extension are structured as follows:

Attribute	Type	Unit	Required	Default value	Description
leg1	LegFortescue	-	no	-	The data on the leg 1 of the three-winding transformer
leg2	LegFortescue	-	no	-	The data on the leg 2 of the three-winding transformer
leg3	LegFortescue	-	no	-	The data on the leg 3 of the three-winding transformer

With LegFortescue being defined as follows:

Attribute	Type	Unit	Required	Default value	Description
rz	double		no	-	The zero-sequence resistance of the leg
xz	double		no	-	The zero-sequence reactance of the leg
freeFlu	boolean	-	no	true	If set to true, then the magnetizing impedance is considered as infinite
connectionType	WindingConnectionType	-	no	WindingConnectionType.DELTA for the first and the third leg, WindingConnectionType.Y_GROUNDED for the second	The connection type of the leg
groundingR	double		no	0	If the leg is connected to the earth, the resistance part of the impedance to the ground
groundingX	double		no	0	If the leg is connected to the earth, the reactance part of the impedance to the ground

This extension is provided in the `com.powsybl:powsybl-iidm-extensions` module.

2.3.20 Injection observability

This extension models injection flow observability, obtained after a state estimation.

Attribute	Type	Unit	Required	Default value	Description
quality P	ObservabilityQuality	MW	no	-	The observability quality of active power
quality Q	ObservabilityQuality	MVar	no	-	The observability quality of reactive power

Observability quality

This extension contains the sub-object `ObservabilityQuality`.

Attribute	Type	Unit	Required	Default value	Description
standard deviation	double	MW or MVar	yes	-	The standard deviation
redundant	boolean	-	yes	-	Indicates if the value is confirmed by redundancy

This extension is provided by the `com.powsybl:powsybl-iidm-extensions` module.

2.3.21 Line Position

This extension is attached to a Line and is used to store the geographical coordinates of the Line. The coordinates are stored using latitude and longitude. The extension consists of a list of coordinates that can be used to trace the line on a map.

Attribute	Type	Unit	Required	Default value	Description
coordinates	list<Coordinate>	-	yes	-	The list of points coordinates forming the line

Example of code to get the coordinates of a line :

```
List<Coordinate> stationCoordinates = network.getLine("L1").getExtension(LinePosition.
↪class)
    .getCoordinates();
```

This extension is provided in the module com.powsybl:powsybl-iidm-extensions.

When adding the extension, the LinePositionAdder extension should be used.

Example of code to add the extension:

```
Line line = network.getLine("L1");
line.newExtension(LinePositionAdder.class)
    .withCoordinates(List.of(new Coordinate(48, 2), new Coordinate(48.1, 2.1)))
    .add();
```

2.3.22 Load asymmetrical

A balanced load is described by its active power setpoint $P0$ and its reactive power setpoint $Q0$. This extension is used to describe the power asymmetry for each ABC phase. In the three-phase representation, the complex power injected at a bus i is constant for each phase and represented by three complex values:

$$S_{Ai_{Load}} = S_A = P_A + j.Q_A \tag{2.1}$$
$$S_{Bi_{Load}} = S_B = P_B + j.Q_B \tag{2.2}$$
$$S_{Ci_{Load}} = S_C = P_C + j.Q_C \tag{2.3}$$
$$\tag{2.4}$$

But for a balanced load flow, the constant power load P and Q refer to the positive sequence load. Given that, in balanced conditions, the load for zero and negative sequences should always be zero. However, in real life, power loads are better defined in the ABC three-phase representation. The load extension addresses this issue keeping the default behavior for balanced conditions.

Balanced load flow conditions:

In balanced conditions given the load at bus:

$$S_{1i_{Load}} = P_{Load} + j.Q_{Load}$$

We must verify:

$$0 = -S_{1i_{Load}} + \sum_{j=\delta(i)} S_{1ij}$$

Unbalanced load flow conditions:

We must take into account that many loads are still balanced and information related to balanced loads is sufficient. The extension proposes a delta approach where unbalances are expressed in the extension. Supposing that:

$$\Delta P_{Ai_{Load}}, \Delta Q_{Ai_{Load}}, \Delta P_{Bi_{Load}}, \Delta Q_{Bi_{Load}}, \Delta P_{Ci_{Load}}, \Delta Q_{Ci_{Load}} \tag{2.5}$$

are provided in input through the extension. The three-phase power values used as inputs of an unbalanced load flow calculation are:

$$S_{Ai_{Load}} = (P_{Load} + \Delta P_{Ai_{Load}}) + j.(Q_{Load} + \Delta Q_{Ai_{Load}}) \tag{2.6}$$
$$S_{Bi_{Load}} = (P_{Load} + \Delta P_{Bi_{Load}}) + j.(Q_{Load} + \Delta Q_{Bi_{Load}}) \tag{2.7}$$
$$S_{Ci_{Load}} = (P_{Load} + \Delta P_{Ci_{Load}}) + j.(Q_{Load} + \Delta Q_{Ci_{Load}}) \tag{2.8}$$
$$\tag{2.9}$$

As a consequence, if no extension provided for the load, the unbalanced load flow will use in input:

$$S_{Ai_{Load}} = P_{Load} + j.Q_{Load} \tag{2.10}$$
$$S_{Bi_{Load}} = P_{Load} + j.Q_{Load} \tag{2.11}$$
$$S_{Ci_{Load}} = P_{Load} + j.Q_{Load} \tag{2.12}$$
$$\tag{2.13}$$

Attribut	Type	Unit	Required	Default value	Description
deltaP	double	MW	No	0	The unbalanced part of the active power setpoint at phase A (balanced parts for each phase are described by its active power setpoint $P0$ and its reactive power setpoint $Q0$)
deltaQ	double	MVar	No	0	The unbalanced part of the reactive power setpoint at phase A
deltaP	double	MW	No	0	The unbalanced part of the active power setpoint at phase B
deltaQ	double	MVar	No	0	The unbalanced part of the reactive power setpoint at phase B
deltaP	double	MW	No	0	The unbalanced part of the active power setpoint at phase C
deltaQ	double	MVar	No	0	The unbalanced part of the reactive power setpoint at phase C

Here is how to add a load detail extension to a load:

```
load.newExtension(LoadAsymmetricalAdder.class)
        .withDeltaPa(-1)
        .withDeltaQa(1)
        .withDeltaPb(-2)
        .withDeltaQc(2)
        .add();
```

This extension is provided by the `com.powsybl:powsybl-iidm-extensions` module.

2.3.23 Load detail

A load is described by its active power setpoint $P0$ and its reactive power setpoint $Q0$. This extension is used to detail :

- In the total amount of active power what is fixed and what is time-dependent (also called variable). The time-dependent part can be adjusted for production equals consumption.
- In the total amount of reactive power what is fixed and what is time-dependent (also called variable).

Attribute	Type	Unit	Required	Default value	Description
variableActivePower	double	MW	yes	-	The part of the active power setpoint that is considered variable
fixedActivePower	double	MW	yes	-	The part of the active power setpoint that is considered constant
variableReactivePower	double	MVar	yes	-	The part of the reactive power setpoint that is considered variable
fixedReactivePower	double	MVar	yes	-	The part of the reactive power setpoint that is considered constant

Here is how to add a load detail extension to a load:

```
load.newExtension(LoadDetailAdder.class)
    .withVariableActivePower(40)
    .withFixedActivePower(20)
    .withVariableReactivePower(5)
    .withFixedReactivePower(2)
    .add();
```

All of this extension's attributes are multi-variants: they can vary from one variant to another.

This extension is provided by the `com.powsybl:powsybl-iidm-extensions` module.

2.3.24 Measurements

This extension is used to store measurements collected in substations.

Attribute	Type	Unit	Required	Default value	Description
measurements	Collection	-	no	-	Contains a collection of Measurement objects

The Measurement class characteristics are the following:

Attribute	Type	Unit	Required	Default value	Description
id	String	-	no	-	The ID of the measurement if it exists
type	Measurement.Type	-	no	-	The type of measurement (ANGLE, ACTIVE_POWER, VOLTAGE etc.)
properties	Map<String, String>	-	no	-	The properties (name and value) associated with the measurement
value	double	-	no	-	The measurement value
standardDeviation	double	-	no	-	The standard deviation (NaN if not specified)
valid	boolean	-	no	-	The validity status (if true, the measured value cannot be NaN)
side	ThreeSides	-	no	-	The equipment side associated to the measurement

2.3.25 Operating status

This is an extension of `Identifiable`, but it is restricted to some identifiable types: busbar sections, all branches, three-winding transformers, HVDC line and a boundary line. The status could be:

- `IN_OPERATION`: equipment in service.
- `PLANNED_OUTAGE`: outage due to an unscheduled putting out of service of the equipment.
- `FORCED_OUTAGE`: outage due to a programmed taking out of service of the equipment.

2.3.26 Reference Priority

This extension is attached to a Generator, or a BusBarSection or a Load and is used to define the angle reference bus of a power flow calculation, i.e. which bus will be used with a zero-voltage angle. Use this extension before a computation to force the reference bus selection. The support of this feature by Load Flow implementations may vary. For example, the OpenLoadFlow implementation today supports Reference Priorities on generators only when this feature is activated.

The reference bus is defined through the terminal of the equipment and an integer specifying the reference priority. 0 means "do not use as reference", 1 is "highest priority", 2 "second priority", etc.

Attribute	Type	Unit	Required	Default value	Description
Terminal	`Terminal`	-	yes	-	The reference terminal
Priority	`Integer`	-	yes	0	The reference priority

```
ReferencePriority.set(generator, 1);

int priority = ReferencePriority.get(generator); // note: returns zero if none defined
```

This extension is provided by the `com.powsybl:powsybl-iidm-api` module.

2.3.27 Reference Terminals

This extension is attached to a Network and is used to define the angle references of a Power Flow solution. The support of this feature by Load Flow implementations may vary. For example, the OpenLoadFlow implementation today supports writing to the Network the terminals of the reference generators chosen via the *Reference Priority extension*.

The reference bus is defined through the terminal of the equipment and an integer specifying the reference priority. 0 means "do not use as reference", 1 is "highest priority", 2 "second priority", etc.

Attribute	Type	Unit	Required	Default value	Description
terminals	Set<Terminal>	-	yes	-	The reference terminals

```
Set<Terminal> referenceTerminals = ReferenceTerminals.getTerminals(network);

ReferenceTerminals.reset(network);

ReferenceTerminals.add(terminal);
```

This extension is provided by the `com.powsybl:powsybl-iidm-api` module.

2.3.28 Remote reactive power control

This extension is used for generators with a remote reactive control.

Attribute	Type	Unit	Required	Default value	Description
enabled	boolean	-	yes	-	If the reactive remote control is activated of not
targetQ	double	MVar	yes	-	The targetQ at remote regulating terminal
regulatingTerminal	Terminal	-	yes	-	The regulating terminal

2.3.29 Slack terminal

This extension is attached to a *voltage level* and is used to define the slack bus of a power flow calculation i.e. which bus will be used to balance the active and reactive power in load flow analysis. Use this extension before a computation to force the slack bus selection. You should enable default load flow parameter *readSlackBus*. Use this extension after a computation to attach to the network the slack bus that has been selected by the load flow engine (one by connected component). You should enable default load flow parameter *writeSlackBus*.

The slack bus is defined through the terminal of a connectable that belongs to the bus. It is totally allowed to define a disconnected terminal as slack as the connectable could be reconnected during a grid study.

Attribute	Type	Unit	Required	Default value	Description
Terminal	Terminal	-	yes	-	The slack terminal

```
SlackTerminal.attach(bus);
```

This extension is provided by the `com.powsybl:powsybl-iidm-api` module.

2.3.30 Substation Position

This extension is attached to a Substation and is used to store the geographical coordinates of the Substation. The coordinates are stored using latitude and longitude.

Attribute	Type	Unit	Required	Default value	Description
coordinate	Coordinate	-	yes	-	The latitude and longitude of the substation

Example of code to get the coordinates of a substation :

```
Coordinate stationCoordinate = network.getSubstation("P1").
↪getExtension(SubstationPosition.class)
    .getCoordinate();
```

This extension is provided in the module `com.powsybl:powsybl-iidm-extensions`.

When adding the extension, the SubstationPositionAdder extension should be used.

Example of code to add the extension:

```
Substation station = network.getSubstation("P1");
station.newExtension(SubstationPositionAdder.class)
        .withCoordinate(new Coordinate(48, 2))
        .add();
```

2.3.31 Three-winding transformer phase angle clock

This extension is used to model the Vector Group of a three-winding transformer. The phase angle clock could be modeled at leg 2, leg 3 or both legs 2 and 3 and of a three-winding transformer (network side). The voltage phase angle displacement is represented with clock hours. The valid values are `0` to `11`. This extension is attached to a *three-winding transformer*.

Attribute	Type	Unit	Required	Default value	Description
PhaseAngleClockLeg2	int [0-11]	hours	yes	-	The voltage phase angle displacement at leg 2
PhaseAngleClockLeg3	int [0-11]	hours	yes	-	The voltage phase angle displacement at leg 3

```
transformer.newExtension(ThreeWindingsTransformerPhaseAngleClock.class)
    .withPhaseAngleClockLeg2(10)
    .withPhaseAngleClockLeg3(1)
    .add();
```

This extension is provided by the `com.powsybl:powsybl-iidm-extensions` module.

2.3.32 Three-winding transformer to be estimated

This extension is used to indicate if a three-winding transformer tap changer is to be estimated during a state estimation, i.e., if its tap position should be an output of the state estimation.

- The three-winding transformer model offers the possibility to have up to 3 ratio tap changers and up to 3 phase tap changers. Each tap changer can be estimated or not.
- If a tap changer is not to be estimated, it should not be changed during a state estimation (its tap position is merely an input of the state estimation).

Attribute	Type	Unit	Required	Default value	Description
NAME	String	-	yes	threeWindingsTransformerToBeEstimated	Name of the extension

Example of code to get the status of the n°1 phase tap changer:

```
3wt.getExtension(ThreeWindingsTransformerToBeEstimated.class).
↪shouldEstimatePhaseTapChanger1();
```

This extension is provided in the module `com.powsybl:powsybl-iidm-extensions`.

When adding the extension, the ThreeWindingsTransformerToBeEstimatedAdder extension should be used.

Example of code to add the extension:

```
transformer.newExtension(ThreeWindingsTransformerToBeEstimatedAdder.class)
        .withRatioTapChanger1Status(true)
        .add();
```

2.3.33 Two-winding transformer phase angle clock

This extension is used to model the Vector Group of a two-winding transformer. The phase angle clock is modeled at side 2 of a two-winding transformer. The voltage phase angle displacement is represented with clock hours. The valid values are 0 to 11. This extension is attached to a *two-winding transformer*.

Attribute	Type	Unit	Required	Default value	Description
PhaseAngleClock	int [0-11]	hours	yes	-	The voltage phase angle displacement

```
transformer.newExtension(TwoWindingsTransformerPhaseAngleClockAdder.class)
    .withPhaseAngleClock(3)
    .add();
```

This extension is provided in the module `com.powsybl:powsybl-iidm-extensions`.

2.3.34 Two-winding transformer to be estimated

This extension is used to indicate if a two-winding transformer tap changer is to be estimated during a state estimation, i.e., if its tap position should be an output of the state estimation.

- A two-winding transformer has a ratio tap changer and/or a phase tap changer. Each tap changer can be estimated or not.
- If a tap changer is not to be estimated, it should not be changed during a state estimation (its tap position is merely an input of the state estimation).

Attribute	Type	Unit	Required	Default value	Description
NAME	String	-	yes	twoWindingsTransformerToBeEstimated	Name of the extension

Example of code to get the status of the ratio tap changer:

```
2wt.getExtension(TwoWindingsTransformerToBeEstimated.class).
↪shouldEstimateRatioTapChanger();
```

This extension is provided in the module `com.powsybl:powsybl-iidm-extensions`.

When adding the extension, the TwoWindingsTransformerToBeEstimatedAdder extension should be used.

Example of code to add the extension:

Example of code:

```
transformer.newExtension(TwoWindingsTransformerToBeEstimatedAdder.class)
        .withPhaseTapChangerStatus(true)
        .add();
```

2.3.35 Voltage per reactive power control

This extension is used to model voltage control of static VAR compensators. This extension is attached to a *static VAR compensator*.

Attribute	Type	Unit	Required	Default value	Description
Slope	double	kV per MVar	yes	-	The sensibility of the voltage with respect to reactive power

When this extension is present and the slope greater than zero, the reactive output of the static VAR compensator is defined by:

$$Q = \frac{VoltageSetpoint - V}{slope}$$

where V is the voltage at regulating terminal and $VoltageSetpoint$ the target value in voltage given as attribute in a static VAR compensator.

Here is how to add a voltage per reactive power control extension to a static VAR compensator:

```
svc.newExtension(VoltagePerReactivePowerControlAdder.class)
    .withSlope(0.5)
    .add();
```

This extension is provided by the `com.powsybl:powsybl-iidm-extensions` module.

2.4 Adders by copy

Adders by copy allow users to quickly and easily create objects by taking existing ones as "models" and then changing what needs to be changed before adding the object to the network.

At the moment, adders by copy are available for the objects listed below.

2.4.1 Lines

You can create a `Line` object from another existing `Line` object. Some characteristics of the new *line* are pre-filled with the characteristics of the "model" *line*:

- R
- X
- G1 and G2

- B1 and B2
- The *voltage levels* at each side of the *line*

Operational limits are also copied from the model line.

Other attributes of the new *line* may (or should, for mandatory ones) be filled up by the user.

NB: it is mandatory for the user to fill up the `id` attribute before calling the `add()` method or else the code will throw an exception.

In the following example, the id, the buses and the connectable buses are set by the user before the new object is added to the network through the `add()` method.

```
network.newLine(network.getLine("existingLineId"))
        .setId("newLineId")
        .setBus1(newLineBus1)
        .setBus2(newLineBus2)
        .setConnectableBus1(newLineBus1)
        .setConnectableBus2(newLineBus2)
        .add();
```

NB : the pre-filled characteristics could also be modified if needed.

2.4.2 Two-winding transformers

You can create a *TwoWindingsTransformer* object from another existing *TwoWindingsTransformer* object. Some characteristics of the new *two-winding transformer* are pre-filled with the characteristics of the "model" *two-winding transformer*:

- R
- X
- G
- B
- ratedU1
- ratedU2

Operational limits are also copied from the existing transformer.

Other attributes may (or should, for mandatory ones) be filled up by the user.

NB: it is mandatory for the user to fill up the `id` attribute before calling the `add()` method or else the code will throw an exception.

In the following example, the id, the buses and the ratedS attribute are set by the user before the new object is added to the network through the `add()` method.

```
TwoWindingsTransformer transformer2 = substation.newTwoWindingsTransformer(transformer1)
        .setId("twt2")
        .setRatedS(7.0)
        .setBus1("busA")
        .setBus2("busB")
        .add();
```

NB : the pre-filled characteristics could also be modified if needed.

2.4.3 Ratio tap changers

You can create a *RatioTapChanger* object from another existing *RatioTapChanger* object. Some characteristics of the new *RatioTapChanger* are pre-filled with the characteristics of the "model" *RatioTapChanger*:

- The regulation terminal
- The regulation mode
- The regulation value
- The`loadTapChangingCapabilities` value
- The `targetV` value
- The `lowTapPosition` value
- The `tapPosition` value
- The `regulating` value
- The `targetDeadBand` value

Besides that, the following characteristics for all the `RatioTapChangerStep` steps of the existing *RatioTapChanger* are also copied to create the new object:

- rho
- b
- g
- x
- r

In the following example, a new ratio tap changer `newRatioTapChanger` is created from the `existingRatioTapChanger` object and added to a two-winding transformer through the `add()` method.

```
RatioTapChanger newRatioTapChanger = network.getTwoWindingsTransformer("transformerId")
        .newRatioTapChanger(existingRatioTapChanger)
        .add();
```

NB : the pre-filled characteristics could also be modified if needed.

2.4.4 Phase tap changers

You can create a *PhaseTapChanger* object from another existing *PhaseTapChanger* object. Some characteristics of the new *PhaseTapChanger* are pre-filled with the characteristics of the "model" *PhaseTapChanger*:

- The regulation terminal
- The regulation mode
- The regulation value
- The `lowTapPosition` value
- The `tapPosition` value
- The `regulating` value
- The `targetDeadBand` value

Besides that, the following characteristics for all the `PhaseTapChangerStep` steps of the existing *PhaseTapChanger* are also copied to create the new object:

- alpha
- rho
- b
- g
- x
- r

In the following example, a new phase tap changer `newPhaseTapChanger` is created from the `existingPhaseTapChanger` object and added to a two-winding transformer through the `add()` method.

```
RatioTapChanger newPhaseTapChanger = network.getTwoWindingsTransformer("transformerId")
        .newPhaseTapChanger(existingPhaseTapChanger)
        .add();
```

NB : the pre-filled characteristics could also be modified if needed.

2.4.5 Reactive capability curve

You can create a *ReactiveCapabilityCurve* object from another existing *ReactiveCapabilityCurve* object. The points of the new *ReactiveCapabilityCurve* are copied from the points of the "model" *ReactiveCapabilityCurve*.

In the following example, a reactive capability curve is created from the `existingReactiveCapabilityCurve` and added to a *Generator* object:

```
generator.newReactiveCapabilityCurve(existingReactiveCapabilityCurve)
        .add();
```

NB : other points could be added to the new object if needed.

2.4.6 Loading limits

You can add limits to an *OperationalLimitsGroup* by copying existing limits. The following elements are copied:

- The permanent limit value;
- For all the temporary limits: their name, value, acceptable duration and whether they are flagged as fictitious.

In the following example, `CurrentLimits`, `ActivePowerLimits` and `ApparentPowerLimits` are added to an *OperationalLimitsGroup* object from existing objects:

```
operationalLimitsGroup.newCurrentLimits(existingCurrentLimits).add();
operationalLimitsGroup.newActivePowerLimits(existingActivePowerLimits).add();
operationalLimitsGroup.newCurrentLimits(existingApparentPowerLimits).add();
```

It is also possible to use a higher-level function to copy operational limits from an existing *Line* `existingLine` to a new *Line* `otherLine`:

```
LoadingLimitsUtil.copyOperationalLimits(existingLine, otherLine);
```

2.5 Going further

TODO

Powsybl features are strongly based on an internal grid model initially developed under the iTesla project, a research project funded by the European Union 7th Framework programme (FP7). The grid model is known as `iidm` (iTesla Internal Data Model). One of the iTesla outputs was a toolbox designed to support the decision-making process of power system operation from two-days ahead to real time. The `iidm` grid model was at the center of the toolbox.

The equipment of a substation (busbar sections, switches, buses, loads, generators, shunt compensators, static VAR compensators, HVDC converters stations, etc.) is grouped in voltage levels. Transformers present in a substation connect its different voltage levels. Transmission lines (AC and DC) connect the substations.

To build an electrical network model, the common way is to define the substations first, then to define their voltage levels. But for some specific cases, it is also possible to create voltage levels without a substation.

The grid model allows a full representation of the substation connectivity where all the switching devices and bus-bar sections are defined, this topology is called node/breaker view. Automated topology calculation allows for the calculation of the network bus/breaker view as well as the network bus view.

Different states of the network can be efficiently stored together with the power system model. The set of attributes that define a given state of the network (both steady state hypothesis and state variables) are collectively organized in variants. The user can create and remove variants as needed. Setting and getting variant dependent attributes on network objects use the current variant.

A set of networks can be merged together in a single network. The initial subnetworks are kept and can be easily retrieved or detached if needed.

Almost all the elements modeled in the network are identified through a unique `id`, and optionally described by a `name` that is easier to interpret for a human. Almost all components can be *extended* by the user to incorporate additional structured data.

Adders by copy can be used to quickly create similar objects in the network.

CHAPTER

THREE

GRID FEATURES

This section is dedicated to the description of the main network handling features in PowSyBl.

3.1 Import post-processor

The import post-processor is a feature that allows to do automatic modifications or simulations, just after a case is converted to an *IIDM* network. These post-processors rely on the plugin mechanism of PowSyBl meaning that they are discovered at runtime. To enable one or more post-processors, the `postprocessors` property of the `import` module must be defined in the configuration file. Note that if you configure several post-processors, they are executed in the declaration order, like a pipeline: TODO: insert a picture

PowSyBl provides 2 different implementations of post-processors:

- *Groovy*: to execute a groovy script
- *LoadFlow*: to run a power flow simulation

3.1.1 Groovy post-processor

This post-processor executes a groovy script, loaded from a file. The script can access to the network and the computation manager using the variables `network` and `computationManager`. To use this post-processor, add the `com.powsybl:powsybl-iidm-scripting` dependency to your classpath, and configure both `import` and `groovy-post-processor` modules:

YAML configuration:

```
import:
    postProcessors:
        - groovyScript

groovy-post-processor:
    script: /path/to/the/script
```

XML configuration:

```
<import>
    <postProcessors>groovyScript</postProcessors>
</import>
<groovy-post-processor>
    <script>/path/to/the/script</script>
</groovy-post-processor>
```

Note: the `script` property is optional. If it is not defined, the `import-post-processor.groovy` script from the PowSyBl configuration folder is used.

Example

The following example prints meta-information from the network:

```
println "Network " + network.getId() + " (" + network.getSourceFormat()+ ") is imported"
```

3.1.2 LoadFlow post-processor

Mathematically speaking, a *load flow* result is fully defined by the complex voltages at each node. The consequence is that most load flow algorithms converge very fast if they are initialized with voltages. As a result, it happens that load flow results include only voltages and not flows on branches. This post-processors computes the flows given the voltages. The equations (Kirchhoff law) used are the same as the one used in the *load flow validation* to compute P_1^{calc}, Q_1^{calc}, P_2^{calc}, Q_2^{calc} for branches and P_3^{calc}, Q_3^{calc} in addition for three-winding transformers.

To use this post-processor, add the `com.powsybl:powsybl-loadflow-results-completion` to your classpath and enable it setting the `postProcessors` property of the `import` module.

YAML configuration:

```
import:
    postProcessors:
        - loadflowResultsCompletion
```

XML configuration:

```
<import>
    <postProcessors>loadflowResultsCompletion</postProcessors>
</import>
```

Note: This post-processor relies on the load flow results completion module.

3.1.3 Geographical data import post-processor

One way to add geographical positions on a network is to use the import post-processor named GeoJsonAdderPostProcessor, that will automatically add the *LinePosition* and *SubstationPosition* extensions to the network model.

This processor uses geographical position data formatted in two JSON files, as it can be obtained on the websites Open Infra Map or Open Street Map.

To use this import post-processor, add the `com.powsybl:powsybl-iidm-geodata` to your classpath and enable it setting the `postProcessors` and `geo-json-importer-post-processor` modules :

YAML configuration:

```
import:
  postProcessors:
    - geoJsonImporter

geo-json-importer-post-processor:
  substations: /path/to/substations.geojson
  lines: /path/to/lines.geojson
```

XML configuration:

```
<import>
    <postProcessors>geoJsonImporter</postProcessors>
</import>
```

(continues on next page)

(continued from previous page)

```
<geo-json-importer-post-processor>
    <substations>/path/to/substations.geojson</substations>
    <lines>/path/to/lines.geojson</lines>
</geo-json-importer-post-processor>
```

The paths to the different files can be absolute paths or paths relative to the directory where your command is launched.

3.1.4 Going further

- Create a post-processor: Learn how to implement your own post-processor

3.2 Network reduction

This module is used to extract a portion of a network on an area of interest defined by the user.

3.2.1 Define an area of interest

The network reduction is relying on a `NetworkPredicate` instance, to define an area of interest (i.e., a list of equipments to keep in the network after the reduction). The equipments outside this area are removed, and the lines, transformers and HVDC lines connecting voltage levels inside and outside this area will be replaced by injections (loads or boundary lines, depending on the implementation).

Before doing the reduction, one has to define the area of interest, using the `com.powsybl.iidm.reducer.NetworkPredicate` interface. This interface declares two methods:

```
public interface NetworkPredicate {

    boolean test(Substation substation);

    boolean test(VoltageLevel voltageLevel);
}
```

These two methods must return `true` when the given parameter (a *substation* or a *voltage level*) is in the area of interest and should still be in the network after the reduction.

PowSyBl provides two implementations of this interface:

- the `IdentifierNetworkPredicate` implementation defines an area of interest using a list of voltage levels or substation IDs
- the `NominalVoltageNetworkPredicate` implementation defines an area of interest using a range of nominal voltages

You can also provide your own implementation.

By IDs

The `com.powsybl.iidm.reducer.IdentifierNetworkPredicate` class is an implementation of the `NetworkPredicate` interface that contains a set of substations' or voltage levels' IDs.

The `boolean test(Substation substation)` method returns true if the ID of the given substation is contained in the set of IDs, or at least one of the voltage levels IDs of the given substation is found in the set of IDs.

The `boolean test(VoltageLevel voltageLevel)` method returns true if the ID of the given voltage level or the ID of its substation is found in the set of IDs.

Examples

The following example shows how to create new `IdentifierNetworkPredicate` instances:

```
IdentifierNetworkPredicate p1 = new IdentifierNetworkPredicate(Arrays.asList("VL1", "VL2
↪", "S1"));

IdentifierNetworkPredicate p2 = new IdentifierNetworkPredicate(Collections.singleton("VL1
↪"));
```

There is also a more convenient way to create an instance of `IdentifierNetworkPredicate`, using the `of(String...)` static method:

```
NetworkPredicate p1 = IdentifierNetworkPredicate.of("VL1", "VL2", "S1");

NetworkPredicate p2 = IdentifierNetworkPredicate.of("VL1");
```

The list of voltage levels IDs can also be the result of a query:

```
NetworkPredicate p3 = new IdentifierNetworkPredicate(
        network.getVoltageLevelStream()
                .filter(vl -> vl.getNominalV() >= 225.0)
                .filter(vl -> vl.getNominalV() <= 400.0)
                .map(VoltageLevel::getId)
                .collect(Collectors.toList()));
```

By nominal voltages

The `com.powsybl.iidm.reducer.NominalVoltageNetworkPredicate` class is an implementation of the `NetworkPredicate` interface that contains two double values that define a range of nominal voltages.

The `boolean test(Substation substation)` method returns true if at least one of the voltage levels of the given substation has its nominal voltage inside the range.

The `boolean test(Voltage substation)` method returns true if the given voltage level has its nominal voltage inside the range.

Examples

The following example shows how to create a new `NominalVoltageNetworkPredicate` instance:

```
NetworkPredicate p1 = new NominalVoltageNetworkPredicate(0.0, Double.MAX_VALUE);

NetworkPredicate p2 = new NominalVoltageNetworkPredicate(225.0, 400.0);
```

3.2.2 Reduction

The `com.powsybl.iidm.reducer.NetworkReducer` interface provides one method, in charge of the reduction of the network:

```
public interface NetworkReducer {

    void reduce(Network network);
}
```

PowSyBl provides a default implementation of this interface, but you can provide your own.

Default implementation

The `com.powsybl.iidm.reducer.DefaultNetworkReducer` class is the PowSyBl implementation of the `NetworkReducer` interface.

It replaces the lines in the *border* group by *loads* or *boundary lines* depending on the *options*, the two-winding transformers and the HVDC lines by *loads*.

The three-winding transformers are replaced by a *load* if only one connected voltage level is kept. If two out of three connected voltage levels are kept, the third one is automatically added by the `DefaultNetworkReducer` to the voltage levels to keep.

Replacement

Replacements by loads

The load created in place of a branch has the same ID and name as the replaced branch. The type of the load is set as `FICTITIOUS` and its P_0 and Q_0 are set to the P and Q of the relevant terminal, depending on which side is kept in the network. If the branch is disconnected, P_0 and Q_0 are set to `NaN`. The connectivity information (node or bus depending on the voltage level topology) is kept. However, the operational limits and extensions from the original branch are not retained.

Replacements by boundary lines

The boundary line created in place of a line has the same ID and name as the replaced line. The resistance and reactance of the boundary line are equals to half of the resistance and reactance of the replaced line (we consider that the line is cut in the middle). The conductance and susceptance are set to the G_1 and B_1 or to G_2 and B_2, depending on which side is kept in the network.

The P_0 and Q_0 are set to the P and Q of the corresponding terminal, depending on which side is kept in the network. If the line is disconnected, P_0 and Q_0 are set to `NaN`. The connectivity information (node or bus depending on the voltage level topology) is kept. However, the operational limits and extensions from the original branch are not retained.

Options

The network reduction can be configured by passing a `com.powsybl.iidm.reducer.ReductionOptions` instance to the `DefaultNetworkReducer` constructor.

withBoundaryLines

This option defines whether the equipments in the *border* group are replaced by boundary lines or by loads. If this option is set to `false`, which is the default value, the equipments are exclusively replaced by loads.

Examples

The following example shows how to create a new `ReductionOptions` instance to do replacements by boundary lines.

```
ReductionOptions options = new ReductionOptions();
options.withBoundaryLines(true);
```

Note that the `ReductionOptions` class offers a fluent API that allows you to write code like this:

```
ReductionOptions options = new ReductionOptions()
        .withBoundaryLines(true);
```

Observers

The `com.powsybl.iidm.reducer.NetworkReducerObserver` is an interface that allows to be notified each time an `Identifiable` is removed or replaced. This interface provides several methods, one per `Identifiable` subclass managed by the `DefaultNetworkReducer` implementation. There are 2 types of events:

- a *replace* event, when an AC line, a two or three-winding transformer or an HVDC line is replaced by a load or a danging line
- a *remove* event, when a substation, a voltage level, a line, a two or three-winding transformer or an HVDC line is removed.

```
public interface NetworkReducerObserver {

    void substationRemoved(Substation substation);

    void voltageLevelRemoved(VoltageLevel voltageLevel);

    void lineReplaced(Line line, Injection injection);

    void lineRemoved(Line line);

    void transformerReplaced(TwoWindingsTransformer transformer, Injection injection);

    void transformerRemoved(TwoWindingsTransformer transformer);

    void transformerReplaced(ThreeWindingsTransformer transformer, Injection injection);

    void transformerRemoved(ThreeWindingsTransformer transformer);

    void hvdcLineReplaced(HvdcLine hvdcLine, Injection injection);

    void hvdcLineRemoved(HvdcLine hvdcLine);

}
```

PowSyBl provides a default implementation of the `NetworkReducerObserver` that does nothing. Use it as a base class of your own implementation.

3.2.3 Examples

The following Java code shows how to reduce a network, using the default implementation:

```
Network network = Importers.loadNetwork("network.xiidm");

NetworkReducer reducer = NetworkReducer.builder()
        .withNetworkPredicate(new NominalVoltageNetworkPredicate(225.0, 400.0))
        .withBoundaryLines(true)
        .build();
reducer.reduce(network);
```

Groovy scripting

This example shows how to do a network reduction, using the *run-script* command.

First, we need a groovy script to do the reduction:

```
import com.powsybl.iidm.network.Network;
import com.powsybl.iidm.reducer.IdentifierNetworkPredicate;
import com.powsybl.iidm.reducer.NetworkReducer;

network = loadNetwork(args[0])

reducer = NetworkReducer.builder()
        .withNetworkPredicate(IdentifierNetworkPredicate.of("P1"))
        .build()
reducer.reduce(network)

saveNetwork("XIIDM", network, null, args[1])
```

Then, we run the *groovy-script* command to apply the previous script to the `network.xiidm` file, and then export the modified network to the `network2.xiidm` file.

```
$> ./itools run-script --file extraction.groovy network.xiidm network2.xiidm
```

Import post-processor

This example shows how to automatically reduce networks when they are loaded, using the *groovy post-processors* with the same script as above. Note that the script will be applied each time a case file will be loaded. If you want to do it only once, use the *previous method*.

The script is a little different from the previous one:

```
import com.powsybl.iidm.reducer.IdentifierNetworkPredicate;
import com.powsybl.iidm.reducer.NetworkReducer;

reducer = NetworkReducer.builder()
        .withNetworkPredicate(IdentifierNetworkPredicate.of("P1"))
        .build()
reducer.reduce(network)
```

We have to configure the groovy post-processor in your configuration file:

```
import:
    postProcessors: groovyScript

groovy-post-processor:
    script: /home/user/network-reduction.groovy
```

For more information about the configuration of the groovy post-processor, please refer to this *documentation page*.

Then, we run the *convert-network* command:

```
$> ./itools convert-network --input-file /home/user/input.xiidm
--output-file /home/user/output.xiidm --output-format XIIDM
```

Observers

This example shows how to implement the NetworkReducerObserver and log information each time the equipment is replaced.

```
NetworkReducerObserver observer = new DefaultNetworkReducerObserver() {

    private static final Logger LOGGER = LoggerFactory.getLogger(NetworkReducerObserver.
↪class);

    @Override
    public void lineReplaced(Line line, Injection injection) {
        LOGGER.info("Line " + line.getId() + " has be replaced by a " + injection.
↪getType());
    }

    @Override
    public void transformerReplaced(TwoWindingsTransformer transformer, Injection␣
↪injection) {
        LOGGER.info("Transformer " + transformer.getId() + " has be replaced by a " +␣
↪injection.getType());
    }

    @Override
    public void transformerReplaced(ThreeWindingsTransformer transformer, Injection␣
↪injection) {
        LOGGER.info("Transformer " + transformer.getId() + " has be replaced by a " +␣
↪injection.getType());
    }

    @Override
    public void hvdcLineReplaced(HvdcLine hvdcLine, Injection injection) {
        LOGGER.info("HVDC line " + hvdcLine.getId() + " has be replaced by a " +␣
↪injection.getType());
    }
};

NetworkReducer reducer = NetworkReducer.builder()
        .withNetworkPredicate(new NominalVoltageNetworkPredicate(225.0, 400.0))
        .withBoundaryLines(true)
        .withObservers(observer)
        .build();
reducer.reduce(network);
```

3.3 Working with subnetworks

3.3.1 Concept of subnetwork

It is sometimes useful to work on an aggregated network containing the data of several networks. But you may also want to remember their structure to later work on the individual networks.

This is the case, for instance, when working with multi-country CGMES archives. They are imported as a single network on which it is possible to run a load flow, but at export, one EQ file per country should be generated.

To manage this, PowSyBl has a concept of subnetwork:

- Subnetworks are also networks (i.e. they are `Network` objects).
- A network can have several subnetworks.
- There is maximum 2 levels of networks: the main network, and possibly some subnetworks directly inside the main network. Subnetworks cannot contain subnetworks.
- Each network element (substation, voltage level, ...) can be in the main network or in a subnetwork.
- Each network element has 2 methods to retrieve:
  - the main network: `Identifiable.getNetwork()`;
  - the network it is immediately in (the main network or a subnetwork): `Identifiable.getParentNetwork()`.
- When a network element is in a subnetwork `s1`, it is retrievable (for instance by using `network.getIdentifiable(id)`) from the main network and from `s1`, but not from the other subnetworks.
- When a network element is in the main network, it is not retrievable from the subnetworks.

3.3.2 Merging networks

You can merge several independent networks together using one of the following commands:

- `Network n = Network.merge(id, network1, network2, ...)`
- `Network n = Network.merge(network1, network2, ...)`

These commands create a new `Network` `n` having a subnetwork for each network given as parameter.

For each network `networki`, a new subnetwork is created in `n` with the same ID. All its content (network element, extensions, properties, ...) is transferred to the subnetwork. As a result, `networki` will be empty after the operation.

During this merging operation, unpaired `BoundaryLine`s are examined. Two unpaired boundary lines in different networks will be paired (and a `TieLine` will be created) if:

- they have the same non-null pairing key and are the only boundary lines to have this pairing key in their respective networks;
- **OR** they have the same non-null pairing key and are the only **connected** boundary lines to have this pairing key in their respective networks.

If all the `networki` have the same source format, the resulting network will also receive this format. Else, it will be set to `hybrid`.

There are some restrictions preventing some networks to be merged together. For the merge to work:

- All the networks should be independent: they can't be subnetworks, and they can't contain subnetworks.
- The networks should have only one variant.
- There cannot be duplicates among the network element's ids of the networks.

3.3.3 Importing a network

The IIDM format supports subnetworks (starting from version 1.11). So if you import an XIIDM (XML), a JIIDM (JSON) or a BIIDM (binary) file, it could contain subnetworks. In this case, your resulting network will also have subnetworks.

You could also get a network containing subnetworks if you import a multi-country CGMES archive or directory. The content for each country will be stored as a subnetwork.

3.3.4 Creating a subnetwork via the API

You can also create subnetworks manually via the API:

```
Network network = Network.create("Root", "format0");
Network subnetwork1 = network.createSubnetwork("Sub1", "subnetwork #1", "format1");
Network subnetwork2 = network.createSubnetwork("Sub2", "subnetwork #2", "format2");
```

Once created, it is now possible to add substations, voltage levels, lines, ... in the main network ("network"), or in one of the subnetworks ("subnetwork1" or "subnetwork2") using the same methods of the API: they are all `Network` objects.

3.3.5 Detaching a subnetwork

It is possible to "detach" a subnetwork from its main network with the following instruction:

```
Network n = subnetwork.detach();
```

Note that this operation is destructive: after it the `subnetwork`'s content could not be accessed from the main network anymore, and `subnetwork` is empty (all its content is transferred into `n`).

The boundary elements, i.e. the elements linking the subnetwork to an external voltage level (it could be lines, HVDC lines or tie lines), are split if possible. If not, the detach operation will fail.

Some checks are performed which may prevent the detach method to work:

- Only subnetworks can be detached.
- There should not be un-splittable boundary elements.
- The main network should not be multi-variant.

Before calling the `detach` method, you can manually test if these checks will fail or not. You can also detect the un-splittable boundary elements and remove them if you want:

```
if (!subnetwork.isDetachable()) {
    Set<Identifiable<?>> boundaryElements = subnetwork.getBoundaryElements();
    // You can for instance check the boundary elements,
    // remove them if they are not needed, ...
    // then detach `subnetwork`
}
```

3.3.6 Flattening a network

In some particular occasion, the presence of subnetworks in your network may be an obstacle. For instance, it is not possible to merge networks if one of them has subnetworks. To circumvent this problem, you can "flatten" your network, i.e. remove its subnetworks' structure. During this operation, all the data contained in the subnetworks (`Identifiable`s, extensions, and properties) are transferred into the main network and the subnetworks (thus emptied) are removed from it.

```
// n is a network with subnetworks
n.flatten();
// Now, n has no more subnetworks. Their content was transferred into n.
```

Networks can only have a maximum of one extension of a certain type, and it is also the case for subnetworks. But since several subnetworks of the same main network may have an extension of the same type, this may be problematic when flattening the main network: there is no extensions merging generic mechanism, so we cannot automatically keep

all the extensions' content. The same problem may occur with properties since a network can only have one property of a certain name.

To solve this problem, the following policy is applied. Subnetworks are integrated in the whole network following the same order they were merged. For each one, only the properties and the extensions which are not already present in the currently flattened network (i.e. same name for properties or same type for extensions) are transferred. If a duplicate is detected, this latter is not transferred and will remain in its original subnetwork at the end of the flattening operation. It is thus possible to retrieve potential duplicates and to handle them manually.

For instance, if:

- `n0` has 2 subnetworks `s1` and `s2` (merged in this order).
- `s1` has the property `(key = val1)`.
- `s2` has the property `(key = val2)`.

After `n0.flatten()`:

- `n0` will have the property `(key = val1)` (when "integrating" `s1`, no property of key `key` was found in `n0`).
- `s1` will have no property (it was transferred to `n0`).
- `s2` will have the property `(key = val2)` (the property was *not* transferred because the property `(key = val1)` was already in `n0`).

3.4 Load flow validation

A load flow result is considered *acceptable* if it describes a feasible steady-state of a power system given its physics and its logics. More practically, generations of practitioners have set quasi-standard ways to describe them that makes it possible to define precise rules. They are described below for the different elements of the network.

See the documentation *here* to configure correctly this module.

3.4.1 Buses

The first law of Kirchhoff must be satisfied for every bus for active and reactive power:

$$\left| \sum_{branches} P + \sum_{injections} P \right| \leq \epsilon$$
$$\left| \sum_{branches} Q + \sum_{injections} Q \right| \leq \epsilon \tag{3.1}$$

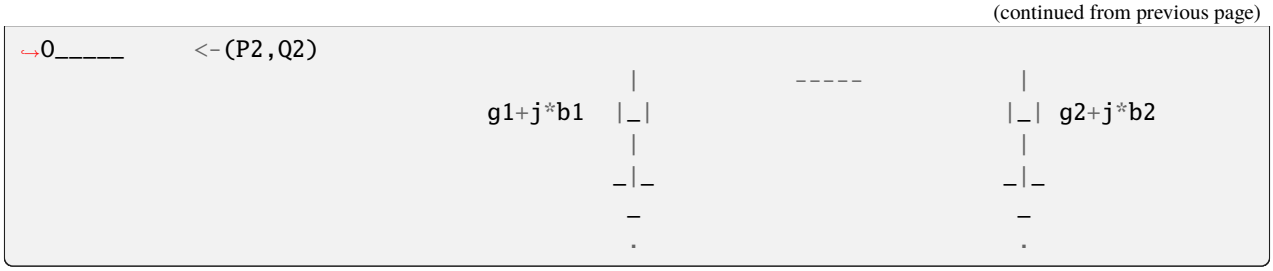
3.4.2 Branches

Lines and two-winding transformers are converted into classical PI models:

```
   V1*exp(j*theta1)     rho1*exp(j*alpha1)            r+j*x            ␣
↪rho2*exp(j*alpha2)   V2*exp(j*theta2)
       (P1,Q1)->      ____O/O_____________________-----_____________________O/
```

(continues on next page)

(continued from previous page)



- Power flow results:
  - $(\|V_1\|, \theta_1)$ and $(\|V_2\|, \theta_2)$: Magnitude (kV) and angle () of the voltage at the sides 1 and 2, respectively.
  - (P_1, Q_1) and (P_2, Q_2): Active power (MW) and reactive power (MVAr) injected in the branch on each side.
- Characteristics:
  - (ρ_1, α_1) and (ρ_2, α_2): Magnitude (no unit) and angle () of the ideal transformers ratios on each side.
  - (g_1, b_1) and (g_2, b_2): Complex shunt impedance on each side (S).
  - (r, x): Complex series impedance (Ω).

Thanks to Kirchhoff laws (see the *line* and *2-winding transformer* documentation), estimations of powers are computed according to the voltages and the characteristics of the branch:

$(P_1^{calc}, Q_1^{calc}, P_2^{calc}, Q_2^{calc}) = f(\text{Voltages}, \text{Characteristics})$

3.4.3 Three-winding transformers

To be implemented, based on a conversion into 3 two-winding transformers.

3.4.4 Generators

Active power

There may be an imbalance between the sum of generator active power setpoints targetP on one side and consumption and losses on the other side, after the load flow optimization process. Note that if it is possible to modify the setpoints during the computation (for example, if the results were computed by an Optimal Power Flow and not a Power Flow), there should be no imbalance left.

In case of an imbalance between the sum of generator active power setpoints targetP on one side and consumption and losses on the other side, the generation P of some units has to be adjusted. The adjustment is done by modifying the generation of the generators connected to the slack node of the network. It may also be done by modifying the loads connected to the slack node. The slack node is a computation point designated to be the place where adjustments are done.

This way of performing the adjustment is the simplest solution from a mathematical point of view, but it presents several drawbacks. In particular, it may not be enough in case of a large imbalance. This is why other schemes have been developed, called "distributed slack nodes".

Generators or loads are usually adjusted proportionally to a shift function to be defined. Three keys have been retained for the validation (g is a generator): Usual ways of defining this function, for each equipment that may be involved in the compensation (generator or load), read:

- proportional to P_{max}: $F = f \times P_{max}$
- proportional to $targetP$: $F = f \times targetP$
- proportional to P_{diff}: $F = f(P_{max} - targetP)$

f is a participation factor, per unit. For example, a usual definition is: $f \in \{0, 1\}$: either the unit participates or not. The adjustment is then done by doing: $P < -P \times \hat{K} \times F$ where $\hat{K}$ is a proportionality factor, usually defined for each unit by $\frac{P_{max}}{\sum F}$, $\frac{targetP}{\sum F}$ or $\frac{P_{diff}}{\sum F}$ depending on the adjustment mode (the sums run over all the units participating in the compensation).

Voltage and reactive power

If the voltage regulation is deactivated, it is expected that:

$|targetQ - Q| < \epsilon$

If the voltage regulation is activated, the generator is modeled as a PV node. The voltage target should be reached, except if reactive bounds are hit. Then, the generator is switched to PQ node and the reactive power should be equal to a limit. Mathematically speaking, one of the following 3 conditions should be met:

$$\begin{aligned} |V - targetV| &\leq \epsilon \quad \& \quad minQ \leq Q \leq maxQ \\ V - targetV &< -\epsilon \quad \& \quad |Q - maxQ| \leq \epsilon \\ targetV - V &< \epsilon \quad \& \quad |Q - minQ| \leq \epsilon \end{aligned}$$

3.4.5 Loads

To be implemented, with tests similar to generators with voltage regulation.

3.4.6 Shunts

A shunt is expected not to generate or absorb active power:

$|P| < \epsilon$

A shunt is expected to generate reactive power according to the number of activated sections and to the susceptance per section B: $\left|Q + \text{sections} * BV^2\right| < \epsilon$

3.4.7 Static VAR Compensators

Static VAR Compensators behave like generators producing zero active power except that their reactive bounds are expressed in susceptance, so that they are voltage dependent.

$targetP = 0$ MW

- If the regulation mode is `OFF`, then $targetQ$ is constant
- If the regulation mode is `REACTIVE_POWER`, it behaves like a generator without voltage regulation
- If the regulation mode is `VOLTAGE`, it behaves like a generator with voltage regulation with the following bounds (dependent on the voltage, which is not the case for generators): $minQ = -Bmax * V^2$ and $maxQ = -BminV^2$

3.4.8 HVDC lines

To be done.

3.4.9 VSC

VSC converter stations behave like generators with the additional constraints that the sum of active power on converter stations paired by a cable is equal to the losses on the converter stations plus the losses on the cable.

3.4.10 LCC

To be done.

3.4.11 Transformers with a ratio tap changer

Transformers with a ratio tap changer have a tap with a finite discrete number of positions that allows to change their transformer ratio. Let's assume that the logic is based on deadband: if the deviation between the measurement and the setpoint is higher than the deadband width, the tap position is increased or decreased by one unit.

As a result, a state is a steady state only if the regulated value is within the deadband or if the tap position is at minimum or maximum: this corresponds to a valid load flow result for the ratio tap changers tap positions.

3.5 Network modifications

The `powsybl-iidm-modification` module gathers classes and methods used to modify the network easily. Each modification must first be created with the right attributes or parameters and then applied on the network. A `NetworkModification` offers a method to check whether or not its application would have an impact on the given network.

3.5.1 Scaling

TODO

3.5.2 Topology modifications

Powsybl provides classes that can be used to easily modify the topology of the network. This includes: the creation of network elements with automatic creation of switches with respect to the topology of the voltage level, the removal of network elements and their switches, the creation of T-pieces when connecting a line to another line, and the connection of a voltage level to a line. All these classes rely on a builder to create the modification and then apply it on the network.

Naming strategy

The naming strategy aims at clarifying and facilitating the naming of the different network elements created via the different `com.powsybl.iidm.modification.NetworkModification` classes. Based on the name of the network element the user wishes to create (a VoltageLevel, a BranchFeederBay, etc.), all the other elements created during the NetworkModification will be given a name using this name as baseline and prefixes/suffixes according to the naming strategy chosen by the user. The naming strategy can be either the default one `com.powsybl.iidm.modification.topology.DefaultNamingStrategy` or a new implementation of the `NamingStrategy` interface.

Default naming strategy

Default naming strategy is used if no other naming strategy is specified. The `DefaultNamingStrategy` implements a simple naming convention following the pattern: base name + separator + element type + optional index. The default implementation uses underscores as separators and appends element types and indices when necessary to ensure unique naming.

Custom strategies

Other Naming strategies can be implemented based on the `NamingStrategy` interface. This allows for organization-specific naming conventions, different separator characters, or specialized formatting rules.

Naming strategies service loader

The `NamingStrategiesServiceLoader` enables dynamic discovery of available naming strategies through Java's ServiceLoader mechanism.

Network element creation

Create feeder bay

This class should be used to create any type of `Injection`. `Injections` are network elements with one terminal, such as loads, generators… It takes as input:

- The `InjectionAdder`, already created with the right attributes. These attributes depend on the type of `Injection`.
- The ID of the bus or busbar section (in `BUS_BREAKER` or `NODE_BREAKER` voltage levels respectively) to which the injection should be connected.
- The position order of the injection: when adding an injection to a `NODE_BREAKER` voltage level, this integer will be used to create the *`ConnectablePosition` extension* that is used for visualization. It is optional for `BUS_BREAKER` voltage levels and will be ignored if specified.
- Optionally, a name for the *`ConnectablePosition` extension*. By default, the ID of the injection will be used.
- Optionally, the direction of the injection. It is also used to fill the *`ConnectablePosition` extension*. It indicates if the injection should be displayed at the top or at the bottom of the busbar section. By default, it is `BOTTOM`.
- Optionally, a boolean `logOrThrowIfIncorrectPositionOrder`, that indicates what should happen if the position order is incorrect. This is mainly useful for voltage levels with `NODE_BREAKER` topology, since the `ConnectablePosition` extension is not created otherwise. The order position may be incorrect if
  - it has already been taken on the busbar section,
  - if it is higher or lower than the maximum or minimum available order positions for the busbar section,
  - or if the order positions of other adjacent busbar sections do not allow any possible order positions. If the boolean is set to false, then the order position will be ignored and the `ConnectablePostion` extension will not be created, but the `Injection` will be created. If the boolean is set to true, the `Injection` will not be created, and either an exception will be thrown or a log will be returned, depending on the `throwException` boolean given when applying the modification.

When applying this modification on the network, the injection is added to the voltage level associated with the bus or busbar section. If the voltage level topology kind is `BUS_BREAKER`, then the injection is added to the voltage level and connected to the bus without any extension or switches. If the voltage level topology kind is `NODE_BREAKER`, then the injection is added to the voltage level and connected to the busbar section with a closed disconnector and a breaker. Additionally, open disconnectors will be created on every parallel busbar section. To know which busbar sections are parallel, the *`BusbarSectionPosition` extension* is used. The *`ConnectablePosition` extension* will also be created for the injection with the given data, unless there are no extensions yet in the voltage level.

Create Branch Feeder bays

This class allows the creation of lines and two-winding transformers. It takes as input:

- The `BranchAdder`, which should be created beforehand with the electrotechnical characteristics of the branch.
- The ID of the bus or busbar section (in `BUS_BREAKER` or `NODE_BREAKER` voltage levels respectively) to which the side 1 of the branch should be connected.
- The ID of the bus or busbar section (in `BUS_BREAKER` or `NODE_BREAKER` voltage levels respectively) to which the side 2 of the branch should be connected.

- The position order of the branch on side 1. If the voltage level on side 1 of the branch is NODE_BREAKER, then this integer is used to create the *ConnectablePosition extension* for the branch that is used for visualization and for positioning connectables relative to each other. It is optional for BUS_BREAKER voltage levels and will be ignored if specified.
- The position order of the branch on side 2. It is the same but on the other side.
- Optionally, a name for the feeder that will be added in the *ConnectablePosition extension* for side 1. This name is used for visualization. By default, it is the ID of the connectable.
- Optionally, a name for the feeder for side 2.
- Optionally, the direction of the feeder on side 1. This information will be used to fill the field in the *ConnectablePosition extension* and indicates the relative position of the branch with its busbar section on side 1. The default value is TOP.
- Optionally, the direction on side 2.
- Optionally, a boolean logOrThrowIfIncorrectPositionOrder1, that indicates what should happen if the position order is incorrect on side 1 of the branch. This is mainly useful for voltage levels with NODE_BREAKER topology, since the ConnectablePosition extension is not created otherwise. The order position may be incorrect if
  - it has already been taken on the busbar section,
  - if it is higher or lower than the maximum or minimum available order positions for the busbar section,
  - or if the order positions of other adjacent busbar sections do not allow any possible order positions. If the boolean is set to false, then the order position will be ignored and the ConnectablePostion extension will not be created, but the Injection will be created. If the boolean is set to true, the Injection will not be created, and either an exception will be thrown or a log will be returned, depending on the throwException boolean given when applying the modification.
- Optionally, a boolean logOrThrowIfIncorrectPositionOrder2, which is the same but for the side 2 of the branch.

When the modification is applied on the network, the branch is added to both voltage levels and connected on the bus or busbar section specified for both sides. For each side, if the voltage level topology kind is BUS_BREAKER, then the branch is added to the voltage level and connected to the bus without any extension or switches. If the voltage level topology kind is NODE_BREAKER, then the branch is added to the voltage level and connected to the busbar section with a closed disconnector and a breaker. Additionally, open disconnectors will be created on every parallel busbar section. To know which busbar sections are parallel, the *BusbarSectionPosition extension* is used. The *ConnectablePosition extension* will also be created for the branch with the given data, unless no extensions are already available in the voltage level.

Create Coupling Device

This class allows the creation of coupling devices within a voltage level to couple some busbar sections. It takes as input:

- The ID of one bus or busbar section (in BUS_BREAKER or NODE_BREAKER voltage levels respectively)
- The ID of another bus or busbar section
- Optionally, a prefix to be used when creating the switches of the coupling device.

Both buses or busbar sections must be within the same voltage level. If the voltage level has a BUS_BREAKER topology, then a new breaker is created between both buses.

If the voltage level has a NODE_BREAKER topology, then the coupling device is created between the two given buses or busbar sections as such: A closed disconnector will be created on both busbar sections. A closed breaker will be created between the two closed disconnectors. An open disconnector will be created on every parallel busbar section.

To find the parallel busbar sections, the *BusbarSectionPosition extension* is used. The coupling device can be created between busbar sections that are parallel or not. If the two busbar sections are parallel and there are exactly two parallel busbar sections, then no open disconnectors are created.

Create Voltage Level Topology

This class allows the creation of the topology inside a voltage level if it is meant to be symmetrical. The voltage level must already exist and does not have to be empty. When applied to a network, it will create buses or busbar sections in a matrix of aligned buses or busbar sections. In `BUS_BREAKER` topology, the buses will be separated by `Breakers` and in `NODE_BREAKER`, the switch type between each section must be specified. It takes as input:

- The ID of the voltage level
- The count of aligned buses or busbar sections. This integer indicates the "row" number of the matrix of buses or busbar sections.
- The section count. This integer indicates the "column" number of the matrix of buses or busbar sections.
- A list of switch kinds, for `NODE_BREAKER` voltage levels. This list indicates the switches that should be created between each busbar section. In the end, `alignedBusesOrBusbarCount` * `sectionCount` buses or busbar sections will be created, and they will be connected by section either by `Breakers` in `BUS_BREAKER` topology or by the switch specified by the list in `NODE_BREAKER` topology. The length of this list must be equal to the section count - 1.

Additional input can be provided:

- The low-bus or busbar section index. This integer indicates the index of the first "row" of buses or busbar sections that should be created. If the voltage level is not empty, then the buses or busbar sections will be created starting from this index, so it can be below some already existing buses or busbar sections. By default, it is 1 (no bus or busbar section already in the voltage level).
- The low-section index. This integer indicates the index of the first section of buses or busbar sections that should be created. If the voltage level is not empty, it is possible to create buses or busbar sections next to already existing ones. By default, it is 1 (no bus or busbar section already in the voltage level).
- The bus or busbar section prefix ID is optional and used, if specified, as a prefix for the IDs of the created buses or busbar sections. This prefix is followed by the "row" index and the section number. If it is not specified, then the name of the voltage level is used as prefix.
- The switch prefix ID is also optional.
- The boolean connectExistingConnectables indicates whether existing connectables should be connected to the new topology if the busbar sections are created in a non-empty voltage level. If true, they will all be connected with an open switch of the same kind as the first switch that connects the connectable to the other busbar sections. If this boolean is true, the *ConnectFeedersToBusbarSections* modification will be called on the network.

Create Voltage Level Sections

This class allows the creation of new busbar sections inside a voltage level in the NODE_BREAKER topology. The voltage level must already have been created, must already contains some busbar sections, and these busbar sections must have the extension BusbarSectionPosition, which indicate their position in the voltage level busbar sections matrix (busbarIndex and sectionIndex). When applied to a network, it will create new busbar sections before or after a reference busbar section.

It takes as input:

- The ID of the reference busbar section
- A boolean indicating if the new busbar section(s) must be created before(left) or after(right) the reference busbar section

- A boolean indicating if a new busbar section must be created on all busbars, or only on the busbar of the reference busbar section
- The switch kind of the new switch(es) that will be created left to the newly created busbar section :
  - DISCONNECTOR means that only a DISCONNECTOR switch will be created
  - BREAKER means that a BREAKER switch surrounded by two DISCONNECTOR switches will be created
- The switch kind of the new switch(es) that will be created right to the newly created busbar section :
  - DISCONNECTOR means that only a DISCONNECTOR switch will be created
  - BREAKER means that a BREAKER switch surrounded by two DISCONNECTOR switches will be created
- A boolean indicating if the new switches created left to the newly created busbar section(s) are fictitious
- A boolean indicating if the new switches created right to the newly created busbar section(s) are fictitious
- A boolean indicating if the new switches created left to the newly created busbar section(s) will be open
- A boolean indicating if the new switches created right to the newly created busbar section(s) will be open
- The switch prefix ID, used as a prefix for the IDs of the newly created switches.
- The busbar section prefix ID, used as a prefix for the IDs of the newly created busbar sections. This prefix is followed by the busbar index and the section index, if the default naming strategy is used.

TODO: add single line diagrams

Connect feeders to busbar sections

This class allows the connection of feeders to busbar sections in `NODE_BREAKER` topology. The *`ConnectablePosition` extension* must be available for each busbar section in the voltage level.

It takes as input:

- A list of connectables that should be connected. None of them should be a `BusbarSection`.
- A list of busbar sections. The connectables will be connected to these busbar sections if they are not already.
- A boolean `connectCouplingDevices` indicating if the coupling devices of the voltage level should be connected to the busbar sections. If the busbar sections are not already connected on each side of the coupling device breaker, an open switch will be created.
- A string `couplingDeviceSwitchPrefixId` that will be used, if the boolean `connectCouplingDevices` is true, in the naming strategy to determine the IDs of the new disconnectors.

When applied to a network, the network modification will loop through all the parallel busbar sections of each input busbar section to gather the switches that are connecting the feeders. Then, the feeders and coupling devices will be connected by a switch of the same kind as the first switch that connects the feeder to the other busbar sections. If all the feeders are already connected to the busbar sections, then the network modification will do nothing.

Let's take an example. This is the voltage level before applying the modification:

If we apply the modification with both busbar sections, all the connectables and coupling devices, we will get:

If we want to only connect the two-winding transformers on busbar section bbs3, we can specify the right lists as input and we will get:

Network element removal

The classes `com.powsybl.iidm.modification.RemoveFeederBay`, `com.powsybl.iidm.modification.RemoveHvdcLine`, `com.powsybl.iidm.modification.RemoveVoltageLevel` and `com.powsybl.iidm.modification.RemoveSubstation` allow to remove all types of elements from a network.

RemoveFeederBay

This is the class to use to remove any Injection, Branch or Three-winding transformer. The builder should be used to create any instance of this class. Only the ID of the connectable to remove should be given as input. When applied to the network, the connectable will be removed, as well as all the switches connecting it to busbar sections. Note: Busbar sections are not allowed to be removed with this class.

RemoveHvdcLine

This class should be used to remove a HVDC line. The input arguments are:

- The ID of the HVDC line
- If the HVDC line is an LCC, an optional list of IDs of the shunt compensators associated with this HVDC line that should also be removed. When applied to the network, the HVDC line is removed, as well as the two converter stations on each side and the switches connecting them to their voltage levels. If the list of shunt compensators is not empty, then they will also be removed along with their switches.

RemoveVoltageLevel

This class is used to remove an entire voltage level. All the connectables, busbar sections, coupling devices of the voltage level are removed. The lines, two-winding transformers and three-winding transformers are also removed as well as their switches in other voltage levels. The builder to be used to initialize this class takes only the ID of the voltage level to be removed.

RemoveSubstation

This class should be used to remove an entire substation. All the voltage levels of the substation with all their connectables are removed. The branches and three-winding transformers are also removed with their switches in the other substations. The builder takes the ID of the substation as input.

Moving a network element

MoveFeederBay

This class is used to move feeder bays of connectables (except `BusOrBusBarSection` connectables) from one place to another within a network.

This class allows to move a feeder bay from one busbar section to another within the network. The builder should be used to create any instance of this class. It takes as input:

- The ID of the connectable whose feeder bay will be moved (`connectableId`). Note that `BusOrBusBarSection` connectables are not accepted.
- The ID of the target bus or busbar section (`targetBusOrBusBarSectionId`) to which the feeder bay should be connected.
- The ID of the target voltage level (`targetVoltageLevelId`) where the feeder bay will be moved to.
- The terminal object that specifies which terminal of the connectable should be moved.

When the modification is applied on the network, the system identifies and updates all relevant switches and connections to move the feeder bay from its current position to the specified target place. This includes disconnecting from

the original busbar section and reconnecting to the target busbar section. If the target voltage level topology kind is `BUS_BREAKER`, the connectable is connected to the target bus without additional switches. If the target voltage level topology kind is `NODE_BREAKER`, the appropriate disconnectors and breakers are created to connect the feeder bay to the target busbar section, maintaining the correct topology. This modification ensures that the connectivity of the network is preserved while moving the feeder bay to its new position.

Connect a line on a line or a voltage level on a line

ConnectVoltageLevelOnLine

TODO

RevertConnectVoltageLevelOnLine

TODO

CreateLineOnLine

TODO

RevertCreateLineOnLine

TODO

ReplaceTeePointbyVoltageLevelOnLine

TODO

3.5.3 Tripping

Battery tripping

TODO

Branch tripping

TODO

Busbar section tripping

TODO

Bus tripping

TODO

Boundary line tripping

TODO

Generator tripping

TODO

Hvdc line tripping

TODO

Line tripping

TODO

Load tripping

TODO

Shunt compensator tripping

TODO

Static Var compensator tripping

TODO

Switch tripping

TODO

Three-winding transformer tripping

TODO

Tie line tripping

TODO

Two-winding transformer tripping

TODO

3.5.4 Other modifications

List

This modification is used to apply a list of any Powsybl `NetworkModification`.

Class: `NetworkModificationList`

Area interchange

This modification is used to update the target of an area interchange.

The target is in MW in load sign convention (negative for export, positive for import). Providing `Double.NaN` removes the target.

Class: `AreaInterchangeTargetModification`

Battery

This modification is used to update the target powers (active `targetP` and reactive `targetV`) of a battery.

Class: `BatteryModification`

Connection

This modification is used to connect a network element to the closest bus or bus bar section.

It works on:

- `Connectable` elements by connecting their terminals
- HVDC lines, by connecting the terminals of their converter stations
- Tie lines, by connecting the terminals of their underlying boundary lines

It is possible to specify a side of the element to connect. If no side is specified, the network modification will try to connect every side.

Class: `ConnectableConnection`

Boundary line

This modification is used to update the active and reactive powers of the load part of a boundary line.

If `relativeValue` is set to true, then the new constant active power (`P0`) and reactive power (`Q0`) are set as the addition of the given values to the previous ones. If `relativeValue` is set to false, then the new constant active power (`P0`) and reactive power (`Q0`) are updated to the new given values.

Class: `BoundaryLineModification`

Disconnections

Planned

This modification is used to disconnect a network element from the bus or bus bar section to which it is currently connected. It should be used if the disconnection is planned. If it is not, `UnplannedDisconnection` should be used instead.

It works on:

- `Connectable` elements.
- HVDC lines, by disconnecting their converter stations
- Tie lines, by disconnecting their underlying boundary lines

It is possible to specify a side of the element to connect. If no side is specified, the network modification will try to connect every side.

Class: `PlannedDisconnection`

Unplanned

This modification is used to disconnect a network element from the bus or bus bar section to which it is currently connected. It should be used if the disconnection is unplanned. If it is not, `PlannedDisconnection` should be used instead.

It works on:

- `Connectable` elements.
- HVDC lines, by disconnecting their converter stations
- Tie lines, by disconnecting their underlying boundary lines

It is possible to specify a side of the element to connect. If no side is specified, the network modification will try to connect every side.

Class: `UnplannedDisconnection`

Generator

Modification

This modification is used to apply a set of modifications on a generator.

The data to be updated are optional among:

- `minP`, the minimum active power boundary in MW.
- `maxP`, the maximum active power boundary in MW.
- `targetV`, the target voltage value in kV.
- `targetQ`, the target reactive power value in MVAR.
- `connected`, the connection state of the generator terminal.
- `voltageRegulatorOn`, to activate or deactivate the generator voltage regulator status. If `true` and the generator target voltage is not set then an acceptable value for the generator `targetV` is computed before activating.
- The active power if `targetP` or `deltaTargetP` are given. An active power is determined by the new `targetP` if given, and if not then the `deltaTargetP` is considered instead and the new value of the generator `targetP` is the addition of the old generator value with the given delta target P value. Then, according to the given `ignoreCorrectiveOperations` parameter:
  - If `ignoreCorrectiveOperations` is true, this determined active power is applied as the new generator target P value.
  - If `ignoreCorrectiveOperations` is false, then the new active power will also depend on the limits and will be the minimum value between the generator `maxP` and the maximum value between the generator `minP` and the previously determined active power value. Besides, if the generator connection state has not been updated before within this `NetworkModification` then the generator is connected if necessary.

Class: `GeneratorModification`

Connection

This modification is used to connect a given generator.

If the generator terminal is regulating then it will also set its target voltage if an acceptable value is found.

Class: `ConnectGenerator`

Set to local regulation

This modification is used to set the generator regulating terminal to a local regulation.

The target voltage value is set to the same value for all the generators of the bus that are regulating locally. In case other generators are already regulating locally on the same bus, targetV value is determined by being the closest value to the voltage level nominal voltage among the regulating terminals. If no other generator is regulating on the same bus, targetV engineering unit value is adapted to the voltage level nominal voltage, but the per unit value remains the same.

Class:`SetGeneratorToLocalRegulation`

HVDC line

This modification is used to modify a given HVDC line (and potentially its angle droop active power control extension).

- Modify the HVDC line `activePowerSetpoint` if given, relatively to the existent `activePowerSetpoint` if `relativeValue` is true or as a replacement value if not.
- Modify the `convertersMode` with the given one if set
- Modify the angle droop active power control extension (if existing but will not crash if not found for the HVDC line):
  - Enable or disable the AC emulation if `acEmulationEnabled` is provided
  - Update the active power if `p0` is provided
  - Update the droop in MW/degree if `droop` is provided

Class: `HvdcLineModification`

Load

Modification

This modification updates the `P` and `Q` values of the load.

If `relativeValue` is set to true, then the new constant active power (`P0`) and reactive power (`Q0`) are set as the addition of the given values to the previous ones. If `relativeValue` is set to false, then the new constant active power (`P0`) and reactive power (`Q0`) are updated to the new given values.

Class: `LoadModification`

Percent modification

This modification is used to add or remove a percentage of the P and Q of the load. The percentage to add or remove for P and Q cannot be less than -100 (in percentage).

Class: `PercentChangeLoadModification`

Phase shifters

Optimize tap modification

This modification is used to find the optimal phase tap changer position of a given two-winding transformer phase shifter id.

A phase shifter optimization load flow is run with the configured `load-flow-based-phase-shifter-optimizer` to determine the optimal tap position.

Class: `PhaseShifterOptimizeTap`

Fixed tap modification

This modification updates the phase tap changer of a given two-winding transformer phase shifter id.

It updates its `tapPosition` with the given value and set the phase tap changer as not regulating.

Class: `PhaseShifterSetAsFixedTap`

Shift tap modification

This modification is used to update the phase tap changer of a given two-winding transformer phase shifter id.

It sets the phase tap changer as not regulating and updates its `tapPosition` by adjusting it with the given `tapDelta` applied on the current tap position. The resulting tap position is bounded by the phase tap changer lowest and highest possible positions.

Class: `PhaseShifterShiftTap`

Replace tie lines by lines

This modification is used to replace all the tie lines of a network to simple lines built from the original tie line and its 2 boundary lines.

- The two voltage levels are set from the tie line boundary lines terminal voltage levels (the first voltage level from the first boundary line and the second from the second one).
- For each voltage level the topology kind is taken into account to create node (for `NODE_BREAKER` kind) or bus and connectable bus (for `BUS_BREAKER` kind)
- The tie line id, name, r, x, b1, b2, g1, g2 are set in the new line
- Active power limits, apparent power limits and current limits are set on each side of the line from the limits of the 2 boundary lines
- Terminal active and reactive powers are set for both terminals from each boundary line active and reactive powers
- Line properties are set from the merge of the tie line and its 2 boundary lines properties
- Line aliases are set from the merge of the tie line and its 2 boundary lines aliases
- If the tie line has a pairing key then it is added to the new line as a pairing key alias
- The tie line and its boundary lines are removed from the network

Class: `ReplaceTieLinesByLines`

Shunt compensator

This modification is used to (dis)connect a shunt compensator and/or change its section count in service.

If the modification connects the shunt compensator and its terminal is regulating then it will also set its target voltage if an acceptable value is found.

Class: `ShuntCompensatorModification`

Static var compensator

This modification modifies the voltage and reactive power setpoints of a static var compensator, following a load convention.

Class: `StaticVarCompensatorModification`

Switch

Close

This modification is used to close a switch.

Class: `CloseSwitch`

Open

This modification is used to open a switch.

Class: `OpenSwitch`

Transformers

Three-winding transformers legs rated voltage

This modification is used to modify the rated voltage of each leg of a three-winding transformer.

On each leg the new rated voltage is computed from the given common rated voltage multiplied by the ratio (leg old rated voltage / rated voltage of the three-winding transformer (the `ratedU0` also used as nominal voltage) at the fictitious bus (in kV)).

Class: `ThreeWindingsTransformerModification`

Replace 1 three-winding transformer by 3 two-winding transformers

This modification is used to replace all or a given list of `ThreeWindingsTransformer` by triplets of `TwoWindingsTransformer`.

For each `ThreeWindingsTransformer` to be replaced:

- A new voltage level is created for the star node with nominal voltage of ratedU0.
- Three `TwoWindingsTransformers` are created, one for each leg of the `ThreeWindingsTransformer` to transform.
- The following attributes are copied from each leg to the new associated `TwoWindingsTransformer`:
  - Electrical characteristics, ratio tap changers, and phase tap changers. No adjustments are required.
  - Operational Loading Limits are copied to the non-star end of the two-winding transformers.
  - Active and reactive powers at the terminal are copied to the non-star terminal of the two-winding transformer.
- Aliases:
  - Aliases for known CGMES identifiers (terminal, transformer end, ratio, and phase tap changer) are copied to the right `TwoWindingsTransformer` after adjusting the alias type.
  - Aliases that are not mapped are recorded in the functional log.
- Properties:
  - Star bus voltage and angle are set to the bus created for the star node.
  - The names of the operational limits are copied to the right `TwoWindingsTransformer`.
  - The rest of the properties of the `ThreeWindingsTransformer` are transferred to all 3 `TwoWindingsTransformer`.
- Extensions:
  - Only IIDM extensions are copied: `TransformerFortescueData`, `PhaseAngleClock`, and `TransformerToBeEstimated`.
  - CGMES extensions can not be copied, as they cause circular dependencies.
  - Extensions that are not copied are recorded in the functional log.
- All the controllers using any of the `ThreeWindingsTransformer` terminals as regulated terminal are updated.

- New and removed equipment is recorded in the functional log.
- Internal connections are created to manage the replacement.

Class: `ReplaceThreeWindingsTransformersBy3TwoWindingsTransformers`

Replace 3 two-winding transformers by 1 three-winding transformer

This modification is used to replace all or a given list of `TwoWindingsTransformer` by `ThreeWindingsTransformer`.

In the list of `TwoWindingsTransformer` if only one of a triplet of `TwoWindingsTransformer` is given then the 3 `TwoWindingsTransformer` will be transformed to a `ThreeWindingsTransformer`.

Conditions to detect a triplet of `TwoWindingsTransformer` to transform:

- `BusbarSections` and the three `TwoWindingsTransformer` are the only connectable equipment allowed in the voltage level associated with the star bus.
- The three `TwoWindingsTransformer` must be connected to the star bus.
- The star terminals of the two-winding transformers must not be regulated terminals for any controller.
- Each `TwoWindingsTransformer` is well oriented if the star bus is located at the end 2.

Then a `ThreeWindingsTransformer` is created to replace them:

- The following attributes are copied from each `TwoWindingsTransformer` to the new associated leg:
  - Electrical characteristics, ratio tap changers, and phase tap changers. Adjustments are required if the `TwoWindingsTransformer` is not well oriented.
  - Only the operational loading limits defined at the non-star end are copied to the leg.
  - Active and reactive powers at the non-star terminal are copied to the leg terminal.
- Aliases:
  - Aliases for known CGMES identifiers (terminal, transformer end, ratio, and phase tap changer) are copied to the `ThreeWindingsTransformer` after adjusting the alias type.
  - Aliases that are not mapped are recorded in the functional log.
- Properties:
  - Voltage and angle of the star bus are added as properties of the `ThreeWindingsTransformer`.
  - Only the names of the transferred operational limits are copied as properties of the `ThreeWindingsTransformer`.
  - All the properties of the first `TwoWindingsTransformer` are transferred to the `ThreeWindingsTransformer`, then those of the second that are not in the first, and finally, the properties of the third that are not in the first two.
  - Properties that are not mapped are recorded in the functional log.
- Extensions:
  - Only IIDM extensions are copied: `TransformerFortescueData`, `PhaseAngleClock`, and `TransformerToBeEstimated`.
  - CGMES extensions can not be copied, as they cause circular dependencies.
  - Extensions that are not copied are recorded in the functional log.
- All the controllers using any of the `TwoWindingsTransformer` terminals as regulated terminal are updated.

- New and removed equipment is recorded in the functional log.
- Internal connections are created to manage the replacement.

Class: `Replace3TwoWindingsTransformersByThreeWindingsTransformers`

Tap changers

Phase tap changer position

This modification is used to modify a phase tap changers tap position of a given `PhaseTapChangerHolder` (for two or three-winding transformer).

The new tap position can be either the one given in parameter or a relative position added to the existing one. The `PhaseTapChangerHolder` can be from:

- A two-winding transformers
- A three-winding transformer with a single phase tap changer
- A leg of a three-winding transformer

Class: `PhaseTapPositionModification`

Ratio tap changer position

This modification is used to modify a ratio tap changers tap position of a given `RatioTapChangerHolder` (for two or three-winding transformer).

The `RatioTapChangerHolder` can be from:

- A two-winding transformers
- A three-winding transformer with a single phase tap changer
- A leg of a three-winding transformer

Class: `RatioTapPositionModification`

VSC converter station

This modification is used to modify the voltage and reactive power setpoints of a VSC converter station, following a generator convention.

Class: `VscConverterStationModification`

CHAPTER

FOUR

SIMULATION

One major aim of the PowSyBl project is to make it easy to plug it with different simulators or optimization tools for grid calculations. At the moment, several types of calculations are supported: load flow, security analysis, sensitivity analysis, short-circuit analysis, or time-domain simulation (also called dynamic simulation).

For each of them, one or several simulators are supported, but it is also possible to plug your own simulator within the library. Below comes a description of the various types of simulation.

4.1 Load flow

4.1.1 Configuration

load-flow-based-phase-shifter-optimizer

The `load-flow-based-phase-shifter-optimizer` module is used by the `com.powsybl.action.util.LoadFlowBasedPhaseShifterOptimizer` class, which is an implementation of the `com.powsybl.action.util.PhaseShifterOptimizer` interface. The `LoadFlowBasedPhaseShifterOptimizer` tries to solve a current violation on a phase tap changer.

Required properties

load-flow-name The `load-flow-name` property is an optional property that defines the implementation name to use for running the load flow. If this property is not set, the default load flow implementation is used. See *Loadflow Configuration* to configure the default load flow.

Examples

YAML configuration:

```
load-flow-based-phase-shifter-optimizer:
    load-flow-name: Mock
```

XML configuration:

```
<load-flow-based-phase-shifter-optimizer>
    <load-flow-name>Mock</load-flow-name>
</load-flow-based-phase-shifter-optimizer>
```

Implementation

The `load-flow` module is used to configure the load flow default implementation name. Each load flow implementation provides a subclass of `com.powsybl.loadflow.LoadFlowProvider` correctly configured to be found by `java.util.ServiceLoader`. A load flow provider exposes a name that can be used in the LoadFlow Java API to find a specific load flow implementation. It can also be used to specify a default implementation in this platform config module. If only one `com.powsybl.loadflow.LoadFlowProvider` is present in the classpath, there is no need to specify a default LoadFlow implementation name. In the case where more than one `com.powsybl.loadflow.LoadFlowProvider` is present in the classpath, specifying the default implementation name allows LoadFlow API user to use LoadFlow.run(...) and LoadFlow.runAsync(...) methods to run a load flow. Using these methods when no default load flow name is configured and multiple implementations are in the classpath will throw an exception. An exception is also thrown if no implementation at all is present in the classpath, or if specifying a load flow name that is not present on the classpath.

If you have several implementations in your classpath, you need to choose which implementation to use in your configuration file:

```
load-flow:
  default-impl-name: "<IMPLEMENTATION_NAME>"
```

XML configuration:

```
<load-flow>
    <default-impl-name>Mock</default-impl-name>
</load-flow>
```

Each implementation is identified by its name, that should be unique in the classpath:

- use "OpenLoadFlow" to use PowSyBl OpenLoadFlow
- use "DynaFlow" to use DynaFlow implementation

default-parameters-loader

To define a set of load flow parameters (generic or specific) that should be set for your application before any configuration file is read, you can use an implementation of `LoadFlowDefaultParametersLoader`. It uses a JSON parameters file in your java classpath to override default values with whenever a new `LoadFlowParameters` object is created.

If multiple `LoadFlowDefaultParametersLoader` classes are present in your classpath, you should specify which one you want to use using the `default-parameters-loader` parameter of module `load-flow`:

```
load-flow:
  default-parameters-loader: "MyDefaultParameters"
```

Parameters

Optional properties

You may configure some generic parameters for all load flow implementations:

```
load-flow-default-parameters:
    dc: false
    voltageInitMode: UNIFORM_VALUES
    distributedSlack: true
    balanceType: PROPORTIONAL_TO_GENERATION_P_MAX
    countriesToBalance:
```

(continues on next page)

(continued from previous page)

```
    - FR
    - BE
  readSlackBus: false
  writeSlackBus: false
  useReactiveLimits: true
  phaseShifterRegulationOn: false
  transformerVoltageControlOn: false
  shuntCompensatorVoltageControlOn: false
  componentMode: MAIN_CONNECTED
  twtSplitShuntAdmittance: false
  dcUseTransformerRatio: true
  dcPowerFactor: 1.0
  hvdcAcEmulation: true
```

The parameters may also be overridden with a JSON file, in which case the configuration will look like:

```
{
  "version": "1.8",
  "dc": false,
  "voltageInitMode": "UNIFORM_VALUES",
  "distributedSlack": true,
  "balanceType": "PROPORTIONAL_TO_GENERATION_P_MAX",
  "countriesToBalance": ["FR", "BE"],
  "readSlackBus": false,
  "writeSlackBus": false,
  "useReactiveLimits": true,
  "phaseShifterRegulationOn": false,
  "transformerVoltageControlOn": false,
  "shuntCompensatorVoltageControlOn": false,
  "componentMode": "MAIN_CONNECTED",
  "twtSplitShuntAdmittance": false,
  "dcUseTransformerRatio": true,
  "dcPowerFactor": 1.0,
  "hvdcAcEmulation": true
}
```

dc The `dc` property is an optional property that defines if you want to run an AC power flow (`false`) or a DC power flow (`true`). The default value is `false`.

voltageInitMode The `voltageInitMode` property is an optional property that defines the policy used by the load flow to initialize the voltage values. The available values are:

- `UNIFORM_VALUES`: $v = 1pu$, $\theta = 0$
- `PREVIOUS_VALUES`: use previous computed value from the network
- `DC_VALUES`: $v = 1pu$, θ initialized using a DC load flow

The default value is `UNIFORM_VALUES`.

distributedSlack The `distributedSlack` property is an optional property that defines if the active power mismatch is distributed over the network or not. The default value is `true`.

balanceType The `balanceType` property is an optional property that defines, if `distributedSlack` parameter is set to true, how to manage the distribution. Several algorithms are supported. All algorithms follow the same scheme: only some elements are participating in the slack distribution, with a given participation factor. Six options are available:

- If using `PROPORTIONAL_TO_GENERATION_P_MAX` then the participating elements are the generators. The participation factor is computed using the maximum active power target $MaxP$ and the active power control droop. The default droop value is 4. If present, the simulator uses the droop of the generator given by the *active power control extension*.
- If using `PROPORTIONAL_TO_GENERATION_P` then the participating elements are the generators. The participation factor is computed using the active power set point $TargetP$.
- If using `PROPORTIONAL_TO_GENERATION_REMAINING_MARGIN` then the participating elements are the generators. The participation factor is computed using the difference between the active power limit and active power set point $TargetP$. The limit used is $MaxP$ if production needs to increase and $MinP$ if it needs to decrease.
- If using `PROPORTIONAL_TO_GENERATION_PARTICIPATION_FACTOR` then the participating elements are the generators. The simulator uses the participation factors of the generators given by the *active power control extension*.
- If using `PROPORTIONAL_TO_LOAD` then the participating elements are the loads. The participation factor is computed using the active power $P0$.
- If using `PROPORTIONAL_TO_CONFORM_LOAD` then the participating elements are the loads which have a conform active power part. The participation factor is computed using the *load detail extension*, which specifies the variable and the fixed parts of $P0$. The slack is distributed only on loads that have a variable part. If the extension is not available on a load, the whole $P0$ is considered as a variable.

Some algorithms may not be supported by all LoadFlow providers or all simulation types. If you plan to use it, check the documentation of your LoadFlow provider. In the case of OpenLoadFlow there are limitations for sensitivity analysis, see OpenLoadFlow sensitivity analysis documentation.

This default value is `PROPORTIONAL_TO_GENERATION_P_MAX`.

countriesToBalance The `countriesToBalance` property is an optional property that defines the list of ISO-3166 country which participating elements are used for slack distribution. If the slack is distributed but this parameter is not set, the slack distribution is performed over all countries present in the network.

readSlackBus The `readSlackBus` is an optional property that defines if the slack bus has to be selected in the network through the *slack terminal extension*. The default value is `true`.

writeSlackBus The `writeSlackBus` is an optional property that says if the slack bus has to be written in the network using the *slack terminal extension* after a load flow computation. The default value is `true`.

useReactiveLimits
The `useReactiveLimits` property is an optional property that defines whether the load flow should take into account equipment's reactive limits. Applies to generators, batteries, static VAR compensators, boundary lines, and HVDC VSCs.
The default value is `true`.

phaseShifterRegulationOn The `phaseShifterRegulationOn` property is an optional property that defines whether phase shifter regulating controls should be simulated in the load flow. The default value is `false`.

transformerVoltageControlOn The `transformerVoltageControlOn` property is an optional property that defines whether transformer voltage regulating controls should be simulated in the load flow. The default value is `false`.

shuntCompensatorVoltageControlOn The `shuntCompensatorVoltageControlOn` property is an optional property that defines whether shunt compensator voltage regulating controls should be simulated in the load flow. The default value is `false`.

componentMode The `componentMode` property is an optional property that defines 3 possibles modes to run power flow. These modes can be :

- `ALL_CONNECTED`: the power flow is computed over all synchronous components of all connected components

- `MAIN_CONNECTED` : the power flow is computed over all synchronous components of the main (largest) connected component
- `MAIN_SYNCHRONOUS` : the power flow is computed on the main (largest) synchronous component The default value is `MAIN_CONNECTED`.
-

twtSplitShuntAdmittance The `twtSplitShuntAdmittance` property is an optional property that defines whether the shunt admittance is split at each side of the series impedance for transformers. The default value is `false`.

dcUseTransformerRatio The `dcUseTransformerRatio` property is an optional property that defines if the ratio of transformers should be used in the flow equations in a DC power flow. The default value of this parameter is `true`.

dcPowerFactor The `dcPowerFactor` property is an optional property that defines the power factor used to convert current limits into active power limits in DC calculations. The default value is `1.0`.

hvdcAcEmulation The `hvdcAcEmulation` property is an optional property that defines whether AC emulation for HVDC should be simulated in the load flow or not (HVDC that are in AC emulation mode should have the hvdc-angle-droop-active-power-control extension). The default value is `true`.

Specific parameters

Some implementations use specific parameters that can be defined in the configuration file or in the JSON parameters file:

- PowSyBl OpenLoadFlow
- DynaFlow

A load flow is a numerical analysis of the flow of electric power in an interconnected system, where that system is considered to be in normal steady-state operation. Load flow studies are important for planning future expansion of power systems as well as in determining the best operation of existing systems. The principal outputs obtained from the load flow study are the magnitude and phase angle of the voltage at each bus, and the active and reactive power flowing in each line; the current is computed from both of them. In this page we will go into some details about what are the inputs and outputs of a load flow simulation, what is expected from a load flow result, how the load flow validation feature works, what load flow implementations are compatible with the generic interface, and how to configure it for the different implementations.

4.1.2 Inputs

The only input for a load flow simulation is a network and optionally a set of parameters. The parameters could be generic (use `LoadFlowParameters`), meaning that there are shared between all implementations. They also could be specific to an implementation.

4.1.3 Outputs

The load flow simulation outputs consists of:

- A network, which has been modified based on the simulation results. The modified variables are the active and reactive power at the terminals, the voltage magnitude and voltage angle at all buses, the solved tap changers positions, the solved shunt compensator sections.
- A global status indicating if the simulation succeeded for all synchronous components (`Fully Converged` status), or for only some of them (`Partially Converged` status), or for none of them (`Failed` status).
- Detailed results per synchronous component: a convergence status, the number of iterations (could be equal to `-1` if not relevant for a specific implementation), the selected reference bus (voltage angle reference), the selected slack buses (the buses at which the power balance has been done), active power mismatch at slack buses, and amount of distributed active power (zero MW if slack distribution is disabled).

- Metrics regarding the computation. Depending on the load flow implementation, the content of these metrics may vary.
- Logs in a simulator specific format.

4.1.4 Implementations

The following power flow implementations are supported:

- PowSyBl OpenLoadFlow
- Dynaflow

4.1.5 Going further

- *Run a power flow through an iTools command*: Learn how to perform a power flow calculation from the command line
- Load flow tutorial

4.2 Dynamic simulation

The dynamic simulation aims at capturing the transient response of the system, and not only to compute the steady state solution. It may or not involve the activation of events like a line disconnection for example.

4.2.1 Configuration

The `dynamic-simulation` module is used to configure the execution of the *dynamic-simulation* command and simulation.

Implementation

If you have several implementations in your classpath, you need to choose which implementation to use in your configuration file with the `default-impl-name` property. Each implementation is identified by its name, that may be unique in the classpath:

- Use "Dynawo" to use powsybl-dynawo implementation

The `com.powsybl.dynamicsimulation.DynamicSimulationParameters` class provides the generic parameters for all `com.powsybl.dynamicsimulation.DynamicSimulation` implementations. Specific parameters should be provided as an extension of the `DynamicSimulationParameters` class.

YAML configuration:

```
dynamic-simulation:
  default-impl-name: Mock
```

XML configuration:

```
<dynamic-simulation>
  <default-impl-name>Mock</default-impl-name>
</dynamic-simulation>
```

Parameters

The `dynamic-simulation-default-parameters` module is used every time a dynamic-simulation is run. It defines the default values for the most common parameters a `com.powsybl.dynamicsimulation.DynamicSimulation` implementation should be able to handle.

You may configure some generic parameters for all implementations:

```
dynamic-simulation-default-parameters:
    startTime: 0
    stopTime: 1
    debugDir: /tmp/debugDir
```

The parameters may also be overridden with a JSON file, in which case the configuration will look like:

```
{
  "version" : "1.0",
  "startTime" : 0,
  "stopTime" : 1,
  "debugDir": "/tmp/debugDir",
  "extensions" : {
    ...
  }
}
```

Optional properties

startTime `startTime` defines when the simulation begins, in seconds. The default value of this property is `0`.

stopTime `stopTime` defines when the simulation stops, in seconds. The default value of this property is `1`.

debugDir This property specifies the directory path where debug files will be dumped. If `null`, no file will be dumped.

Specific parameters

Some implementations use specific parameters that can be defined in the configuration file or in the JSON parameters file:

- Dynawo and its default parameters.

Examples

YAML configuration:

```
dynamic-simulation-default-parameters:
  startTime: 0
  stopTime: 3600
  debugDir: /tmp/debugDir
```

XML configuration:

```
<dynamic-simulation-default-parameters>
  <startTime>0</startTime>
  <stopTime>3600</stopTime>
  <debugDir>/tmp/debugDir</debugDir>
</dynamic-simulation-default-parameters>
```

4.2.2 Inputs

The inputs of a dynamic simulation are the following:

- a static network
- a set of dynamic models provided by the simulator
- a set of parameters associated to each dynamic model, with carefully chosen values
- optionally, a description of events occurring in the dynamic simulation (disconnection of a line, change of tap for a transformer, etc.)
- a set of parameters for the simulator itself (simulation start and stop time, solver parameters, etc.)
- a configuration file to configure the output variables to export at the end of the simulation

Dynamic models configuration

The dynamic models may be provided through a groovy script thanks to the `GroovyDynamicModelsSupplier` provided in powsybl-dynamic-simulation-dsl artifact. Note that the syntax of this groovy script is specific to each simulator. See Dynawo dynamic model configuration for Dynawo specific DSL and others configuration methods.

Event models configuration

The event models may be provided through a groovy script thanks to the `GroovyEventModelsSupplier` provided in powsybl-dynamic-simulation-dsl artifact. Note that the syntax of this groovy script is specific to each simulator. See Dynawo event model configuration for Dynawo specific DSL and others configuration methods.

Output variables configuration

The output variables configuration may be provided through a groovy script thanks to the `GroovyOutputVariablesSupplier` provided in powsybl-dynamic-simulation-dsl artifact. Note that the syntax of this groovy script is specific to each simulator. See Dynawo output variables configuration for Dynawo specific DSL and others configuration methods.

4.2.3 Outputs

The outputs of a dynamic simulation are:

- the updated static network (which may have been topologically modified depending on the events or automatons defined as inputs)
- the different results of the dynamic simulation:
  - some curves or final state values asked for by the user to track the evolution of specific variables throughout the simulation
  - some aggregated data regarding constraints, like a security analysis output
  - timelines that contain the list of events that occurred during the dynamic simulation, be them planned beforehand through events, or not
  - logs about the execution of the dynamic simulator

4.2.4 Implementations

For the moment, the only available implementation is provided by `powsybl-dynawo`, which links PowSyBl with Dynao open source suite.

4.2.5 Going further

- *Run a dynamic simulation through an iTools command*: learn how to perform a dynamic simulation from the command line
- *List dynamic simulation models with an iTools command*: learn how to load a list of all dynamic simulation models from the command line

4.3 Security analysis

4.3.1 Configuration

The `security-analysis` module is used to configure the execution of the *security-analysis* command and simulation.

Implementation

preprocessor The `preprocessor` property is an optional property which requires that the `SecurityAnalysisPreprocessor` with specified name is used to preprocess inputs, based on the contingency file, before actually running the security analysis.

Such a preprocessor will have the possibility to programmatically transform the following objects before the security analysis is actually executed :

- The `Network`
- The `ContingenciesProvider`
- The `LimitViolationDetector`
- The `LimitViolationFilter`
- The `SecurityAnalysisParameters`
- The `SecurityAnalysisInterceptors`

It enables, for example, to customize what should be considered a limit violation and what should not.

If absent, the default behavior of the tool is used: the contingency file is simply interpreted by the configured contingency provider.

YAML configuration:

```
security-analysis:
    preprocessor: my_custom_preprocessor_name
```

XML configuration:

```
<security-analysis>
    <preprocessor>my_custom_preprocessor_name</preprocessor>
</security-analysis>
```

Parameters

Violations increase thresholds

The user can provide parameters to define which violations must be raised after a contingency, if the violation was already present in the pre-contingency state (`IncreasedViolationsParameters`).

flow-proportional-threshold After a contingency, only flow violations (either current, active power or apparent power violations) that have increased in proportion by more than a threshold value, compared to the pre-contingency state, are listed in the limit violations. The other ones are filtered. The threshold value is unitless and should be positive. This

method gets the flow violation proportional threshold. The default value is 0.1, meaning that only violations that have increased by more than 10% appear in the limit violations.

low-voltage-proportional-threshold After a contingency, only low-voltage violations that have increased by more than the proportional threshold compared to the pre-contingency state, are listed in the limit violations, the other ones are filtered. This method gets the low-voltage violation proportional threshold (unitless, should be positive). The default value is 0.0, meaning that only violations that have increased by more than 0.0 % appear in the limit violations (note that for low-voltage violation, it means that the voltage in the post-contingency state is lower than the voltage in the pre-contingency state).

low-voltage-absolute-threshold After a contingency, only low-voltage violations that have increased by more than an absolute threshold compared to the pre-contingency state, are listed in the limit violations, the other ones are filtered. This method gets the low-voltage violation absolute threshold (in kV, should be positive). The default value is 0.0, meaning that only violations that have increased by more than 0.0 kV appear in the limit violations (note that for low-voltage violation, it means that the voltage in the post-contingency state is lower than the voltage in the pre-contingency state).

high-voltage-proportional-threshold Same as before but for high-voltage violations.

high-voltage-absolute-threshold Same as before but for high-voltage violations.

Violations filtering

The violations listed in the results can be filtered to consider only a certain type of violations, to consider only a few voltage levels or to limit the geographical area by filtering equipment by countries. Check out the documentation of the *limit-violation-default-filter* configuration module.

Example Using the following configuration, the results will contain only voltage violations for equipment in France or Belgium:

```
limit-violation-default-filter:
    countries:
        - FR
        - BE
    violationTypes:
        - LOW_VOLTAGE
        - HIGH_VOLTAGE
```

4.3.2 Contingency DSL

The contingency DSL is a domain specific language written in groovy for the creation of a contingency list, used in *security analyses* or *sensitivity analyses*.

N-1 contingency

An N-1 contingency is a contingency that triggers a single piece of equipment at a time.

```
contingency('contingencyID') {
    equipments 'equipmentID'
}
```

where the `contingencyID` is the identifier of the contingency and the `equipmentID` is the identifier of a supported piece of equipment. If the equipment doesn't exist or is not supported, an error will occur.

N-K contingency

An N-K contingency is a contingency that triggers several equipments at a time. The syntax is the same as for the N-1 contingencies, except that you have to pass a list of equipments' IDs.

```
contingency('contingencyID') {
    equipments 'equipment1', 'equipment2'
}
```

Manual contingency list

A manual contingency list is a set of contingencies that are explicitly defined. In the following example, the list contains two contingencies that trigger respectively the equipment `equipment1` and `equipment2`:

```
contingency('contingency1') {
    equipments 'equipment1'
}

contingency('contingency2') {
    equipments 'equipment2'
}
```

Automatic contingency list

As the DSL is written in Groovy, it's possible to write a more complex script. For example, it's possible to iterate over the equipment of the network to generate the contingency list. The network is accessible using the `network` variable.

The following example creates an N-1 contingency for each line of the network. We use the ID of the lines as identifier for the contingencies.

```
for (l in network.lines) {
    contingency(l.id) {
        equipments l.id
    }
}
```

It's also possible to filter the lines, for example, to consider on the boundary lines:

```
import com.powsybl.iidm.network.Country

for (l in network.lines) {
    country1 = l.terminal1.voltageLevel.substation.country
    country2 = l.terminal2.voltageLevel.substation.country
    if (country1 != country2) {
        contingency(l.id) {
            equipments l.id
        }
    }
}
```

The following example creates a list of contingencies for all 380 kV lines:

```
for (l in network.lines) {
    nominalVoltage1 = l.terminal1.voltageLevel.nominalV
    nominalVoltage2 = l.terminal2.voltageLevel.nominalV
```

(continues on next page)

(continued from previous page)

```
    if (nominalVoltage1 == 380 || nominalVoltage2 == 380) {
        contingency(l.id) {
            equipments l.id
        }
    }
}
```

In the following example, we use the Stream API to create a list of contingencies with the 3 first 225 kV french lines:

```
import com.powsybl.iidm.network.Country

network.lineStream
        .filter({ l -> l.terminal1.voltageLevel.substation.country == Country.FR })
        .filter({ l -> l.terminal2.voltageLevel.substation.country == Country.FR })
        .filter({ l -> l.terminal1.voltageLevel.nominalV == 225 })
        .filter({ l -> l.terminal2.voltageLevel.nominalV == 225 })
        .limit(3)
        .forEach({ l ->
            contingency(l.id) {
                equipments l.id
            }
        })
```

The following example creates a list of contingencies with the 3 first French nuclear generators with a maximum power greater than 1000 MW.

```
import com.powsybl.iidm.network.Country
import com.powsybl.iidm.network.EnergySource

network.generatorStream
        .filter({ g -> g.terminal.voltageLevel.substation.country == Country.FR })
        .filter({ g -> g.energySource == EnergySource.NUCLEAR })
        .filter({ g -> g.maxP > 1000 })
        .limit(3)
        .forEach({ g ->
            contingency(g.id) {
                equipments g.id
            }
        })
```

Configuration

To provide a contingency list using this DSL, you have to add the following lines to your configuration file:

YAML configuration:

```
componentDefaultConfig:
  ContingenciesProviderFactory: com.powsybl.contingency.dsl.
↪GroovyDslContingenciesProviderFactory

groovy-dsl-contingencies:
  dsl-file: /path/to/contingencies.groovy
```

XML configuration:

```
<componentDefaultConfig>
    <ContingenciesProviderFactory>com.powsybl.contingency.dsl.
↪GroovyDslContingenciesProviderFactory</ContingenciesProviderFactory>
</componentDefaultConfig>
<groovy-dsl-contingencies>
    <dsl-file>/path/to/contingencies.groovy</dsl-file>
</groovy-dsl-contingencies>
```

Going further

- *Action DSL*: Lean how to write scripts for security analyses with remedial actions

4.3.3 Action DSL

The action DSL is a domain-specific language written in groovy for the creation of a strategy to solve violations, used in *security analyses with remedial actions*. This strategy is constituted of a set of *contingencies*, a set of modification on the network called **actions**, and a set of rules to determine in which circumstances to apply the actions.

Contingencies

The contingencies are defined in the same way described in the *contingency DSL*. Are supported:

- N-1 contingencies that trigger a single piece of equipment at a time
- N-K contingencies that triggers multiple equipments at a time
- busbar contingencies that trigger a *busbar section*, a N-K contingency that triggers all the equipment connected to that busbar section.

Actions

The actions are modifications that could be made on the network to solve a security issue. It could be topology modification like opening or closing a switch, setpoint modification, tap position modification. PowSyBl provides a set of predefined actions, but this possibility is infinite as you can create custom actions or groovy scripts.

An action is constituted of:

- a unique identifier to reference this action in the *rules*
- an optional description to explain the purpose of the action
- a list of tasks that are executed when the action is applied

The following snippet shows how to create an action:

```
action('actionID') {
    description "A short description for this actions"
    tasks {
        // Put here the tasks to execute when the action is applied
    }
}
```

Opening / Closing a switch

Topology changes are usually used to reduce the intensity on an AC branch. The following snippet shows how to create an action that opens the switch `SW1` and another one to close it.

```
action('open-switch-SW1') {
    description "Open switch SW1"
    tasks {
        openSwitch 'SW1'
    }
}

action('close-switch-SW1') {
    description "Close switch SW1"
    tasks {
        closeSwitch 'SW1'
    }
}
```

Note that it's possible to open or close several switches at a time:

```
action('open-SW1-and-SW2') {
    tasks {
        openSwitch 'SW1'
        openSwitch 'SW2'
    }
}
```

Generator modification

The `generatorModification` task is a task that can modify the setpoints and the regulation mode of a *generator*. It supports the modification of:

- the active power limits
- the active power setpoints as an absolute value or with an increment
- the voltage setpoint, the reactive power setpoint and the regulation mode
- the connection status

```
action('change-active-power-limits') {
    description "Change the active power limits of generator GEN to [0, 100]"
    tasks {
        generatorModification('GEN') {
            minP 0
            maxP 100
        }
    }
}

action('change-active-power-setpoint') {
    description "Change the active power setpoint of generator GEN to 100"
    tasks {
        generatorModification('GEN') {
            targetP 100
        }
    }
}
```

(continues on next page)

(continued from previous page)

```
action('increment-active-power-setpoint') {
    description "Increment the active power setpoint of generator GEN by 10"
    tasks {
        generatorModification('GEN') {
            deltaTargetP 10
        }
    }
}

action('change-regulation-mode') {
    description "Change the regulation's mode to voltage and change the setpoint"
    tasks {
        generatorModification('GEN') {
            voltageRegulatorOn true
            targetV 400
            targetQ 0
        }
    }
}

action('disconnect-generator') {
    description "Disconnect the generator GEN"
    tasks {
        generatorModification('GEN') {
            connected false
        }
    }
}
```

Changing a phase tap changer position

Changing the tap position of a phase tap changer is really useful to change how the active power is spread over parallel branches.

phaseShifterFixedTap

The `phaseShifterFixedTap` task is used to set the tap position to a fixed value. As power flow simulator could change the tap position during the simulation, it's necessary to also change the regulating value to false in this case.

```
action('fix-tap-position') {
    description "Set the tap position of TWT to tap 10"
    tasks {
        phaseShifterFixedTap('TWT', 10)
    }
}
```

phaseShifterTap

The `phaseShifterTap` task is used to increment or decrement the tap position. As power flow simulator could change the tap position during the simulation, it's necessary to also change the regulating value to false in this case. If the new tap position is lower than the minimal tap position, or greater than the maximal tap position, the tap position is adjusted to be in the bounds.

```
action('increment-tap-position') {
    description "Increment the tap position of TWT by 4"
    tasks {
        phaseShifterTap('TWT', 4)
    }
}

action('decrement-tap-position') {
    description "Decrement the tap position of TWT by 4"
    tasks {
        phaseShifterTap('TWT', -4)
    }
}
```

optimizePhaseShifterTap

The `optimizePhaseShifterTap` task is used to change the tap position of a phase tap changer until the intensity is closest to the limit but does not exceed it. This task runs a *load flow* each time the tap is changed to compute the new intensity value of the PST.

```
action('optimize-tap-position') {
    description "Find the tap position to be closest to the limit"
    tasks {
        optimizePhaseShifterTap 'TWT'
    }
}
```

As this task relies on a power flow simulator, this task needs to be configured in the configuration. If the simulator's name is not specified, the default one is used.

Example in YAML

```
load-flow-based-phase-shifter-optimizer:
    load-flow-name: Default
```

Example in XML

```
<load-flow-based-phase-shifter-optimizer>
    <load-flow-name>Default</load-flow-name>
</load-flow-based-phase-shifter-optimizer>
```

Scripting

The `script` task allow you to execute groovy code to modify the network. You can access to the network and the computation manager, using the `network` and `computationManager` variables. With this task, possibilities are unlimited as you have complete access to the IIDM API.

```
action('custom-action') {
    description "Disconnect LOAD1 and LOAD2, open the coupler COUPLER and change the␣
↪setpoints of LOAD3"
    tasks {
        script {
            network.getLoad("LOAD1").getTerminal().disconnect()
```

(continues on next page)

(continued from previous page)

```
            network.getLoad("LOAD2").getTerminal().disconnect()

            network.getSwitch("COUPLER").setOpen(true)

            network.getLoad("LOAD3").setP0(100).setQ0(60)
        }
    }
}
```

Rules

The rules are the most important in this DSL: they define the activation criteria of the actions. A rule is constituted of:

- a unique identifier
- an optional description to explain the purpose of the rule
- an activation criteria
- a list of actions to be applied if the activation criteria is verified
- an optional life count to limit the number of times a rule can be verified and its actions applied

Activation criteria

The activation criteria is a logical expression using the network's API and arithmetic operations on its variables. To create actions scripts more easily, a set of predefined functions can be used, but it's also possible to create custom ones.

```
rule('rule-ID') {
    description "A short description"
    when contingencyOccured()
    apply 'action1', 'action2'
    life 1
}
```

Note: The activation criteria are evaluated during the simulation. If you want to save the initial value of a network's variable, you should declare a global variable at the beginning of your script.

Network binding

The network binding adds keywords in the DSL to get equipment by their IDs. At the moment, the following keywords are supported:

- `line`: to retrieve a *line*
- `transformer`: to retrieve a *two-winding transformer*
- `branch`: to retrieve a *line*, a *tie line* or a *two-winding transformer*
- `generator`: to retrieve a *generator*
- `load`: to retrieve a *load*
- `_switch` and `switch_` to retrieve a *switch*

Note: the `switch` keyword is reserved in Groovy, so pay attention to prefix or postfix with an underscore.

Note: if you try to access to an undefined characteristic, a property is automatically created. Be very careful to typography mistakes because it could lead to unexpected results of your simulation.

Predefined functions

The following predefined functions are available and can be used in the `when` statement:

- `actionTaken`: returns `true` if the given action has already been applied
- `contingencyOccured`: returns `true` if a contingency is currently simulated, and `false` if the N state is simulated
- `loadingRank`: returns the rank of a given branch among a list of branches regarding their overload level
- `mostLoaded`: returns the ID of the most loaded branch among a list of branches
- `isOverloaded`: returns `true` if at least one of the given branches is overloaded
- `allOverloaded`: returns `true` if all the given branches are overloaded

Examples In this first example, when a contingency has occurred and `action1` has already been applied, we apply `actions2`:

```
action('action1') {
}

action('action2') {
}

rule('example1') {
    description "Apply action2 if a contingency has occured, and action1 has been applied
↪"
    when contingencyOccured() and actionTaken('action1')
    apply 'action2'
}
```

In this example, we disconnect the line `LINE1` if it's the most loaded of `LINE1`, `LINE2` and `LINE3`:

```
action('disconnect-line1') {
    tasks {
        script {
            line('LINE1').getTerminal1().disconnect()
            line('LINE1').getTerminal2().disconnect()
        }
    }
}

rule('example2') {
    description "Disconnect the line LINE1 if it is the most loaded"
    when contingencyOccured() and mostLoaded("LINE1", ["LINE1", "LINE2", "LINE3"])
    apply 'disconnect-line1'
}
```

In this second example, when both `LINE1` and `LINE2` are overloaded, we change the tap position of the PST `TWT`:

```
action('change-tap-position') {
    tasks {
        phaseShifterFixedTap('TWT', 10)
    }
}
```

(continues on next page)

(continued from previous page)

```
rule('example3') {
    description "If LINE1 and LINE2 are overloaded then change the PST tap position"
    when allOverloaded(["LINE1", "LINE2"])
    apply 'change-tap-position'
}
```

Dry-run mode

TODO

Configuration

TODO

Going further

TODO

4.3.4 Limit reductions

General description

Limit reductions can be specified in order to detect when a specific limit is **nearly** reached, without having to artificially modify the limit itself. For instance, with a limit reduction set to 95% for a limit of 1000 MW, the security analysis will flag a limit violation for any value exceeding 950 MW.

Each limit reduction has its own criteria specifying for which limits and under what conditions it should be applied. These criteria can include:

- the type of limit (current, active power or apparent power)
- the use case: for monitoring only or also for applying remedial actions
- the contingency context (pre-contingency, after a specific contingency or after all contingencies, etc.)
- the network elements targeted by the reduction (branches, three-winding transformers, ...), which can be described by the following criteria:
  - a set of their ids;
  - their countries;
  - their nominal voltages.
- which operational limits are affected by the reduction:
  - the severity of the limit: permanent or temporary;
  - and for temporary limits, their acceptable duration:
    - equal to a specific value;
    - inside an interval.

These criteria can be cumulative; multiple criteria can be used simultaneously to define a limit reduction.

Since a network operational limit may meet the criteria of several limit reductions, the order in which these reductions are declared is important: the last one encountered when reading them from start to finish is applied.

Criteria details

Limit type

The type of limits targeted by the reduction must be specified (mandatory item). The supported types are:

- `CURRENT`: for current limits;
- `ACTIVE_POWER`: for active power limits;
- `APPARENT_POWER`: for apparent power limits.

Use cases (monitoring or action)

The reduction may affect results:

1. Monitoring only (`monitoringOnly` set to `true`) means that if reductions are provided in a security analysis, only reported violations are affected.
2. Else (`monitoringOnly` set to `false`) if reductions are provided in a security analysis, they affect not only the reported violations but also the conditions for applying remedial actions.

Contingency context

A contingency context can be optionally specified. It contains:

- a type among:
  - `ALL`: corresponding to all contingencies and pre-contingency situations;
  - `NONE`: corresponding to pre-contingency situations;
  - `SPECIFIC`: corresponding to a specific contingency situation;
  - `ONLY_CONTINGENCIES`: corresponding to all contingency situations (without the pre-contingency one).
- and when the type is `SPECIFIC`, the id of the contingency.

When no contingency context is present, the `ALL` policy is used.

Network elements

The network elements whose limits will be affected by the limit reductions can be selected in using several criteria:

- a set of the network elements' ids;
- one or two countries (respectively for elements with one or two substations);
- their nominal voltages, by defining an interval for each of the voltage levels.

If no network elements are specified, the limit reduction applies to all of them.

Limit duration criteria

Duration criteria can be optionally specified. It contains:

- a type among:
  - `PERMANENT`: corresponding to permanent limits only;
  - `TEMPORARY`: corresponding to temporary limits only.
- and when the type is `TEMPORARY`, one of the following options to restrict them accordingly to their acceptable duration:

  - `ALL`: to select all temporary limits, regardless their acceptable duration;
  - `EQUALITY`: to select the temporary limits whose acceptable duration is equal to a specified value, with:
    * `value`: the said value;
  - `INTERVAL`: to select the temporary limits whose acceptable duration is within an interval, with:
    * `lowBound` and `highBound`: minimum and maximum duration, each can be null;
    * `lowClosed` and `highClosed`: to indicate if the interval is open (`false`) or closed (`true`) on respectively the lower and the upper boundaries. This attribute is facultative if the corresponding bound value is `null`.

When no duration criteria are present, the reduction is applied to all permanent and temporary limits.

When several duration criteria are specified, the limit reductions apply to each one. For instance, if both criteria `PERMANENT` and (`TEMPORARY` ; `EQUALITY`: 600) are defined, the limit reduction will apply to permanent limits and 600 s limits.

The security analysis is a simulation that checks violations on a network. These checks can be done on the base case or after a contingency, with or without remedial actions. A security analysis can monitor network states, in pre-contingency state, after a contingency and after a remedial action.

There is a violation if the computed value is greater than the maximum allowed value. Depending on the equipment, the violations can have different types:

- Current, active power and apparent power: this kind of violation can be detected on a branch (line, two-winding transformer, tie line) or on a three-winding transformer, if the computed value is greater than its *permanent limit* or one of its *temporary limits*.
- Voltage: this kind of violation can be detected on a bus or a bus bar section, if the computed voltage is out of the low-high voltage limits of a *voltage level*.
- Voltage angle: this kind of violation can be detected if the voltage angle difference between the buses associated to two terminals is out of the low-high voltage angle limits defined in the network.

4.3.5 Inputs

Network

The first input of the security analysis is a network. As this simulation is based on a *load flow* engine for a list of contingencies, this network should converge in the pre-contingency state.

Contingencies

The security analysis needs a list of contingencies as an input. When contingencies are provided, the violations are detected on the network at pre-contingency state, but also after applying each contingency. The supported elementary contingencies are:

- Generator contingency
- Static var compensator contingency
- Load contingency
- Bus contingency for bus/breaker topologies
- Busbar section contingency for node/breaker topologies
- Line, two-winding transformer and tie line contingencies (branch contingency)
- Three-winding transformer contingency
- Boundary line contingency

- HVDC line contingency
- Battery contingency
- Shunt Compensator contingency
- Switch contingency
- DC line contingency
- Voltage source converter contingency
- DC ground contingency
- DC node contingency

A contingency is made of contingency elements. A contingency can trigger one element at a time (N-1) or several elements at a time (N-K). Bus bar and bus contingencies are special N-K contingencies as they trigger all the equipments connected to a given bus bar section.

Operator strategies

An operator strategy is applied in pre-contingency or after a contingency, depending on the contingency context provided. A contingency context can be:

- `NONE` a pre-contingency state only
- `SPECIFIC` a post-contingency state on a specific contingency
- `ONLY_CONTINGENCIES` a post-contingency state on every contingency
- `ALL` both pre-contingency and post-contingency states

An operator strategy groups a condition and a list of remedial actions.

Remedial actions

Remedial actions are actions that are applied when limit violations occur. Supported actions are:

Action	Description
`LoadAction`	Change the relative or absolute `P0` and/or `Q0` of a load.
`BoundaryLineActi`	Change the relative or absolute `P0` and/or `Q0` of the load part of a boundary line.
`SwitchAction`	Open or close a switch.
`TerminalsConnecti`	Open or close a terminal.
`PhaseTapChangerTa`	Change the tap of a phase tap changer.
`RatioTapChangerTa`	Change the tap of a ratio tap changer.
`PercentChangeLoa`	Change the active and/or reactive power of a load (by setting the values, or defining changes in value or percentage).
`ShuntCompensatorI`	Change the section of a shunt compensator.
`PhaseTapChangerRe`	Change the regulation status of a phase tap changer.
`RatioTapChangerRe`	Change the regulation status of a ratio tap changer.
`GeneratorAction`	Change `targetP`, `targetQ`, regulation status and `targetV` of a generator.
`StaticVarCompensa`	Change the regulation mode of a static var compensator and its associated set point.
`HvdcAction`	Enabled or disabled AC emulation for HVDC line (with the possibility to change `P0` and `droop` for AC emulation and active power set point and converter mode for set point operating mode).
`AreaInterchangeTa`	Change the interchange target of an area by specifying a new interchange target in MW.

Remedial actions can be *preventive* or *curative*:

- preventive: these actions are implemented before the violation occurs, for example if the flow of a monitored line is between `90%` and `100%`. Use contingency context `NONE` for that.
- curative: these actions are implemented after a violation occurs, for example, if the flow of the monitored line is greater than `100%`.

Note: you can find the current list of remedial actions implemented in the PowSyBl Open Load Flow security analysis provider in the PowSyBl Open Load Flow documentation.

Conditions

Actions are applied if a condition is met. The conditions can be diversified and extended in the future:

- True condition: meaning that the list of actions is applied.
- All violations condition on a list of elements: meaning that the list of actions is applied only if all elements provided are overloaded.
- At least one violation condition: meaning that the list of actions is applied only if a violation occurs on the network.
- Any violation condition on a list of elements: meaning that the list of actions is applied if one or more elements provided are overloaded.
- Threshold condition: condition triggered when a threshold violation is detected on the network. The threshold can refer to an active power, reactive power, current or target P value on a specific point of the network. Four condition types are available depending on the equipment to target: branch threshold condition, three-winding transformer threshold condition, injection threshold condition and AC/DC converter threshold condition.

State monitors

A stateMonitor allows getting information about branch, bus and three-winding transformers on the network after a security analysis computation. Contingency context allows to specify if the information asked are about pre-contingency state or post-contingency state with a contingency id or both. For example:

- If we want information about a branch after security analysis on contingency `c1`, the contingencyContext will contain the contingencyId `c1`, contextType `SPECIFIC` and the state monitor will contain the id of the branch.
- If we want information about a branch in pre-contingency state, the contingencyContext will contain a null contingencyId, contextType `NONE` and the state monitor will contain the id of the branch.
- If we want information about a branch in pre-contingency state and after security analysis on contingency `c1`, the contingencyContext will contain contingencyId `c1`, contextType `ALL` and the state monitor will contain the id of the branch.

Limit reductions

Limit reductions can be specified in order to detect when a specific limit is **nearly** reached, without having to artificially modify the limit itself. For instance, with a limit reduction set to 95% for a limit of 1000 MW, the security analysis will flag a limit violation for any value exceeding 950 MW.

Each limit reduction has its own criteria specifying for which limits and under what conditions it should be applied. These criteria can include:

- the type of limit (current, active power or apparent power);
- the use case (for monitoring only or also for applying remedial actions);
- the contingency context (pre-contingency, after a specific contingency or after all contingencies, etc.);
- the network elements targeted by the reduction (by ids, countries and/or nominal voltages);
- which operational limits are affected by the reduction (permanent or temporary + acceptable duration).

You can find more details about limit reductions *here*.

4.3.6 Outputs

Pre-contingency results

The violations are detected on the network at state N, meaning before a contingency occurred. This determines a reference for the simulation. For each violation, we get the ID of the overloaded equipment, the limit type (`CURRENT`, `ACTIVE_POWER`, `APPARENT_POWER`, `LOW_VOLTAGE` or `HIGH_VOLTAGE`, `LOW_VOLTAGE_ANGLE` or `HIGH_VOLTAGE_ANGLE`), the acceptable value and the computed value. For branches and three-winding transformers, we also have the side where the violation has been detected.

The pre-contingency results also contain the network results based on given state monitors. A network result groups branch results, bus results and three-winding transformer results. All elementary results are fully extendable.

Post-contingency results

The post-contingency results contain the complete list of the contingencies that have been simulated, and for each of them the violations detected. To limit information to the user, only new violations or worsened violations can be listed.

The post-contingency results also contain the network results based on given state monitors.

Operator strategy results

The operator strategy results contain the complete list of the actions that have been simulated, and for each of them the violations detected in order to check if remedial actions were efficient.

The operator strategy results also contain the network results based on given state monitors.

Active power distribution results

Pre-contingency, post-contingency, and operator strategy results all report the volume of needed active power balancing, in MW:

- for pre-contingency: if the input network is not properly balanced, how much slack active power was distributed to balance the base case.
- for post-contingency: how much *additional* slack active power was distributed compared to the pre-contingency state because of:
  - losses changes,
  - injections (generators, loads, ...) disconnected by the contingency, if any.
- for operator strategy actions results, how much *additional* slack active power was distributed compared to the post-contingency state because of:
  - losses changes,
  - injections (generators, loads, ...) changes by the operator strategy actions, if any, such as disconnections, reconnections, or setpoint modifications.

Extensions

The results of a security analysis are extendable, meaning you can have additional information attached to the network, the contingencies or the violations.

Example

The following example is a result of a security analysis with remedial action, exported in JSON:


```
{
  "version" : "1.9",
  "network" : {
    "id" : "sim1",
    "sourceFormat" : "test",
    "caseDate" : "2018-01-01T11:00:00.000+01:00",
    "forecastDistance" : 0
  },
  "preContingencyResult" : {
    "status" : "CONVERGED",
    "limitViolationsResult" : {
      "limitViolations" : [ {
        "subjectId" : "NHV1_NHV2_1",
        "operationalLimitsGroupId" : "activated_1_1",
        "limitType" : "CURRENT",
        "limit" : 100.0,
        "limitReduction" : 0.95,
        "value" : 110.0,
        "side" : "ONE",
        "extensions" : {
          "ActivePower" : {
            "value" : 220.0
          }
        }
      } ],
      "actionsTaken" : [ ]
    },
    "networkResult" : {
      "branchResults" : [ {
        "branchId" : "branch1",
        "p1" : 1.0,
        "q1" : 2.0,
        "i1" : 3.0,
        "p2" : 1.1,
        "q2" : 2.2,
        "i2" : 3.3
      }, {
        "branchId" : "branch2",
        "p1" : 0.0,
        "q1" : 0.0,
        "i1" : 0.0,
        "p2" : 0.0,
        "q2" : 0.0,
        "i2" : 0.0,
        "flowTransfer" : 10.0
      } ],
      "busResults" : [ {
        "voltageLevelId" : "voltageLevelId",
        "busId" : "busId",
        "v" : 400.0,
```


(continues on next page)

(continued from previous page)


```
        "angle" : 3.14
      } ],
      "threeWindingsTransformerResults" : [ {
        "threeWindingsTransformerId" : "threeWindingsTransformerId",
        "p1" : 1.0,
        "q1" : 2.0,
        "i1" : 3.0,
        "p2" : 1.1,
        "q2" : 2.1,
        "i2" : 3.1,
        "p3" : 1.2,
        "q3" : 2.2,
        "i3" : 3.2
      } ]
    },
    "distributedActivePower" : 1.23
  },
  "postContingencyResults" : [ {
    "contingency" : {
      "id" : "contingency",
      "elements" : [ {
        "id" : "NHV1_NHV2_2",
        "type" : "BRANCH",
        "voltageLevelId" : "VLNHV1"
      }, {
        "id" : "NHV1_NHV2_1",
        "type" : "BRANCH"
      }, {
        "id" : "GEN",
        "type" : "GENERATOR"
      }, {
        "id" : "BBS1",
        "type" : "BUSBAR_SECTION"
      } ]
    },
    "status" : "CONVERGED",
    "limitViolationsResult" : {
      "limitViolations" : [ {
        "subjectId" : "NHV1_NHV2_2",
        "operationalLimitsGroupId" : "activated_2_1",
        "limitType" : "CURRENT",
        "limitName" : "20'",
        "acceptableDuration" : 1200,
        "limit" : 100.0,
        "limitReduction" : 1.0,
        "value" : 110.0,
        "side" : "TWO",
        "extensions" : {
          "ActivePower" : {
            "preContingencyValue" : 220.0,
            "postContingencyValue" : 230.0
          },
```


(continues on next page)

(continued from previous page)

```
        "Current" : {
          "preContingencyValue" : 95.0
        }
      }
    }, {
      "subjectId" : "GEN",
      "violationLocation" : {
        "type" : "BUS_BREAKER",
        "busIds" : [ "BUSID" ]
      },
      "limitType" : "HIGH_VOLTAGE",
      "limit" : 100.0,
      "limitReduction" : 0.9,
      "value" : 110.0
    }, {
      "subjectId" : "GEN2",
      "violationLocation" : {
        "type" : "NODE_BREAKER",
        "voltageLevelId" : "VL",
        "nodes" : [ 0, 1 ]
      },
      "limitType" : "LOW_VOLTAGE",
      "limit" : 100.0,
      "limitReduction" : 0.7,
      "value" : 115.0,
      "extensions" : {
        "Voltage" : {
          "preContingencyValue" : 400.0
        }
      }
    }, {
      "subjectId" : "NHV1_NHV2_2",
      "limitType" : "ACTIVE_POWER",
      "limitName" : "20'",
      "acceptableDuration" : 1200,
      "limit" : 100.0,
      "limitReduction" : 1.0,
      "value" : 110.0,
      "side" : "ONE"
    }, {
      "subjectId" : "NHV1_NHV2_2",
      "limitType" : "APPARENT_POWER",
      "limitName" : "20'",
      "acceptableDuration" : 1200,
      "limit" : 100.0,
      "limitReduction" : 1.0,
      "value" : 110.0,
      "side" : "TWO"
    } ],
    "actionsTaken" : [ "action1", "action2" ]
  },
  "networkResult" : {
```

(continues on next page)

(continued from previous page)

```
      "branchResults" : [ ],
      "busResults" : [ ],
      "threeWindingsTransformerResults" : [ ]
    },
    "connectivityResult" : {
      "createdSynchronousComponentCount" : 0,
      "createdConnectedComponentCount" : 0,
      "disconnectedLoadActivePower" : 0.0,
      "disconnectedGenerationActivePower" : 0.0,
      "disconnectedElements" : [ ]
    },
    "distributedActivePower" : 2.34
  } ],
  "operatorStrategyResults" : [ {
    "operatorStrategy" : {
      "id" : "strategyId",
      "contingencyContextType" : "SPECIFIC",
      "contingencyId" : "contingency1",
      "conditionalActions" : [ {
        "id" : "stage1",
        "condition" : {
          "type" : "AT_LEAST_ONE_VIOLATION",
          "violationIds" : [ "violationId1" ]
        },
        "actionIds" : [ "actionId1" ]
      } ]
    },
    "conditionalActionsResults" : [ {
      "conditionalActionsId" : "strategyId",
      "status" : "CONVERGED",
      "limitViolationsResult" : {
        "limitViolations" : [ ],
        "actionsTaken" : [ ]
      },
      "networkResult" : {
        "branchResults" : [ ],
        "busResults" : [ ],
        "threeWindingsTransformerResults" : [ ]
      },
      "distributedActivePower" : 3.45
    } ]
  } ]
}
```

4.3.7 Implementations

The following power flow implementations are supported:

- PowSyBl OpenLoadFlow
- Dynaflow

4.3.8 Going further

- *Run a security analysis through an iTools command*: Learn how to perform a security analysis from the command line.

4.4 Dynamic security analysis

4.4.1 Configuration

Implementation

If you have several implementations in your classpath, you need to choose which implementation to use in your configuration file with the `default-impl-name` property. Each implementation is identified by its name, that may be unique in the classpath:

- Use "Dynawo" to use powsybl-dynawo implementation

YAML configuration:

```
dynamic-security-analysis:
  default-impl-name: Mock
```

XML configuration:

```
<dynamic-security-analysis>
  <default-impl-name>Mock</default-impl-name>
</dynamic-security-analysis>
```

Parameters

The `dynamic-security-analysis-default-parameters` module is used every time a dynamic security analysis is run. It defines the default values for the most common parameters a `com.powsybl.security.dynamic.DynamicSecurityAnalysis` implementation should be able to handle. In addition to its own set of parameters, the dynamic security analysis reuses *dynamic simulation parameters*.

You may configure some generic parameters for all implementations:

```
dynamic-simulation-default-parameters:
  startTime: 0
  stopTime: 100

dynamic-security-analysis-default-parameters:
  contingencies-start-time: 10
  debugDir: /tmp/debugDir
```

The parameters may also be overridden with a JSON file, in which case the configuration will look like:

```
{
  "version" : "1.0",
  "dynamic-simulation-parameters" : {
    "version" : "1.0",
    "startTime" : 0.0,
    "stopTime" : 20.5
  },
  "contingencies-parameters" : {
    "contingencies-start-time" : 5.5
```

(continues on next page)

(continued from previous page)

```
  },
  "debugDir": "/tmp/debugDir"
}
```

Optional properties

contingencies-start-time `contingencies-start-time` defines when the contingencies start, in seconds. The default value of this property is `5`.

debugDir This property specifies the directory path where debug files will be dumped. If `null`, no file will be dumped.

Examples

YAML configuration:

```
dynamic-security-analysis-default-parameters:
  contingencies-start-time: 10
  debugDir: /tmp/debugDir
```

XML configuration:

```
<dynamic-security-analysis-default-parameters>
  <contingencies-start-time>10</contingencies-start-time>
  <debugDir>/tmp/debugDir</debugDir>
</dynamic-security-analysis-default-parameters>
```

The dynamic security analysis is a *security analysis* using dynamic models associated with static equipment of the network.

4.4.2 Inputs

Dynamic models configuration

The dynamic models configuration is exactly the same *configuration* used for a dynamic simulation.

Event models configuration

The event models configuration is exactly the same *configuration* used for a dynamic simulation.

Other inputs

Besides dynamic models configuration, the dynamic security analysis requires the same *inputs as the standard one*.

4.4.3 Outputs

The dynamic security analysis produces the same outputs as the standard one. All outputs can be found *here*.

4.4.4 Implementations

For the moment, the only available implementation is provided by `powsybl-dynawo`, which links PowSyBl with Dynao open source suite.

4.4.5 Going further

- Security analysis *Action DSL*.
- Security analysis *Contingency DSL*.
- *Run a dynamic security analysis through an iTools command*: Learn how to perform a security analysis from the command line.
- *List dynamic simulation models with an iTools command*: learn how to load a list of all dynamic simulation models from the command line.

4.5 Sensitivity analysis

4.5.1 Configuration

Implementation

If you have several implementations in your classpath, you need to choose which implementation to use in your configuration file:

```
sensitivity-analysis:
  default-impl-name: "<IMPLEMENTATION_NAME>"
```

Each implementation is identified by its name, that should be unique in the classpath. Use "OpenLoadFlow" to use PowSyBl OpenLoadFlow's sensitivity analysis implementation.

Parameters

flowFlowSensitivityValueThreshold

The `flowFlowSensitivityValueThreshold` is the threshold under which sensitivity values having a variable type among `INJECTION_ACTIVE_POWER`, `INJECTION_REACTIVE_POWER` and `HVDC_LINE_ACTIVE_POWER` and function type among `BRANCH_ACTIVE_POWER_1/2/3`, `BRANCH_REACTIVE_POWER_1/2/3` and `BRANCH_CURRENT_1/2/3` will be filtered from the analysis results. The default value is 0.0.

voltageVoltageSensitivityValueThreshold

The `voltageVoltageSensitivityValueThreshold` is the threshold under which sensitivity values having variable type `BUS_TARGET_VOLTAGE` and function type `BUS_VOLTAGE` will be filtered from the analysis results. The default value is 0.0.

flowVoltageSensitivityValueThreshold

The `flowVoltageSensitivityValueThreshold` is the threshold under which sensitivity values having a variable type among `INJECTION_REACTIVE_POWER` and function type among `BUS_VOLTAGE`, or variable type among `BUS_TARGET_VOLTAGE` and function type among `BRANCH_REACTIVE_POWER_1/2/3`, `BRANCH_CURRENT_1/2/3` or `BUS_REACTIVE_POWER` will be filtered from the analysis results. The default value is 0.0.

angleFlowSensitivityValueThreshold

The `angleFlowSensitivityValueThreshold` is the threshold under which sensitivity values having a variable type among `TRANSFORMER_PHASE` and `TRANSFORMER_PHASE_1/2/3` and a function type among `BRANCH_ACTIVE_POWER_1/2/3`, `BRANCH_REACTIVE_POWER_1/2/3` and `BRANCH_CURRENT_1/2/3` will be filtered from the analysis results. The default value is 0.0.

4.5.2 Introduction

The sensitivity analysis module is dedicated to computing the linearized impact of small network variations on the state variables of some equipments.

A sensitivity value is the numerical estimation of the partial derivative of the observed function with respect to the variable of impact. The sensitivity analysis can also be seen as the computation of partial derivatives on the network model. For example, it may be used to know among a group of selected lines, which are the most impacted by a change in a generator production or a change of tap on a phase tap changer. The user story about RSC capacity calculation provides an example of application of the sensitivity analysis.

4.5.3 Inputs

Network

The first input for the sensitivity analysis module is an IIDM network.

Sensitivity factors

Aside from providing an input network, it is necessary to specify which equipments are going to be studied:

- what impacted equipment is selected to be monitored (lines, for example)
- according to a change on which equipment (a generator's production or a group of generator's production, or the tap position of a phase tap changer, etc.)

It is also necessary to specify which quantity is being observed: the active power or the current on the monitored equipment, the voltage of a bus.

This set of information constitutes the sensitivity factors (`SensitivityFactor`). These factors correspond to the definition of the expected partial derivatives to be extracted from the input network. A standard sensitivity analysis input thus comprises a list of sensitivity factors, each one constituted of:

- a sensitivity variable (the variable of impact) which type is defined by a `SensitivityVariableType`.
- a sensitivity function (the observed function) which type is defined by a `SensitivityFunctionType`.
- a contingency context. Usually we compute the impact of an injection increase on a branch flow or current, the impact of a shift of a phase tap changer on a branch flow or current or the impact of a voltage target increase on a bus voltage.

A sensitivity variable represents a change on an equipment or on a group of equipments. The supported variable types are:

- Use `INJECTION_ACTIVE_POWER` to model a change on active production of a generator or on a group of generators, on the active consumption of a load or on a group of loads or on GLSK (for Generation and Load Shift keys) that describes a linear combination of active power injection shifts on generators and loads. The variable increase is in MW.
- Use `INJECTION_REACTIVE_POWER` to model a change on reactive production of a generator or on the reactive consumption of a load. The variable increase is in MVar.
- Use `TRANSFORMER_PHASE` to model the change of the tap position of a phase tap changer of a two-winding transformer. The increase is in degree.
- Use `BUS_TARGET_VOLTAGE` to model an increase of the voltage target of a generator, a static var compensator, a two or three-winding transformer, a shunt compensator or a VSC converter station. The increase is in KV.
- Use `HVDC_LINE_ACTIVE_POWER` to model the change of the active power set point of an HVDC line. The increase is in MW.
- Use `TRANSFORMER_PHASE_1`, `TRANSFORMER_PHASE_2` or `TRANSFORMER_PHASE_3` to model the change of the tap position of a phase tap changer of a three-winding transformer that contains several phase tap changers.

The supported sensitivity function types, related to the equipment to monitor, are:

- Use BRANCH_ACTIVE_POWER_1 and BRANCH_ACTIVE_POWER_2 if you want to monitor the active power in MW of a network branch (lines, two-winding transformer, boundary lines, etc.). Use 1 for side 1 and 2 for side 2. In case of a three-winding transformer, use BRANCH_ACTIVE_POWER_3 to monitor the active power in MW of leg 3 (network side).
- Use BRANCH_REACTIVE_POWER_1 and BRANCH_REACTIVE_POWER_2 if you want to monitor the reactive power in MVar of a network branch (lines, two-winding transformer, boundary lines, etc.). Use 1 for side 1 and 2 for side 2. In case of a three-winding transformer, use BRANCH_REACTIVE_POWER_3 to monitor the reactive power in MVar of leg 3 (network side).
- Use BRANCH_CURRENT_1 and BRANCH_CURRENT_2 if you want to monitor the current in A of a network branch (lines, two-winding transformer, boundary lines, etc.). Use 1 for side 1 and use 2 for side 2. In case of a three-winding transformer, use BRANCH_CURRENT_3 to monitor the current in A of leg 3 (network side).
- BUS_VOLTAGE if you want to monitor the voltage in KV of a specific network bus.
- BUS_REACTIVE_POWER if you want to monitor the reactive power injection in MVar of a specific network bus.

A sensitivity variable can group some equipment and has to be modeled as a variable set. In a SensitivityVariableSet, we have a list of individual variables, each one with a weight (called WeightedSensitivityVariable). We use variable sets to model what it commonly called GLSK.

How to provide the sensitivity factors input

The sensitivity factors may be created directly through Java code, or be provided to PowSyBl via a JSON file. This file should contain a list of JSON objects, each one representing a sensitivity factor. The example below shows how to write a JSON file to perform a sensitivity analysis on the active power through a line, with respect to a GLSK and to an injection on the network.

```
[ {
  "functionType" : "BRANCH_ACTIVE_POWER_1",
  "functionId" : "l45",
  "variableType" : "INJECTION_ACTIVE_POWER",
  "variableId" : "glsk",
  "variableSet" : true,
  "contingencyContextType" : "ALL"
}, {
  "functionType" : "BRANCH_ACTIVE_POWER_1",
  "functionId" : "l12",
  "variableType" : "INJECTION_ACTIVE_POWER",
  "variableId" : "g2",
  "variableSet" : false,
  "contingencyContextType" : "ALL"
} ]
```

Contingencies

The sensitivity analysis may also take, optionally, a list of contingencies as an input. When contingencies are provided, the sensitivity values shall be calculated on the network at state N, but also after the application of each contingency. The contingencies are provided in the same way as for the *security analysis*. This then constitutes a systematic sensitivity analysis.

```
{
  "type" : "default",
```

(continues on next page)

(continued from previous page)

```
  "version" : "1.0",
  "name" : "default",
  "contingencies" : [ {
    "id" : "l34",
    "elements" : [ {
      "id" : "l34",
      "type" : "BRANCH"
    } ]
  } ]
}
```

At the moment, the only available sensitivity simulator officially compatible with PowSyBl is the one available through OpenLoadFlow. In this case, the network is provided only once in state N, and then all the calculations are done successively by modifying the Jacobian matrix directly in the solver based on the contingency input. The network is thus loaded only once, which improves performance.

4.5.4 Outputs

Sensitivity values

The outputs of the sensitivity analysis are called sensitivity values. A sensitivity value represents an elementary result given a sensitivity factor and a contingency, and contains:

- The actual value of the partial derivative
- The reference value of the function at linearization point in case of contingency context `NONE` or in its post-contingency state in case of a factor associated to a contingency.

These results may be serialized in JSON format.

Example of interpretation

Let's imagine that one wants to compute the impact of an increase of active power generation of the generator G on the branch B. The sensitivity analysis input will contain one sensitivity factor, with sensitivity function type `BRANCH_ACTIVE_POWER_1` and sensitivity variable type `INJECTION_ACTIVE_POWER`, and we do not provide any input contingencies.

After the computation, let us consider that the values of the three elements of the sensitivity result are:

- a value of -0.05 for the partial derivative
- a variable reference value of 150
- a function reference value of 265

This can be interpreted in the following way:

- an increase of 100 MW on generator G may be approximated on branch B as a 5MW decrease of the active flow from side 1 to side 2
- the initial generation on generator G is 150MW
- the initial active flow on branch B is 265MW from side 1 to side 2

4.5.5 Implementations

The following sensitivity analysis implementations are supported:

- PowSyBl OpenLoadFlow

4.5.6 Going further

To go further about the sensitivity analysis, check the following content:

- Sensitivity analysis tutorial

4.6 Short-circuit analysis

When a short circuit occurs in a network, the currents on equipment can be so high that they exceed their rated values. Simulating faults on the network is important to verify that the short circuits are well detected and do not damage the equipment.

The short-circuit API allows the calculation of currents and voltages on a network after a fault. A first implementation of the API is available in powsybl-open-sc.

4.6.1 Parameters

Available parameters

The parameters to be used for the short-circuit calculation should be defined in the config.yml file. For example, here are some valid short-circuit parameters:

```
short-circuit-parameters:
  with-voltage-result: false
  with-feeder-result: true
  with-limit-violations: true
  study-type: TRANSIENT
  with-fortescue-result: false
  min-voltage-drop-proportional-threshold: 20
  with-loads: true
  with-shunt-compensators: true
  with-vsc-converter-stations: false
  with-neutral-position: true
  initial-voltage-profile-mode: CONFIGURED
  voltage-ranges: /path/to/voltage/ranges/file
  detailedReport: true
  debugDir: null
```

Available parameters in the short-circuit API are stored in `com.powsybl.shortcircuit.ShortCircuitParameters`. They are all optional.

with-limit-violations

This property indicates whether limit violations should be returned after the computation. The violations that should be used are `LOW_SHORT_CIRCUIT_CURRENT` and `HIGH_SHORT_CIRCUIT_CURRENT`. It can be used to filter results where the computed short-circuit current is too high or too low. The default value is `true`.

with-fortescue-result

This property indicates if the computed results, like currents and voltages, should be returned only in three-phased magnitude or detailed with magnitude and angle on each phase. According to this property, different classes to return results can be used. If it is set to false, the classes `MagnitudeFaultResult`, `MagnitudeFeederResult` and

MagnitudeShortCircuitBusResult should be used. If the property is true, the classes FortescueFaultResult, FortescueFeederResult and FortescueShortCircuitBusResult should be used. All these classes are in com.powsybl.shortcircuit. The default value is true.

with-feeder-result

This property indicates if the contributions of each feeder to the short-circuit current at the fault should be computed. If the property is set to true, the results can be stored in class com.powsybl.shortcircuit.FeederResult. The default value is true.

study-type

This property indicates the type of short-circuit study. It can be:

- SUB_TRANSIENT: it is the first stage of the short circuit, right when the fault happens. In this case, it is the sub-transient reactance of generators and batteries that is used. This reactance can either be stored in the network or calculated from the transient reactance of generators or batteries with a coefficient defined by the parameter sub-transient-coefficient.
- TRANSIENT: the second stage of the short circuit, before the system stabilizes. The transient reactance of generators and batteries will be used.
- STEADY_STATE: the last stage, once all transient effects are gone.

The default value is TRANSIENT. The transient and sub-transient reactances of the generators are stored in the *short-circuit generator extension.* and the ones of batteries are stored in the *short-circuit battery extension.*

sub-transient-coefficient

This property allows defining an optional coefficient, in case of a sub-transient study, to apply to the transient reactance of generators to get the sub-transient one:

$X''_d = c \times X'_d$

with:

- X''_d: the sub-transient reactance
- c: the sub-transient coefficient defined in this property
- X'_d: the transient reactance

By default, the value of the coefficient is 0.7, and it should not be higher than 1.

with-voltage-result

This property indicates if the voltage profile should be computed on every node of the network. The results, if this property is true, should be stored in class com.powsybl.shortcircuit.ShortCircuitBusResult. The default value is true.

min-voltage-drop-proportional-threshold

This property indicates a threshold to filter the voltage results. Thus, it only makes sense if with-voltage-result is set to true. Only the nodes where the voltage drop due to the short circuit in absolute value is above this property are kept. The voltage drop is calculated as the ratio between the initial voltage magnitude on the node and the voltage magnitude on the node after the fault. The default value is 0.

with-loads

This property indicates whether loads should be taken into account during the calculation. If true, the loads are modeled as an equivalent reactance, and the equivalent reactance at the fault point will be lowered. If false, then they will be ignored.

with-shunt-compensators

This property indicates if shunt compensators should be taken into account during the computation. If `true`, the shunt compensators will be modeled as an equivalent reactance. If `false`, then the shunt compensators will be ignored.

with-vsc-converter-stations

This property is a boolean property that indicates whether the VSC converter stations should be included in the calculation. If `true`, the VSC converter stations will be modeled as an equivalent reactance. If `false`, they will be ignored.

with-neutral-position

This property indicates which position of the tap changer of transformers should be used for the calculation. If `true`, the neutral step of the tap changer is used. The neutral step is the one for which $\rho = 1$ and $\alpha = 0$. If `false`, then the step that is in the model will be used. By default, this property is set to false. For more information about tap changers, see *the documentation about it*.

initial-voltage-profile-mode

This property defines the voltage profile that should be used for the calculation. Three options are available:

- `NOMINAL`: the nominal voltage profile is used for the calculation
- `PREVIOUS`: the voltage profile from the load flow is used for the calculation
- `CONFIGURED`: the voltage profile is specified by the user In the case of `CONFIGURED` voltage profile, ranges of nominal voltages with multiplicative coefficients must be specified in the `voltage-ranges` property. By default, the initial voltage profile mode is set to `NOMINAL`.

voltage-ranges

This property specifies a path to a JSON file containing the voltage ranges and associated coefficients to be used when `initial-voltage-profile-mode` is set to `CONFIGURED`. The JSON file must contain a list of voltage ranges and for each range, coefficients and/or voltages. Then, for each nominal voltage in the network that belongs to the range, the coefficient is applied to calculate the voltage to be used in the calculation. All the coefficients should be between 0.8 and 1.2. They are optional, and if they are missing for a range, 1 will be used. The voltage attribute of the voltage range defines the nominal voltage of all the voltage levels in the network that are in the range. It is also optional, and if it is not defined, then the nominal voltage already in the network will be used. Here is an example of this JSON file:

```
[
    {
      "minimumNominalVoltage": 350.0,
      "maximumNominalVoltage": 400.0,
      "voltageRangeCoefficient": 1.1,
      "voltage": 380.0
    },
    {
      "minimumNominalVoltage": 215.0,
      "maximumNominalVoltage": 235.0,
      "voltageRangeCoefficient": 1.2,
      "voltage": 225.0
    },
    {
      "minimumNominalVoltage": 80.0,
      "maximumNominalVoltage": 150.0,
      "voltageRangeCoefficient": 1.05,
      "voltage": 105.0
    }
]
```

detailedReport This property indicates whether the computation should produce detailed reporting. If `true`, detailed reports are returned. If `false`, summarized reports are returned.

debugDir This property specifies the directory path where debug files will be dumped. If `null`, no file will be dumped.

FaultParameters

It is possible to override parameters for each fault by creating an instance of `com.powsybl.shortcircuit.FaultParameters`. This object will take the fault to which it applies and all the parameters for this specific fault. One `FaultParameters` corresponds to one `Fault`. A list of `FaultParameters` can be given as an input to the API with specific parameters for one or multiple faults. If a fault has no `FaultParameters` corresponding, then the general parameters will be used.

4.6.2 Inputs

The API takes as inputs:

A network

It is the network on which the computation will be done.

A list of faults

The API takes as input a list of faults on which the calculation should be done. Faults on buses and on lines are supported. Each fault can either be an instance of `com.powsybl.shortcircuit.BusFault` or `com.powsybl.shortcircuit.BranchFault`.

The attributes to fill of a `BusFault` are:

Attribute	Type	Unit	Required	Default value	Description
id	String	-	yes	-	The id of the fault
elementId	String	-	yes	-	The id of the bus on which the fault will be simulated (bus/view topology)
r	double		no	0	The fault resistance to ground
x	double		no	0	The fault reactance to ground
connection	ConnectionType	-	no	`ConnectionTyp` `SERIES`	The way the resistance and reactance of the fault are connected to the ground: in series or in parallel
faultType	FaultType	-	no	`FaultType.` `THREE_PHASE`	The type of fault simulated: can be three-phased or single-phased

The attributes to fill of a `BranchFault` are:

Attribute	Type	Unit	Required	Default value	Description
id	String	-	yes	-	The id of the fault
elementId	String	-	yes	-	The id of the branch on which the fault will be simulated
r	double		no	0	The fault resistance to ground
x	double		no	0	The fault reactance to ground
connection	ConnectionType	-	no	`ConnectionTyp SERIES`	The way the resistance and reactance of the fault are connected to the ground: in series or in parallel
faultType	FaultType	-	no	`FaultType. THREE_PHASE`	The type of fault simulated: can be three-phased or single-phased
proportionalLocation	double	%	yes	-	The position where the fault should be simulated, in percent of the line

A list of FaultParameters

Optionally, it is possible to specify a list of `FaultParameters`. Each `FaultParameter` will override the default parameters for a given fault. For more information on parameters, see *above*.

4.6.3 Outputs

The results of the short-circuit analysis are stored in `com.powsybl.shortcircuit.ShortCircuitAnalysisResult`. This class gathers the results for every fault, they are accessible either by the ID of the fault or the ID of the element on which the fault is simulated. For each fault, an object `com.powsybl.shortcircuit.FaultResult` is returned.

Depending on `with-fortescue-result`, the returned result should either be an instance of `com.powsybl.shortcircuit.MagnitudeFaultResult` or `com.powsybl.shortcircuit.FortescueFaultResult`.

Both classes contain the following attributes:

Attribute	Type	Unit	Required	Default value	Description
fault	Fault	-	yes	-	The fault that was simulated
status	Status	-	yes	-	The status of the computation, see below for more details
shortCircuitPower	double	MVA	yes	-	The value of the short-circuit power
timeConstant	Duration	-	yes	-	The duration before reaching the permanent short-circuit current
feederResults	List	-	no	Empty list	A list of FeederResult, should not be empty if the parameter `with-feeder-result` is set to `true`.
limitViolations	List	-	no	Empty list	A list of LimitViolation, should be empty if the parameter `with-limit-violations` is set to `false`.
shortCircuitBusResults	List	-	no	Empty list	A list of ShortCircuitBusResult, should be empty if the parameter `with-voltage-result` is set to `false`.

However, in these classes, the short-circuit current and voltage are represented differently.

In `MagnitudeFaultResult`, the additional attributes are:

Attribute	Type	Unit	Required	Default value	Description
current	double	A	yes	-	The three-phased magnitude of the computed short-circuit current
voltage	double	kV	yes	-	The three-phased magnitude of the computed short-circuit voltage

In `FortescueFaultResult`, they are:

Attribute	Type	Unit	Required	Default value	Description
current	`FortescueVa`	A	yes	-	The magnitude and angle of the computed short-circuit current detailed on the three phases
voltage	`FortescueVa`	kV	yes	-	The magnitude and angle of the computed short-circuit voltage detailed on the three phases

The status of the computation

This status can be:

- `SUCCESS`: the computation went as planned, and the results are full considering the parameters.
- `NO_SHORT_CIRCUIT_DATA`: this status should be returned if no short-circuit data are available in the network, i.e., the sub-transient or transient reactance of generators or batteries and the minimum and maximum admissible short-circuit currents.
- `SOLVER_FAILURE`: the computation failed because of an error linked to the solver.
- `FAILURE`: the computation failed for any other reason.

FeederResults

This field contains the contributions of each feeder to the short-circuit current as a list. It should only be returned if `with-feeder-result` is set to `true`. Depending on the value of `with-fortescue-result`, it should be an instance of `com.powsybl.shortcircuit.MagnitudeFeederResult` or `com.powsybl.shortcircuit.FortescueFeederResult`.

The attributes of `MagnitudeFeederResults` are:

Attribute	Type	Unit	Required	Default value	Description
connectableId	String	-	yes	-	ID of the feeder
current	double	A	yes	-	Three-phased current magnitude of the feeder participating to the short-circuit current at the fault point
side	ThreeSides	-	no	-	If the feeder is a branch or a three-winding transformer, the side on which the result is

The attributes of `FortescueFeederResuts` are:

Attribute	Type	Unit	Required	Default value	Description
connectableI	String	-	yes	-	ID of the feeder
current	Fortescue	A	yes	-	Current magnitudes and angles on the three phases of the feeder participating to the short-circuit current at the fault point
side	ThreeSides	-	no	-	If the feeder is a branch or a three-winding transformer, the side on which the result is

Note: For results on branches, the side can be retrieved as a `TwoSides` object by using the method `getSideAsTwoSides`.

LimitViolations

This field contains a list of all the violations after the fault. This implies to have defined in the network the maximum or the minimum acceptable short-circuit currents on the voltage levels. Then, with comparing to the computed short-circuit current, two types of violations can be created: `LOW_SHORT_CIRCUIT_CURRENT` and `HIGH_SHORT_CIRCUIT_CURRENT`. This list should be empty if the property `with-limit-violations` is set to `false`.

ShortCircuitBusResults

This field contains a list of voltage results on every bus of the network after simulating the fault. It should be empty if `with-voltage-result` is set to `false`. The value of the property `with-voltage-drop-threshold` allows to filter these results by keeping only those where the voltage drop is higher than this defined threshold. Depending on the value of `with-fortescue-result`, the list should contain instances of either `com.powsybl.shortcircuit.MagnitudeShortCircuitBusResult` or `com.powsybl.shortcircuit.FortescueShortCircuitBusResult` objects.

The attributes of `MagnitudeShortCircuitBusResult` are:

Attribute	Type	Unit	Required	Default value	Description
voltageLevelId	String	-	yes	-	ID of the voltage level containing the bus
busId	String	-	yes	-	ID of the bus
initialVoltageMagnitude	double	kV	yes	-	Magnitude of the three-phased voltage before the fault
voltageDropProportional	double	%	yes	-	Voltage drop after the fault
voltage	double	kV	yes	-	Magnitude of the three-phased voltage after the fault

The attributes of `FortescueShortCircuitBusResult` are:

Attribute	Type	Unit	Required	Default value	Description
voltageLevelId	String	-	yes	-	ID of the voltage level containing the bus
busId	String	-	yes	-	ID of the bus
initialVoltageMagnitude	double	kV	yes	-	Magnitude of the three-phased voltage before the fault
voltageDropProportional	double	%	yes	-	Voltage drop after the fault
voltage	FortescueVa	kV	yes	-	Magnitudes and angles of the voltage on the three phases after the fault

CHAPTER

FIVE

DATA MODELS

In this section, you'll discover how data is modelled in PowSyBl.

5.1 Time series

5.1.1 Time series modeling

In PowSyBl, time series are modeled by:

- A name to uniquely identify a time series inside a store.
- A data type which is either `double` or `String`.
- A time index to define a list of instants for which data exists. Three different implementations of the time index are available in the framework, depending on the need:
  - Regular index: the time step size is constant
  - Irregular index: the time step size varies
  - Infinite index: the time series contains only two points, one at instant 0 and another at instant `Long.MAX_VALUE`
- Metadata: a list of key/value string data
- Data chunks: an ordered list of data that will be associated with instants of the time index. The data chunks may be compressed or uncompressed.

An uncompressed JSON data chunk looks like:

```
{
  "offset" : 0,
  "values" : [ 1.0, 1.0, 1.0, 3.0 ]
}
```

An uncompressed data chunk is modeled with a double (or String) array and an offset. It defines values associated to instants of the time index from `offset` to `offset + values.length`.

It is possible to compress the data chunks, using, for example, the RLE. The JSON serialization of compressed data chunks looks like: Output:

```
{
  "offset" : 0,
  "uncompressedLength" : 4,
  "stepValues" : [ 1.0, 3.0 ],
  "stepLengths" : [ 3, 1 ]
}
```

Time series can be imported from CSV data:

```
// Creating a map of TimeSeries per version by parsing a CSV file
Map<Integer, List<TimeSeries>> timeSeriesPerVersion = TimeSeries.parseCsv(pathToCSV);

// Creating a map of TimeSeries per version by parsing the CSV string with a specified␣
↪configuration
TimeSeriesCsvConfig timeSeriesCsvConfig = new TimeSeriesCsvConfig(';', true, TimeFormat.
↪MILLIS);
Map<Integer, List<TimeSeries>> timeSeriesPerVersion = TimeSeries.parseCsv(csvAsString,␣
↪timeSeriesCsvConfig);
```

5.1.2 Calculated time series

Starting from a double time series, it is possible to create calculated time series using a Groovy script.

For instance, let us consider the following example. Let's say we have created a first double time series named `dts` in a script, it is then possible to create new time series `a` and `b` by writing:

```
ts['a'] = ts['dts'] + 1
ts['b'] = ts['a'] * 2
```

The time series `a` and `b`, serialized in JSON format, then look like:

```
[ {
  "name" : "a",
  "expr" : {
    "binaryOp" : {
      "op" : "PLUS",
      "timeSeriesName" : "dts",
      "integer" : 1
    }
  }
}, {
  "name" : "b",
  "expr" : {
    "binaryOp" : {
      "op" : "MULTIPLY",
      "binaryOp" : {
        "op" : "PLUS",
        "timeSeriesName" : "dts",
        "integer" : 1
      },
      "integer" : 2
    }
  }
} ]
```

The calculated time series are evaluated on the fly during array conversion or iteration (through iterators or streams): only the arithmetic expression is stored.

Here is the list of supported vector operations:

Operator	Purpose	Example	Return type
`+`	addition	`ts['a'] + ts['b']`	Numerical
`-`	substraction	`ts['a'] - ts['b']`	Numerical
`*`	multiplication	`ts['a'] * ts['b']`	Numerical
`/`	division	`ts['a'] / ts['b']`	Numerical
`==`	1 if equals, 0 otherwise	`ts['a'] == ts['b']`	Boolean
`!=`	1 if not equals, 0 otherwise	`ts['a'] != ts['b']`	Boolean
`<`	1 if less than, 0 otherwise	`ts['a'] < ts['b']`	Boolean
`<=`	1 if less than or equals to, 0 otherwise	`ts['a'] <= ts['b']`	Boolean
`>`	1 if greater, 0 otherwise	`ts['a'] > ts['b']`	Boolean
`>=`	1 if greater than or equals to, 0 otherwise	`ts['a'] >= ts['b']`	Boolean
`-`	negation	`-ts['a']`	Numerical
`abs`	absolute value	`ts['a'].abs()`	Numerical
`time`	convert to time index vector (epoch)	`ts['a'].time()`	Numerical
`min`	min value	`ts['a'].min(10)`	Boolean
`max`	max value	`ts['a'].max(10)`	Boolean
`min`	point-to-point min values between timeseries	`min(ts['a'], ts['b'])`	Numerical
`max`	point-to-point max values between timeseries	`max(ts['a'], ts['b'])`	Numerical

In the Groovy DSL syntax, both `timeSeries['a']` and `ts['a']` are supported and are equivalent.

To compare a time index vector to a literal date, the `time('2018-01-01T00:00:01Z')` function is available. For instance, the following code creates a time series of 0 and 1 values:

```
a = ts['dts'].time() < time('2018-01-01T00:00:01Z')
```

CHAPTER

SIX

USER DOCUMENTATION

In this section, you'll discover the basics about PowSyBl: how to install it, run your first commands, configure it for your needs, etc.

6.1 Configuration

6.1.1 Modules

componentDefaultConfig

The `componentDefaultConfig` module is used to configure the implementation of plugins that the framework has to use for specific features (e.g. computation, etc.). Contrary to the other modules, it is impossible to give an exhaustive list of the existing properties.

The names of the properties are the names of Java interfaces of the powsybl framework. Each value must be the complete name of a class which implements this interface.

- ContingenciesProviderFactory
- MpiStatisticsFactory

ContingenciesProviderFactory implementations

The `com.powsybl.contingency.EmptyContingencyListProvider` is a special implementation of the `ContingenciesProvider` interface that provides an empty list of contingencies. This implementation should be use to check violations on a N-state.

Other implementations:

- `com.powsybl.action.dsl.GroovyDslContingenciesProviderFactory`
- `com.powsybl.contingency.dsl.GroovyDslContingenciesProviderFactory`

```
componentDefaultConfig:
    ContingenciesProviderFactory: com.powsybl.contingency.
↪EmptyContingencyListProviderFactory
```

```
<componentDefaultConfig>
    <ContingenciesProviderFactory>com.powsybl.action.dsl.
↪GroovyDslContingenciesProviderFactory</ContingenciesProviderFactory>
</componentDefaultConfig>
```

Example

In the configuration below, we define these functionalities:

- A description of contingencies

The chosen implementation is:

- The contingencies expressed in Groovy DSL language

YAML configuration:

```
componentDefaultConfig:
    ContingenciesProviderFactory: com.powsybl.action.dsl.
↪GroovyDslContingenciesProviderFactory
```

XML configuration

```
<componentDefaultConfig>
    <ContingenciesProviderFactory>com.powsybl.action.dsl.
↪GroovyDslContingenciesProviderFactory</ContingenciesProviderFactory>
</componentDefaultConfig>
```

Removed properties

load-flow-factory: The `load-flow-factory` property has been removed in PowSyBl 3.0.0. Use the `default-impl-name` property of the *load-flow* module instead.

SensitivityComputationFactory The `SensitivityComputationFactory` property has been removed in PowSyBl 3.0.0. Use the *sensitivity-analysis* module instead.

SensitivityFactorsProviderFactory The `SensitivityFactorsProviderFactory` property has been removed in PowSyBl 3.0.0. Use the *sensitivity-analysis* module and properties instead.

SecurityAnalysisFactory The `SecurityAnalysisFactory` property has been removed in PowSyBl 3.0.0. Use the *security-analysis* module instead.

computation-local

The `computation-local` module is used by the `com.powsybl.computation.local.LocalComputationManager` to run computations on the local host, if it is configured in the *default-computation-manager* module.

Optional properties

available-core The `available-core` property is an optional property that defines the maximum number of parallel computations. The default value of this property is `1`. To use all the processors of the system, set this property to `0`.

tmp-dir The `tmp-dir` property is an optional property that defines a list of paths where the temporary files generated during the computations can be stored. The temporary files will be generated in the first existing path of this list. If none of the paths exists, a `ConfigurationException` is thrown. The default value of this property is initialized with the `java.io.tmpdir` JVM system property.

Deprecated properties

availableCore The `availableCore` property is deprecated since v2.1.0. Use the `available-core` property instead.

tmpDir The `tmpDir` property is deprecated since v2.1.0. Use the `tmp-dir` property instead.

Examples

YAML configuration:

```
computation-local:
    available-core: 1
    tmp-dir:
      - /home/user/tmp
      - /tmp
```

XML configuration:

```
<computation-local>
    <available-core>1</available-core>
    <tmp-dir>/home/user/tmp:/tmp</tmp-dir>
</computation-local>
```

default-computation-manager

The `default-computation-manager` module is an optional module loaded when an `iTools` command starts, to determine which `com.powsybl.computation.ComputationManager` implementation should be used for short-time and long-time computations. The choice of using the short-time or the long-time computation manager factory is done by the implementation of each type of computation (e.g. load-flow, security-analysis, etc.).

If this module is not set, the `com.powsybl.computation.local.LocalComputationManager` implementation is used. Read the *computation-local* page to learn how to configure the `LocalComputationManager`.

Required properties

short-time-execution-computation-manager-factory The `short-time-execution-computation-manager-factory` property is a required property that defines the name of the `com.powsybl.computation.ComputationManagerFactory` implementation to use for short-time computations.

Optional properties

long-time-execution-computation-manager-factory The `long-time-execution-computation-manager-factory` property is an optional property that defines the name of the `com.powsybl.computation.ComputationManagerFactory` implementation to use for long-time computations. If not defined, this property returns the same value as the `short-time-execution-manager-factory`.

Examples

YAML configuration:

```
default-computation-manager:
    long-time-execution-computation-manager-factory: com.powsybl.computation.local.
↪LocalComputationManagerFactory
    short-time-execution-computation-manager-factory: com.powsybl.computation.local.
↪LocalComputationManagerFactory
```

XML configuration:

```
<default-computation-manager>
    <long-time-execution-computation-manager-factory>com.powsybl.computation.local.
```

(continues on next page)

(continued from previous page)

```
↪LocalComputationManagerFactory</long-time-execution-computation-manager-factory>
    <short-time-execution-computation-manager-factory>com.powsybl.computation.local.
↪LocalComputationManagerFactory</short-time-execution-computation-manager-factory>
</default-computation-manager>
```

external-security-analysis-config

The `external-security-analysis-config` module is used in the `com.powsybl.security.distributed.ExternalSecurityAnalysis` class, an implementation of the `com.powsybl.security.SecurityAnalysis` interface, that submits the execution of a *security-analysis* command to the ComputationManager, when it's launched in external mode.

Required properties

itools-command The `itools-command` property is a required property that defines the iTools command to run. It throws a `ConfigurationException` if this property is not set.

Optional property

debug The `debug` property is an optional property that defines whether the `security-analysis` should run in debug mode or not. The default value of this property is `false`.

Examples

YAML configuration:

```
external-security-analysis-config:
    debug: false
    itools-command: itools
```

XML configuration:

```
<external-security-analysis-config>
    <debug>false</debug>
    <itools-command>itools</itools-command>
</external-security-analysis-config>
```

geo-json-importer-post-processor

The `geo-json-importer-post-processor` module is used to configure the paths of the files containing the geographical data.

The names of the properties are the two types of elements for which data are expected. Each value must be the complete path to the corresponding file.

- substations
- lines

Example

YAML configuration:

```
geo-json-importer-post-processor:
  substations: /path/to/substations.geojson
  lines: /path/to/lines.geojson
```

XML configuration

```
<geo-json-importer-post-processor>
    <substations>/path/to/substations.geojson</substations>
    <lines>/path/to/lines.geojson</lines>
</geo-json-importer-post-processor>
```

groovy-dsl-contingencies

The `groovy-dsl-contingencies` module is used by the `com.powsybl.action.dsl.GroovyDslContingenciesProviderFactory`, which is an implementation of the `com.powsybl.contingency.ContingenciesProviderFactory` used by the *security-analysis* command.

Required properties

dsl-file The `dsl-file` property is a required property that defines the path of the groovy script defining the list of contingencies to simulate. Read the *documentation* page to learn more about the syntax of the `GroovyDslContingenciesProvider`.

Examples

YAML configuration:

```
groovy-dsl-contingencies:
    dsl-file: /home/user/contingencies.groovy
```

XML configuration:

```
<groovy-dsl-contingencies>
    <dsl-file>/home/user/contingencies.groovy</dsl-file>
</groovy-dsl-contingencies>
```

import-export-parameters-default-value

The `import-export-parameters-default-value` module is an optional module used to configure the network importers and exporters.

The parameters are different from a format importer/exporter to another, please refer to the documentation of each *supported format* to learn more about their specific configuration.

Examples

In this example we configure:

- the IIDM importer to throw an exception on trying to import an unknown or not deserializable extension
- the IIDM exporter to export to IIDM in its version 1.12

YAML configuration:

```
import-export-parameters-default-value:
    iidm.import.xml.throw-exception-if-extension-not-found: true
    iidm.export.xml.version: "1.12"
```

XML configuration:

```
<import-export-parameters-default-value>
    <iidm.import.xml.throw-exception-if-extension-not-found>true</iidm.import.xml.throw-
↪exception-if-extension-not-found>
    <iidm.export.xml.version>1.12</iidm.export.xml.version>
</import-export-parameters-default-value>
```

limit-violation-default-filter

The `limit-violation-default-filter` module is used by the *security-analysis* and the *action-simulator* commands to filter the violations displayed.

Optional properties

countries The `countries` property is an optional property that defines a list of ISO-3166 country codes used for violations filtering. A violation is displayed only if at least one of its sides has its substation's country in the list. If this property is not set, there is no filtering based on the countries.

minBaseVoltage The `minBaseVoltage` property is an optional property that defines a threshold value for the nominal voltage of the voltage levels. The default value of this property is `0`.

violationTypes The `violationTypes` property is an optional property that defines a list of `com.powsybl.contingency.violations.LimitViolationType` used for violations filtering. a violation is displayed if its type is in the list. The available `LimitViolationType` values are:

- CURRENT
- LOW_VOLTAGE
- HIGH_VOLTAGE
- LOW_SHORT_CIRCUIT_CURRENT
- HIGH_SHORT_CIRCUIT_CURRENT
- OTHER

Examples

YAML configuration:

```
limit-violation-default-filter:
    countries:
        - FR
        - BE
    minBaseVoltage: 225
    violationTypes:
        - CURRENT
        - LOW_VOLTAGE
        - HIGH_VOLTAGE
```

XML configuration:

```
<limit-violation-default-filter>
    <countries>FR,BE</countries>
    <minBaseVoltage>225</minBaseVoltage>
    <violationTypes>CURRENT,LOW_VOLTAGE,HIGH_VOLTAGE</violationTypes>
</limit-violation-default-filter>
```

loadflow-results-completion-parameters

The `loadflow-results-completion-parameters` module is used by the *load flow validation* and the *load flow post processor* features.

Optional properties

apply-reactance-correction The `apply-reactance-correction` property is an optional property that defines whether the too small reactance values have to be fixed to `epsilon-x` value or not. To solve numeric issues with very small reactance values, it's necessary to set the too small values to a minimal value. The default value of this property is `false`.

epsilon-x The `epsilon-x` property is an optional property that defines the reactance value used for fixing. The default value of this property is `0.1`.

Examples

YAML configuration:

```
loadflow-results-completion-parameters:
    apply-reactance-correction: true
    epsilon-x: 0.1
```

XML configuration:

```
<loadflow-results-completion-parameters>
    <apply-reactance-correction>true</apply-reactance-correction>
    <epsilon-x>0.1</epsilon-x>
</loadflow-results-completion-parameters>
```

load-flow-action-simulator

The `load-flow-action-simulator` module is used by the *action-simulator* tool if it's configured to use the `LoadFlowActionSimulator` implementation.

Properties

copy-strategy Use the `copy-strategy` to define how the action-simulator will store and restore network state internally. This choice can greatly impact performances. Possible values are:

- `STATE`: will only save and restore state data. Optimizes performances but will not behave correctly if some actions modify the structure of the network.
- `DEEP`: will save and restore all network data. Decreases performance but allows using any type of action.

ignore-pre-contingency-violations Set the `ignore-pre-contingency-violations` to `true` to ignore the pre-contingency violations and continue the simulation even if there are still violations after the pre-contingency simulation.

load-flow-name The `load-flow-name` property is an optional property that defines the implementation name to use for running the load flow. If this property is not set, the default load flow implementation is used. See *Loadflow Configuration* to configure the default load flow.

max-iterations Use the `max-iterations` parameter to limit the number of iterations needed to solve the violations.

Examples

YAML configuration:

```
load-flow-action-simulator:
    copy-strategy: STATE
    debug: false
    ignore-pre-contingency-violations: false
    load-flow-name: Mock
    max-iterations: 10
```

XML configuration:

```
<load-flow-action-simulator>
    <copy-strategy>STATE</copy-strategy>
    <debug>false</debug>
    <ignore-pre-contingency-violations>false</ignore-pre-contingency-violations>
    <load-flow-name>Mock</load-flow-name>
    <max-iterations>10</max-iterations>
</load-flow-action-simulator>
```

loadflow-validation

The `loadflow-validation` module is used by the *loadflow-validation* command and the *load flow validation* feature. It defines the parameters used during the validation of load flow results.

Optional properties

apply-reactance-correction The `apply-reactance-correction` property is an optional property that defines whether the too small reactance values have to be fixed to `epsilon-x` value or not. To solve numeric issues with very small reactance values, it's necessary to set the too small values to a minimal value. The default value of this property is `false`.

check-main-component-only The `check-main-component-only` property is an optional property that defines whether the validation checks are done only on the equipments in the main connected component or in all components. The default value of this property is `true`.

compare-results Set the `compare-results` property to true to compare the results of 2 validations, i.e. print output files with data of both ones. The default value of this property is `false`.

epsilon-x The `epsilon-x` property is an optional property that defines the value used to correct the reactance in flows validation. The default value of this property is `0.1`.

load-flow-name The `load-flow-name` property is an optional property that defines the implementation name to use for running the load flow. If this property is not set, the default load flow implementation is used. See *Loadflow Configuration* to configure the default load flow.

Note: In previous PowSyBl releases (before 3.0.0), this was configured by the `load-flow-factory` property with the full classname of the implementation.

no-requirement-if-reactive-bound-inversion The `no-requirement-if-reactive-bound-inversion` property is an optional property that defines whether the validation checks fail if there is a reactive bounds inversion (maxQ < minQ) or not. The default value of this property is `false`.

no-requirement-if-setpoint-outside-power-bounds The `no-requirement-if-setpoint-outside-power-bounds` property is an optional property that defines whether the validation checks fail if there is a setpoint outside the active power bounds (targetP < minP or targetP > maxP) or not. The default value of this property is `false`.

ok-missing-values The `ok-missing-values` property is an optional property that defines whether the validation checks fail if some parameters of connected components have `NaN` values or not. The default value of this property is `false`.

output-writer The `output-writer` property is an optional property that defines the output format. Currently, `CSV` and `CSV_MULTILINE` are supported. The default value of this property is set to `CSV_MULTILINE`.

If this property is set to `CSV`, in the output files a line contains all values of validated equipment. If the property is set to `CSV_MULTILINE`, in the output files the equipment values are split in multiple lines, one value for each line, see examples below:

Example of output in CSV format:

```
id;p;q;v;nominalV;reactivePowerSetpoint;voltageSetpoint;connected;regulationMode;bMin;
↪bMax;mainComponent;validation
CSPCH.TC1;-0,00000;93,6368;238,307;225,000;0,00000;238,307;true;VOLTAGE;-0,00197531;0,
↪00493827;true;success
CSPDO.TC1;-0,00000;0,00000;240,679;225,000;0,00000;240,713;true;VOLTAGE;-0,00493827;0,
↪00493827;true;success
...
```

Example of output in CSV_MULTILINE format:

```
id;characteristic;value
CSPCH.TC1;p;-0,00000
CSPCH.TC1;q;93,6368
CSPCH.TC1;v;238,307
...
```

table-formatter-factory: The `table-formatter-factory` property is an optional property that defines the `com.powsybl.commons.io.table.TableFormatterFactory` implementation to use for writing the output files. If this property is not set, the `com.powsybl.commons.io.table.CsvTableFormatterFactory` implementation is used.

The available implementation of the `TableFormatterFactory` are:

- `com.powsybl.commons.io.table.CsvTableFormatterFactory`: to create a CSV file
- `com.powsybl.commons.io.table.AsciiTableFormatterFactory`: to render table in ASCII

The table formatter can be configured by the *table-formatter* module.

threshold: The `threshold` property is an optional property that defines the margin used for values comparison. The default value of this property is `0`.

verbose: The `verbose` property is an optional property that defines whether the *loadflow-validation* command runs in verbose or quiet mode.

If this property is set to `true`, the output files contain all the data of the validated equipment; otherwise they contain only the main data of the validated equipment.

Examples

YAML configuration:

```
loadflow-validation:
    threshold: 0.1
    verbose: false
    load-flow-name: Mock
    table-formatter-factory: com.powsybl.commons.io.table.CsvTableFormatterFactory
    epsilon-x: 0.1
    apply-reactance-correction: false
    output-writer: CSV_MULTILINE
    ok-missing-values: false
    no-requirement-if-reactive-bound-inversion: false
    compare-results: false
    check-main-component-only: true
    no-requirement-if-setpoint-outside-power-bounds: false
```

XML configuration:

```
<loadflow-validation>
    <threshold>0.1</threshold>
    <verbose>false</verbose>
    <load-flow-name>Mock</load-flow-name>
    <table-formatter-factory>com.powsybl.commons.io.table.CsvTableFormatterFactory</
↪table-formatter-factory>
    <epsilon-x>0.1</epsilon-x>
    <apply-reactance-correction>false</apply-reactance-correction>
    <output-writer>CSV_MULTILINE</output-writer>
    <ok-missing-values>false</ok-missing-values>
    <no-requirement-if-reactive-bound-inversion>false</no-requirement-if-reactive-bound-
↪inversion>
    <compare-results>false</compare-results>
    <check-main-component-only>true</check-main-component-only>
    <no-requirement-if-setpoint-outside-power-bounds>false</no-requirement-if-setpoint-
↪outside-power-bounds>
</loadflow-validation>
```

network

The `network` module is used to configure the network default implementation name.The network implementation is the set of classes implementing all the network elements, such as VoltageLevel or Generator.

The implementation named "Default" is the classic PowSyBl in-memory implementation.

Required properties

default-impl-name
The `default-impl-name` property is a required property that specifies the name of the default network implementation.

Examples

YAML configuration:

```
network:
    default-impl-name: Default
```

XML configuration:

```
<network>
    <default-impl-name>Default</default-impl-name>
</network>
```

table-formatter

The `table-formatter` module is used to configure the rendering of tables displayed in the console. It is also used to export data in CSV files.

Optional properties

invalidString
The `invalidString` property is an optional property that defines the replacement string to display when a value is absent. The default value of this property is `inv`.

language
The `language` property is an optional property that defines the language code of the locale to use. The default value of this property is the language code (2-characters code) of the system default locale.

printHeader
The `printHeader` property is an optional property that defines whether the headers of the columns are displayed or not. The default value of this property is `true`.

printTitle
The `printTitle` property is an optional property that defines whether the title of the table is displayed or not. The default value of this property is `true`.

separator
The `separator` property is an optional property that defines the column separator used in CSV files. The default value of this property is `;`.

Examples

YAML configuration:

```
table-formatter:
    invalidString: inv
    language: FR
    printHeader: true
    printTitle: true
    separator: ;
```

XML configuration:

```
<table-formatter>
    <invalidString>inv</invalidString>
    <language>FR</language>
```

(continues on next page)

(continued from previous page)

```
    <printHeader>true</printHeader>
    <printTitle>true</printTitle>
    <separator>;</separator>
</table-formatter>
```

Modules list

- *componentDefaultConfig*
- *computation-local*
- *default-computation-manager*
- *dynamic-simulation*
- *dynamic-simulation-default-parameters*
- *dynawo*
- *dynawo-simulation-default-parameters*
- *external-security-analysis-config*
- *geo-json-importer-post-processor*
- *groovy-dsl-contingencies*
- *groovy-post-processor*
- *import-export-parameters-default-value*
- *load-flow*
- *load-flow-default-parameters*
- *network*
- *open-loadflow-default-parameters* (Open Load Flow implementation of the loadflow)
- *dynaflow-default-parameters* (Dynaflow implementation of the loadflow)
- *load-flow-based-phase-shifter-optimizer*
- *load-flow-action-simulator*
- *loadflow-results-completion-parameters*
- *loadflow-validation*
- *security-analysis*
- *limit-violation-default-filter*
- *dynamic-security-analysis*
- *dynamic-security-analysis-default-parameters*
- *table-formatter*

The configuration mechanism supports YAML and XML file formats. The framework looks inside all the folders specified to the *powsybl_config_dirs* property in the *itools.conf* file for configuration files. The framework uses the *powsybl_config_name* property as the basename of the configuration files. It looks for a YAML file first, then for an XML file. The XML file will be used only if the YAML configuration file has not been found.

The configuration can also be configured using the system's environment variables. These variables should respect the following format: `MODULE_NAME__PROPERTY_NAME`. Note that these variables will overload the XML/YAML configuration files.

The default configuration folder and the configuration file name can be configured in the `POWSYBL_HOME/etc/itools.conf`.

6.1.2 Properties

The configuration file contains a list of modules that can be required or optional. Each module contains one or several properties. These properties can also be required or optional. Names in configuration files are case-sensitive.

Example

YAML configuration

```yaml
module1:
    property1a: value1
    property1b: value2

module2:
    property2a: value3
    property2b: value4
    property2c: value5
```

XML configuration

```xml
<config>
    <module1>
        <property1a>value1</property1a>
        <property1b>value2</property1b>
    </module1>
    <module1>
        <property2a>value3</property2a>
        <property2b>value4</property2b>
        <property2c>value5</property2c>
    </module1>
</config>
```

System's environment variables

Configuration properties can also be overridden using system's environment variables. The module and the property are separated using two underscores. The table below gives examples on the way to declare environment variables for PowSyBl:

Environment variable	Module name	Property name
MODULE1__PROPERTY1=1	module1	property1
LOWER_HYPHEN__PROPERTY2=2	lower-hyphen	property2
LOWER_CAMEL__PROPERTY3=3	lowerCamel	property3
UPPER_CAMEL__PROPERTY4=4	UpperCamel	property4
SNAKE_CASE__PROPERTY5=5	snake_case	property5

6.1.3 Modules

The module list is available *here*.

6.2 iTools

6.2.1 Commands

iTools action-simulator

The `action-simulator` command loads a grid file and run a *security analysis with remedial actions*. This tool is used to create or validate a strategy in order to solve violations.

Usage

```
$> itools action-simulator --help
usage: itools [OPTIONS] action-simulator [--apply-if-solved-violations]
       --case-file <FILE> [--contingencies <CONTINGENCY1,CONTINGENCY2,...>]
       --dsl-file <FILE> [--export-after-each-round] [--help] [-I
       <property=value>] [--import-parameters <IMPORT_PARAMETERS>]
       [--output-case-folder <CASEFOLDER>] [--output-case-format <CASEFORMAT>]
       [--output-compression-format <COMPRESSION_FORMAT>] [--output-file <FILE>]
       [--output-format <FORMAT>] [--verbose]

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name

Available arguments are:
    --apply-if-solved-violations                        apply the first tested
                                                        action which solves all
                                                        violations
    --case-file <FILE>                                  the case path
    --contingencies <CONTINGENCY1,CONTINGENCY2,...>     contingencies to test
    --dsl-file <FILE>                                   the Groovy DSL path
    --export-after-each-round                           export case after each
                                                        round
    --help                                              display the help and quit
-I <property=value>                                     use value for given
                                                        importer parameter
    --import-parameters <IMPORT_PARAMETERS>             the importer configuation
                                                        file
    --output-case-folder <CASEFOLDER>                   output case folder path
    --output-case-format <CASEFORMAT>                   output case format [CSV,
                                                        AMPL, XIIDM]
    --output-compression-format <COMPRESSION_FORMAT>    output compression format
                                                        [BZIP2, GZIP, XZ, ZIP,
                                                        ZSTD]
    --output-file <FILE>                                the output file path
    --output-format <FORMAT>                            the output file format
                                                        [JSON]
    --verbose                                           verbose mode
```

Required arguments

`--case-file` This option defines the path of the case file on which the power flow simulation is run. The *supported formats* depend on the execution class path.

`--dsl-file` This option defines the path of the strategy to evaluate. This is a groovy script that respects the *action DSL* syntax.

Optional parameters

`--apply-if-solved-violations` TODO

`--contingencies` This option defines the list of contingencies to simulate. If this parameter is omitted, all the contingencies defined in the DSL file are simulated.

`--export-after-each-round` If this option is passed, a case file is exported after each round of the simulation. Otherwise, a single case file is exported at the end of the simulation (once there are no more violations or matching rules).

`--import-parameters` This option defines the path of the importer's configuration file. It's possible to overload one or many parameters using the `-I property=value` syntax. The list of supported properties depends on the *input format*.

`--output-case-folder` This option defines the path to the folder in which the case files will be exported.

`--output-case-format` This option defines the format of the output case files. The list of *supported formats* are listed between brackets in the command help.

`--output-compression-format` This option defines the compression format of the case files. The list of supported formats is mentioned between brackets in the command help.

`--output-file` This option defines the path of the result file. If this option is omitted, the results are displayed in the console.

`--output-format` This option defines the format of the result file. This option is required if `--output-file` is used. The supported formats are listed between brackets in the command help.

`--verbose` This option enables the verbose mode, to display more information during the simulation.

Simulators

Currently, the only simulator which is supported is the load-flow-based simulator.

Parameters

TODO

Results

TODO

Examples

This example shows a small *action DSL* script:

```
contingency('HV_line_1') {
    equipments 'NHV1_NHV2_1'
}
```

(continues on next page)

(continued from previous page)

```
contingency('HV_line_2') {
    equipments 'NHV1_NHV2_2'
}

rule('apply_shedding_for_line_1') {
    description 'Test load sheddings when line 1 is overloaded'
    life 8
    when isOverloaded(['NHV1_NHV2_1'])
    apply 'load_shed_100'
}

rule('apply_shedding_for_line_2') {
    description 'Test load sheddings when line 2 is overloaded'
    life 8
    when isOverloaded(['NHV1_NHV2_2'])
    apply 'load_shed_100'
}

action('load_shed_100') {
    description 'load shedding 100 MW'
    tasks {
        script {
            load('LOAD').p0 -= 100
        }
    }
}
```

The following example shows the results of the simulation of the previous script:

```
$> itools action-simulator --case-file $HOME/eurostag-tutorial.xiidm --dsl-file $HOME/
↪actions.groovy
Loading network '$HOME/eurostag-tutorial.xiidm'
Loading DSL 'file:$HOME/actions.groovy'
Using 'loadflow' rules engine
Starting pre-contingency analysis
    Round 0
        No more violation
Starting post-contingency 'HV_line_1' analysis
    Round 0
        Violations:
+---------------+-------+---------+--------------+----------------+-----------------+----
↪-------+----------+------------------+----------------+
| Equipment (2) | End   | Country | Base voltage | Violation type | Violation name  |␣
↪Value     | Limit    | abs(value-limit) | Loading rate % |
+---------------+-------+---------+--------------+----------------+-----------------+----
↪-------+----------+------------------+----------------+
| NHV1_NHV2_2   | VLHV1 | FR      |          380 | CURRENT        | Permanent limit |␣
↪1008.9287 | 500.0000 |         508.9287 |         201.79 |
| NHV1_NHV2_2   | VLHV2 | FR      |          380 | CURRENT        | Permanent limit |␣
↪1047.8258 | 500.0000 |         547.8258 |         209.57 |
+---------------+-------+---------+--------------+----------------+-----------------+----
↪-------+----------+------------------+----------------+
```

(continues on next page)

(continued from previous page)

```
        Rule 'apply_shedding_for_line_2' evaluated to TRUE
        Applying action 'load_shed_100'
    Round 1
        Violations:
+---------------+-------+---------+--------------+----------------+-----------------+----
↪------+----------+------------------+----------------+
| Equipment (2) | End   | Country | Base voltage | Violation type | Violation name  |␣
↪Value    | Limit    | abs(value-limit) | Loading rate % |
+---------------+-------+---------+--------------+----------------+-----------------+----
↪------+----------+------------------+----------------+
| NHV1_NHV2_2   | VLHV1 | FR      |          380 | CURRENT        | Permanent limit |␣
↪831.3489 | 500.0000 |         331.3489 |         166.27 |
| NHV1_NHV2_2   | VLHV2 | FR      |          380 | CURRENT        | Permanent limit |␣
↪871.7283 | 500.0000 |         371.7283 |         174.35 |
+---------------+-------+---------+--------------+----------------+-----------------+----
↪------+----------+------------------+----------------+
        Rule 'apply_shedding_for_line_2' evaluated to TRUE
        Applying action 'load_shed_100'
    Round 2
        Violations:
+---------------+-------+---------+--------------+----------------+-----------------+----
↪------+----------+------------------+----------------+
| Equipment (2) | End   | Country | Base voltage | Violation type | Violation name  |␣
↪Value    | Limit    | abs(value-limit) | Loading rate % |
+---------------+-------+---------+--------------+----------------+-----------------+----
↪------+----------+------------------+----------------+
| NHV1_NHV2_2   | VLHV1 | FR      |          380 | CURRENT        | Permanent limit |␣
↪667.6796 | 500.0000 |         167.6796 |         133.54 |
| NHV1_NHV2_2   | VLHV2 | FR      |          380 | CURRENT        | Permanent limit |␣
↪711.4252 | 500.0000 |         211.4252 |         142.29 |
+---------------+-------+---------+--------------+----------------+-----------------+----
↪------+----------+------------------+----------------+
        Rule 'apply_shedding_for_line_2' evaluated to TRUE
        Applying action 'load_shed_100'
    Round 3
        Violations:
+---------------+-------+---------+--------------+----------------+-----------------+----
↪------+----------+------------------+----------------+
| Equipment (2) | End   | Country | Base voltage | Violation type | Violation name  |␣
↪Value    | Limit    | abs(value-limit) | Loading rate % |
+---------------+-------+---------+--------------+----------------+-----------------+----
↪------+----------+------------------+----------------+
| NHV1_NHV2_2   | VLHV1 | FR      |          380 | CURRENT        | Permanent limit |␣
↪516.0706 | 500.0000 |          16.0706 |         103.21 |
| NHV1_NHV2_2   | VLHV2 | FR      |          380 | CURRENT        | Permanent limit |␣
↪566.1081 | 500.0000 |          66.1081 |         113.22 |
+---------------+-------+---------+--------------+----------------+-----------------+----
↪------+----------+------------------+----------------+
        Max number of iterations reached
Starting post-contingency 'HV_line_2' analysis
    Round 0
        Violations:
```

(continues on next page)

(continued from previous page)

```
+---------------+-------+---------+--------------+----------------+-----------------+----
↪-------+-----------+------------------+----------------+
| Equipment (2) | End   | Country | Base voltage | Violation type | Violation name  |␣
↪Value     | Limit     | abs(value-limit) | Loading rate % |
+---------------+-------+---------+--------------+----------------+-----------------+----
↪-------+-----------+------------------+----------------+
| NHV1_NHV2_1   | VLHV1 | FR      |          380 | CURRENT        | Permanent limit |␣
↪1008.9287 | 1000.0000 |           8.9287 |         100.89 |
| NHV1_NHV2_1   | VLHV2 | FR      |          380 | CURRENT        | Permanent limit |␣
↪1047.8258 | 1000.0000 |          47.8258 |         104.78 |
+---------------+-------+---------+--------------+----------------+-----------------+----
↪-------+-----------+------------------+----------------+
        Rule 'apply_shedding_for_line_1' evaluated to TRUE
        Applying action 'load_shed_100'
    Round 1
        No more violation
Final result
Pre-contingency violations:
+--------+---------------+-----+---------+--------------+----------------+---------------
↪-+-------+-------+------------------+----------------+
| Action | Equipment (0) | End | Country | Base voltage | Violation type | Violation␣
↪name | Value | Limit | abs(value-limit) | Loading rate % |
+--------+---------------+-----+---------+--------------+----------------+---------------
↪-+-------+-------+------------------+----------------+
Post-contingency limit violations:
+-------------+----------+---------------+---------------+-------+---------+-------------
↪-+----------------+-----------------+----------+----------+------------------+---------
↪-------+
| Contingency | Status   | Action        | Equipment (2) | End   | Country | Base␣
↪voltage | Violation type | Violation name  | Value    | Limit    | abs(value-limit) |␣
↪Loading rate % |
+-------------+----------+---------------+---------------+-------+---------+-------------
↪-+----------------+-----------------+----------+----------+------------------+---------
↪-------+
| HV_line_1   | converge |               | Equipment (2) |       |         |            ␣
↪ |                |                 |          |          |                  |        ␣
↪       |
|             |          | load_shed_100 |               |       |         |            ␣
↪ |                |                 |          |          |                  |        ␣
↪       |
|             |          | load_shed_100 |               |       |         |            ␣
↪ |                |                 |          |          |                  |        ␣
↪       |
|             |          | load_shed_100 |               |       |         |            ␣
↪ |                |                 |          |          |                  |        ␣
↪       |
|             |          |               | NHV1_NHV2_2   | VLHV1 | FR      |          ␣
↪380 | CURRENT        | Permanent limit | 516.0706 | 500.0000 |          16.0706 |      ␣
↪   103.21 |
|             |          |               | NHV1_NHV2_2   | VLHV2 | FR      |          ␣
↪380 | CURRENT        | Permanent limit | 566.1081 | 500.0000 |          66.1081 |      ␣
↪   113.22 |
```

(continues on next page)

(continued from previous page)

```
+-------------+----------+---------------+---------------+-------+---------+-------------
↪-+----------------+-----------------+----------+----------+------------------+---------
↪-------+
```

iTools cim-anonymizer

The `cim-anonymizer` command is used to anonymize one or several CIM files. All the identifiers are replaced by new generated identifiers. The mapping between old and new identifiers is exported in a CSV file.

Usage

```
$> itools cim-anonymizer --help
usage: itools [OPTIONS] cim-anonymizer --cim-path <PATH> [--help] --mapping-file
       <FILE> --output-dir <DIR> [--skip-external-refs]

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name

Available arguments are:
    --cim-path <PATH>       CIM zip file or directory
    --help                  display the help and quit
    --mapping-file <FILE>   File to store the ID mapping
    --output-dir <DIR>      Directory to write anonymized zip files
    --skip-external-refs    Do not anonymize external references
```

Required arguments

`--cim-path` This option defines the CIM file (zip) or a directory where to look for CIM files.

`--mapping-file` This option defines the CSV file where the mapping between original and new identifiers is exported.

`--output-dir` This option defines the path of the directory where to write the anonymized CIM files. If the output directory doesn't exist, an exception is thrown.

Optional arguments

`--skip-external-refs` This option defines if the XML external references should also be anonymized or not. The default value is `false`, meaning that the external references are also anonymized.

iTools compare security analysis results

The `compare-security-analysis-results` command is used to compare *security-analysis* results, stored in JSON.

The outcome of the comparison is `success` if the results are equivalent, `fail` otherwise.

This tool compares for the pre-contingency state and for all the post-contingency states of the two results:

- the outcome (convergence/divergence) of the load flow computations
- the values of the constraints violations

Two security analysis results are considered equivalent if all the following conditions are satisfied:

- for all the pre-contingency and post-contingency states, the corresponding (i.e., related to the same state) outcome of the load flow computation is the same

- for all the constraint violations, the difference of value of a corresponding (i.e., related to the same contingency and equipment) violation is less than a predefined threshold
- if a constraint violation is contained in just one result, the violation is less than a predefined threshold
- if a contingency is contained in just one result, all the post-contingency violations are less than a predefined threshold

The comparison process can optionally output in a CSV file all the compared values (pre- and post-contingency load flow computation outcomes, and related constraints violations), with a corresponding comparison result (`equivalent`,`different`). See example below.

```
Contingency;StatusResult1;StatusResult2;Equipment;End;ViolationType;ViolationNameResult1;
↪ValueResult1;LimitResult1;ViolationNameResult2;ValueResult2;LimitResult2;
↪ActionsResult1;ActionsResult2;Comparison
;converge;converge;;;;;;;;;;;;equivalent
;;;NHV1_NHV2_1;ONE;CURRENT;;1100,00;950,000;;1100,09;950,000;;;equivalent
contingency1;converge;converge;;;;;;;;;;;;equivalent
contingency1;;;NHV1_NHV2_1;ONE;CURRENT;;1100,00;950,000;;1100,09;950,000;;;equivalent
contingency1;;;NHV1_NHV2_1;TWO;CURRENT;;1100,00;950,000;;1101,00;950,000;;;different
contingency2;converge;converge;;;;;;;;;;;;equivalent
contingency2;;;NHV1_NHV2_1;ONE;CURRENT;;1100,00;950,000;;1100,09;950,000;;;equivalent
contingency2;;;NHV1_NHV2_2;ONE;CURRENT;;1100,00;950,000;;1100,09;950,000;;;equivalent
```

Usage

```
$> itools compare-security-analysis-results --help
usage: itools [OPTIONS] compare-security-analysis-results [--help] --output-file
       <FILE> --result1-file <FILE> --result2-file <FILE> [--threshold
       <THRESHOLD>]

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name

Available arguments are:
    --help                    display the help and quit
    --output-file <FILE>      output file path, where the comparison results
                              will be stored
    --result1-file <FILE>     security analysis result 1 file path
    --result2-file <FILE>     security analysis result 2 file path
    --threshold <THRESHOLD>   threshold used for results comparison, default is 0.0
```

Required parameters

`--output-file` This parameter defines the path of the output file, where the comparison results will be stored.

`--result1-file` This parameter defines the path of the JSON file containing the first security analysis result.

`--result2-file` This parameter defines the path of the JSON file containing the second security analysis result.

Optional parameters

threshold

Use the `--threshold` parameter to specify the threshold used for comparing values of the two security analysis results. Default value is `0`.

Examples

This example shows how to compare two security analysis results and to store the comparison results in the `$HOME/comparison_results.csv` file:

```
$> itools compare-security-analysis-results --result1-file $HOME/result1.json --result2-
↪file $HOME/result2.json --output-file $HOME/comparison_results.csv
Comparison result: fail
```

This example shows how to specify the threshold to be used in the comparison:

```
$> itools compare-security-analysis-results --result1-file $HOME/result1.json --result2-
↪file $HOME/result2.json --output-file $HOME/comparison_results.csv  --threshold 0.1
Comparison result: success
```

iTools convert-network

The `convert-network` command is used to convert a grid file from a format to another. The input format is automatically detected, whereas the output format must be specified.

`Importer` and `Exporter` are interfaces that you can implement to create your own file format. The tutorials show you how to proceed.

Usage

```
$> itools convert-network --help
usage: itools [OPTIONS] convert-network [-E <property=value>]
       [--export-parameters <EXPORT_PARAMETERS>] [--help] [-I <property=value>]
       [--import-parameters <IMPORT_PARAMETERS>] --input-file <INPUT_FILE>
       --output-file <OUTPUT_FILE> --output-format <OUTPUT_FORMAT>

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name

Available arguments are:
 -E <property=value>                          use value for given exporter
                                              parameter
    --export-parameters <EXPORT_PARAMETERS>   the exporter configuration file
    --help                                    display the help and quit
 -I <property=value>                          use value for given importer
                                              parameter
    --import-parameters <IMPORT_PARAMETERS>   the importer configuation file
    --input-file <INPUT_FILE>                 the input file
    --output-file <OUTPUT_FILE>               the output file
    --output-format <OUTPUT_FORMAT>           the output file format

Where OUTPUT_FORMAT is one of [CGMES, AMPL, UCTE, XIIDM]
```

Required arguments

`--input-file` This option defines the path of the input file. The *supported formats* depend on the execution class path.

`--output-file` This option defines the path of the output file.

`--output-format` This option defines the format of the output file. The list of *supported formats* are listed between brackets in the command help.

Optional arguments

`--export-parameters` This option defines the path of the exporter's configuration file. It's possible to overload one or many parameters using the `-E property=value` syntax. The list of supported properties depends on the *output format*.

`--import-parameters` This option defines the path of the importer's configuration file. It's possible to overload one or many parameters using the `-I property=value` syntax. The list of supported properties depends on the *input format*.

Examples

This example shows how to convert a *UCTE-DEF* file to an *XIIDM* file:

```
$> itools convert-network --input-file case-file.uct --output-format XIIDM --output-file␣
↪case-file.xiidm
```

This example shows how to pass an exporter's configuration file, and overload one of the properties:

```
$> itools convert-network --input-file case-file.uct --output-format XIIDM --output-file␣
↪case-file.xiidm --export-parameters xiidm.properties -E iidm.export.xml.indent=false
```

iTools dynamic-security-analysis

The `dynamic-security-analysis` command loads a grid file, apply dynamic models, and run a *dynamic security analysis* simulation, to detect security violations on pre- or post-contingencies states. At the end of the simulation, the results are printed or exported to a file.

Usage

```
$> itools dynamic-security-analysis --help
usage: itools [OPTIONS] dynamic-security-analysis --case-file <FILE>
       [--contingencies-file <FILE>] --dynamic-models-file <FILE>
       [--event-models-file <FILE>] [--external] [--help] [-I <property=value>]
       [--import-parameters <IMPORT_PARAMETERS>] [--limit-types <LIMIT-TYPES>]
       [--log-file <FILE>] [--monitoring-file <FILE>] [--output-file <FILE>]
       [--output-format <FORMAT>] [--parameters-file <FILE>] [--with-extensions
       <EXTENSIONS>]

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name

Available arguments are:
    --case-file <FILE>                          the case path
    --contingencies-file <FILE>                 the contingencies path
    --dynamic-models-file <FILE>                dynamic models description as a
                                                Groovy file: defines the dynamic
                                                models to be associated to chosen
                                                equipments of the network
    --event-models-file <FILE>                  dynamic event models description
                                                as a Groovy file: defines the
```

(continues on next page)

(continued from previous page)

```
                                          dynamic event models to be
                                          associated to chosen equipments of
                                          the network
    --external                            external execution
    --help                                display the help and quit
 -I <property=value>                      use value for given importer
                                          parameter
    --import-parameters <IMPORT_PARAMETERS>   the importer configuration file
    --limit-types <LIMIT-TYPES>           limit type filter (all if not set)
    --log-file <FILE>                     log output path (.zip)
    --monitoring-file <FILE>              monitoring file (.json) to get
                                          network's infos after computation
    --output-file <FILE>                  the output path
    --output-format <FORMAT>              the output format [JSON]
    --parameters-file <FILE>              loadflow parameters as JSON file
    --with-extensions <EXTENSIONS>        the extension list to enable

Allowed LIMIT-TYPES values are [ACTIVE_POWER, APPARENT_POWER, CURRENT,
LOW_VOLTAGE, HIGH_VOLTAGE, LOW_VOLTAGE_ANGLE, HIGH_VOLTAGE_ANGLE,
LOW_SHORT_CIRCUIT_CURRENT, HIGH_SHORT_CIRCUIT_CURRENT, OTHER]
Allowed EXTENSIONS values are []
```

Required arguments

`--case-file` This option defines the path of the case file on which the security analysis is run. The *supported formats* depend on the execution class path.

`--dynamic-models-file` This option defines the path of the file used to associate dynamic models to static equipments of the network or add dynamic automation systems. At the moment, only groovy scripts are supported. The *dynamic models DSL* depends on the simulator used.

Optional arguments

`--contingencies-file` This option defines the path of the contingency files. If this parameter is not set, the security violations are checked on the base state only. This file is a groovy script that respects the *contingency DSL* syntax.

`--event-models-file` This option defines the path of the configuration for the events to simulate during the simulation. At the moment, only groovy scripts are supported. The *event models DSL* depends on the simulator used.

`--external` TODO: Use this argument to run the security analysis as an external process.

`--import-parameters` This option defines the path of the importer's configuration file. It's possible to overload one or many parameters using the `-I property=value` syntax. The list of supported properties depends on the *input format*.

`--limit-types` This option allows filtering certain types of violations. It overrides the default configuration defined in the *limit-violation-default-filter* configuration module. The supported types are the following: `CURRENT`, `LOW_VOLTAGE`, `HIGH_VOLTAGE`, `LOW_SHORT_CIRCUIT_CURRENT`, `HIGH_SHORT_CIRCUIT_CURRENT` and `OTHER`.

`--log-file` TODO

`--output-file` This option defines the path of the result file. If this option is not set, the results are printed to the console.

`--output-format` This option defines the format of the output file. This option is required if the `--output-file` is set. The only supported format is `JSON`.

`--parameters-file` This option defines the path of the *parameters* file of the simulation. If this option is not used, the simulation is run with the default parameters.

`--with-extensions` This option defines the list of extensions to complete the simulation results with additional data. The available extensions are listed in the usage of the command.

Simulators

TODO

Contingencies

TODO

Parameters

TODO

Results

TODO

with-extensions

Use the `--with-extensions` parameter to activate a list of `com.powsybl.security.interceptors.SecurityAnalysisInterceptor` implementations.

See also

- *List dynamic simulation models with an iTools command*: learn how to load a list of all dynamic simulation models from the command line.

iTools dynamic-simulation

The `dynamic-simulation` command loads a grid file and run a *time domain* simulation. In the end, the results and the modified network can be exported to files.

Usage

```
usage: itools [OPTIONS] dynamic-simulation --case-file <FILE> [--output-variables-file
       <FILE>] --dynamic-models-file <FILE> [--event-models-file <FILE>]
       [--help] [-I <property=value>] [--import-parameters <IMPORT_PARAMETERS>]
       [--output-file <FILE>] [--parameters-file <FILE>]

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name

Available arguments are:
    --case-file <FILE>                          the case path
    --output-variables-file <FILE>              output variables description as
                                                Groovy file: defines a list of
```

(continues on next page)

(continued from previous page)

```
                                          variables to plot or get the final
                                          value
   --dynamic-models-file <FILE>           dynamic models description as a
                                          Groovy file: defines the dynamic
                                          models to be associated to chosen
                                          equipments of the network
   --event-models-file <FILE>             dynamic event models description
                                          as a Groovy file: defines the
                                          dynamic event models to be
                                          associated to chosen equipments of
                                          the network
   --help                                 display the help and quit
-I <property=value>                       use value for given importer
                                          parameter
   --import-parameters <IMPORT_PARAMETERS>  the importer configuation file
   --output-file <FILE>                   dynamic simulation results output
                                          path
   --parameters-file <FILE>               dynamic simulation parameters as
                                          JSON file
```

Required options

`--case-file` This option defines the path of the case file on which the simulation is run. The *supported formats* depend on the execution class path.

`--dynamic-models-file` This option defines the path of the file used to associate dynamic models to static equipments of the network or add dynamic automation systems. At the moment, only groovy scripts are supported. The *dynamic models DSL* depends on the simulator used.

Optional options

`--output-variables-file` This option defines the path of the configuration for the output variables to export at the end of the simulation. This configuration file is a groovy script that respects the *output variables DSL* syntax.

`--event-models-file` This option defines the path of the configuration for the events to simulate during the simulation. At the moment, only groovy scripts are supported. The *event models DSL* depends on the simulator used.

`--import-parameters` This option defines the path of the importer's configuration file. It's possible to overload one or many parameters using the `-I property=value` syntax. The list of supported properties depends on the *input format*.

`--output-file` This option defines the path where to export the *results* of the simulation.

`--parameters-file` This option defines the path of the *parameters* file of the simulation. If this option is not used, the simulation is run with the default parameters.

Simulators

The available power flow simulators implementations are described *here*.

Parameters

The available parameters are described *here*.

Results

The expected results are described in the *time domain documentation*

Examples

The following example shows how to run a power flow simulation, using the default configuration:

```
$> itools dynamic-simulation --case-file IEEE14.iidm --dynamic-models-file dynamicModels.
↪groovy --output-variables-file outputVariables.groovy
Loading network '/tmp/mathbagu/IEEE14.iidm'
dynamic simulation results:
+--------+
| Result |
+--------+
| true   |
+--------+
```

The following example shows how to run a time domain simulation, using a parameter file:

```
$> itools dynamic-simulation --case-file IEEE14.iidm --dynamic-models-file dynamicModels.
↪groovy --parameters-file dynawoParameters.json
dynamic simulation results:
+--------+
| Result |
+--------+
| true   |
+--------+
```

See also

- *List dynamic simulation models with an iTools command*: learn how to load a list of all dynamic simulation models from the command line.

iTools list-dynamic-simulation-models

The `list-dynamic-simulation-models` command lists all models used by the *time domain* simulation for a given provider.

Usage

```
usage: itools [OPTIONS] list-dynamic-simulation-models [--dynamic-models]
       [--event-models] [--help]

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name

Available arguments are:
    --dynamic-models   display implemented dynamic models
```

(continues on next page)

(continued from previous page)

```
    --event-models     display implemented event models
    --help             display the help and quit
```

Optional options

By default, all models are displayed, but you can use one of the following options to display specific models:

- `--dynamic-models`: allows to display dynamic models only.
- `--event-models`: allows to display event models only.

See also

- *Run a dynamic simulation through an iTools command*: learn how to perform a dynamic simulation from the command line.

iTools loadflow

The `loadflow` command loads a grid file and run a *load flow* simulation. In the end, the results and the modified network can be exported to files.

Usage

```
$> itools loadflow --help
usage: itools [OPTIONS] loadflow --case-file <FILE> [-E <property=value>]
              [--export-parameters <EXPORT_PARAMETERS>] [--help] [-I <property=value>]
              [--import-parameters <IMPORT_PARAMETERS>] [--output-case-file <FILE>]
              [--output-case-format <CASEFORMAT>] [--output-file <FILE>]
              [--output-format <FORMAT>] [--parameters-file <FILE>]

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name

Available arguments are:
    --case-file <FILE>                              the case path
-E <property=value>                                 use value for given exporter
                                                    parameter
     --export-parameters <EXPORT_PARAMETERS>        the exporter configuration file
     --help                                         display the help and quit
-I <property=value>                                 use value for given importer
                                                    parameter
     --import-parameters <IMPORT_PARAMETERS>        the importer configuation file
     --output-case-file <FILE>                      modified network base name
     --output-case-format <CASEFORMAT>              modified network output format
                                                    [CGMES, AMPL, XIIDM]
     --output-file <FILE>                           loadflow results output path
     --output-format <FORMAT>                       loadflow results output format
                                                    [CSV, JSON]
     --parameters-file <FILE>                       loadflow parameters as JSON file
```

Required options

`--case-file` This option defines the path of the case file on which the power flow simulation is run. The *supported formats* depend on the execution class path.

Optional options

`--export-parameters` This option defines the path of the exporter's configuration file. It's possible to overload one or many parameters using the `-E property=value` syntax. The list of supported properties depends on the *output format*.

`--import-parameters` This option defines the path of the importer's configuration file. It's possible to overload one or many parameters using the `-I property=value` syntax. The list of supported properties depends on the *input format*.

`--output-case-file` This option defines the path where to export the modified network.

`--output-case-format` This option defines the format of the output case file. The list of *supported formats* are listed between brackets in the command help. This option is required if `--output-case-file` is used.

`--output-file` This option defines the path where to export the *results* of the load flow.

`--output-format` This option defines the format of the output file. The supported format are `CSV` and `JSON`. This option is required if the `--output-file` is used.

`--parameters-file` This option defines the path of the *parameters* file of the simulation. If this option is not used, the simulation is run with the default parameters.

Simulators

The available power flow simulators implementations are described *here*.

Parameters

The available parameters are described *here*.

Results

The expected results are described in the *load flow documentation*

Examples

The following example shows how to run a power flow simulation, using the default configuration:

```
$> itools loadflow --case-file case.xiidm
Loading network 'case.xiidm'
loadflow results:
+--------+-------------------------------------------------------------------------------------
↪----------+
| Ok     | Metrics                                                                              ␣
↪          |
+--------+-------------------------------------------------------------------------------------
↪----------+
| true   | {nbIter=4, dureeCalcul=0.001569, cause=0, contraintes=0, statut=OK,␣
↪csprMarcheForcee=0} |
+--------+-------------------------------------------------------------------------------------
↪----------+
```

(continues on next page)

(continued from previous page)

```
Components results:
+------------------+-----------+-----------------+--------------+--------------------+
| Component number | Status    | Iteration count | Slack bus ID | Slack bus mismatch |
+------------------+-----------+-----------------+--------------+--------------------+
| 0                | CONVERGED | 8               | BUS_0        | -0,00954794        |
+------------------+-----------+-----------------+--------------+--------------------+
```

The following example shows how to run a power flow simulation, using a parameter file:

```
$> itools loadflow --case-file case.xiidm --parameters-file loadflowparameters.json
loadflow results:
+--------+-------------------------------------------------------------------------------------
↪----------+
| Ok     | Metrics                                                                              ␣
↪          |
+--------+-------------------------------------------------------------------------------------
↪----------+
| true   | {nbIter=4, dureeCalcul=0.001569, cause=0, contraintes=0, statut=OK,␣
↪csprMarcheForcee=0} |
+--------+-------------------------------------------------------------------------------------
↪----------+
Components results:
+------------------+-----------+-----------------+--------------+--------------------+
| Component number | Status    | Iteration count | Slack bus ID | Slack bus mismatch |
+------------------+-----------+-----------------+--------------+--------------------+
| 0                | CONVERGED | 8               | BUS_0        | -0,00954794        |
+------------------+-----------+-----------------+--------------+--------------------+
```

See also

TODO

iTools loadflow-validation

The `loadflow-validation` command is used to validate load-flow results of a network. The command, besides validating the results, also prints the data of the validated equipments in output files. The consistency checks performed by the load flow validation may also be applied to results obtained with an optimal power flow or to the final state of a long dynamic simulation.

Usage

```
$> itools loadflow-validation --help
usage: itools [OPTIONS] loadflow-validation --case-file <FILE>
       [--compare-case-file <FILE>] [--compare-results <COMPARISON_TYPE>]
       [--help] [-I <property=value>] [--import-parameters <IMPORT_PARAMETERS>]
       [--load-flow] --output-folder <FOLDER> [--output-format
       <VALIDATION_WRITER>] [--run-computation <COMPUTATION>] [--types
       <VALIDATION_TYPE,VALIDATION_TYPE,...>] [--verbose]

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name
```

(continues on next page)

(continued from previous page)

```
Available arguments are:
    --case-file <FILE>                                case file path
    --compare-case-file <FILE>                        path to the case file to
                                                      compare
    --compare-results <COMPARISON_TYPE>               compare results of two
                                                      validations, printing output
                                                      files with results of both
                                                      ones. Available comparisons
                                                      are [COMPUTATION (compare
                                                      the validation of a basecase
                                                      before and after the
                                                      computation), BASECASE
                                                      (compare the validation of
                                                      two basecases)]
    --help                                            display the help and quit
 -I <property=value>                                  use value for given importer
                                                      parameter
    --import-parameters <IMPORT_PARAMETERS>           the importer configuation
                                                      file
    --load-flow                                       run loadflow
    --output-folder <FOLDER>                          output folder path
    --output-format <VALIDATION_WRITER>               output format [CSV,
                                                      CSV_MULTILINE]
    --run-computation <COMPUTATION>                   run a computation on the
                                                      network before validation,
                                                      available computations are
                                                      [loadflow,
                                                      loadflowResultsCompletion]
    --types <VALIDATION_TYPE,VALIDATION_TYPE,...>     validation types [FLOWS,
                                                      GENERATORS, BUSES, SVCS,
                                                      SHUNTS, TWTS, TWTS3W] to
                                                      run, all of them if the
                                                      option if not specified
    --verbose                                         verbose output
```

Required arguments

`--case-file` Use the `--case-file` parameter to define the path of the case file.

`--output-folder` Use the `--output-folder` parameter to define the path of the folder where the output files will be stored.

Optional arguments

`--compare-case-file` Use the `--compare-case-file` parameter to define the path of the second case file, in order to compare the load flow results of two case files.

`--compare-results` Use the `--compare-results` parameter to define the type of results to compare. The available types are:

- `BASECASE`: compare results of the two base cases
- `COMPUTATION`: run a computation on the two base cases and compare results of the resulting states.

`--import-parameters` Use the `--import-parameters` parameter to specify the path of the configuration file of the importer. It is possible to overload one or many parameters using the `-I property=value` parameter. The properties depend on the input format. Refer to the documentation page of each *importer* to know their specific configuration.

`--load-flow` Use the `--load-flow` parameter to run a load-flow before the validation. This option is equivalent to `--run-computation loadflow`.

`--output-format` Use the `--output-format` parameter to specify the format of the output files. The available output formats are `CSV` or `CSV_MULTILINE`.

If this parameter is set to `CSV`, in the output files a line contains all values of validated equipment. If the parameter is set to `CSV_MULTILINE`, in the output files the values of a piece of equipment are split in multiple lines, one value for each line, see examples below:

CSV

```
id;p;q;v;nominalV;reactivePowerSetpoint;voltageSetpoint;connected;regulationMode;bMin;
↪bMax;mainComponent;validation
CSPCH.TC1;-0,00000;93,6368;238,307;225,000;0,00000;238,307;true;VOLTAGE;-0,00197531;0,
↪00493827;true;success
CSPDO.TC1;-0,00000;0,00000;240,679;225,000;0,00000;240,713;true;VOLTAGE;-0,00493827;0,
↪00493827;true;success
...
```

CSV_MULTILINE

```
id;characteristic;value
CSPCH.TC1;p;-0,00000
CSPCH.TC1;q;93,6368
CSPCH.TC1;v;238,307
...
```

`--run-computation` Use the `--run-computation` parameter to run a computation before the validation. The supported computations are:

- `loadflow`: run a load-flow
- `loadflowResultsCompletion`: compute the missing `P`, `Q`, `V` and θ values

`--types` Use the `--types` parameter to define the types of checks to run. If this parameter is not set, run all the checks. The supported types are `FLOWS`, `GENERATORS`, `BUSES`, `SVCS`, `SHUNTS`, `TWTS`.

To learn more about the different checks, read the *loadflow validation* documentation page.

Summary

The following table summarizes the possible combinations of `compare-results` and `run-computation` parameters, and the corresponding case states validated and written in the output files. Some remarks:

- State 1 is the state analyzed in the first validation
- State 2 is the state analyzed in the second validation (columns with the suffix `_postComp` in the output files)
- Case 1 is the value of `case-file` parameter
- Case 2 is the value of `compare-case-file` parameter
- some combinations are not available, e.g., if you use the `compare-results` parameter, with the `COMPUTATION` value, you have to use the `run-computation` (or `load-flow`) parameter.

Number	compare-results	run-computation	State 1	State 2 (_postComp)
1	absent	absent	Case 1 after import	None
2	absent	`loadflow/loadflowResultsC`	Case 1 after import and computation	None
3	`BASECASE`	absent	Case 1 after import	Case 2 after import
4	`BASECASE`	`loadflow/loadflowResultsC`	Case 1 after import and computation	Case 2 after import
5	`COMPUTATION`	`loadflow/loadflowResultsC`	Case 1 after import	Case 1 after import and computation

Parameters

To learn how to configure the `loadflow-validation` command, read the documentation of the *loadflow validation* module.

You may also configure the load flow itself to tune the load flow validation using the `--run-computation` option (check the *loadflow configuration page*).

Load flow results validation

Overall, in the PowSyBl validation, the tests are not made overly tight. In particular, leniency is preferred to tightness in case approximations are needed or when expectations are unclear (typically when the input data is inconsistent). For example, there is a switch to test only the main component because it is not clear what to expect from load flow results on small connected components.

Another important global setting available in the PowSyBl validation is the `ok-missing-values` parameter, which determines if is OK to have missing values or `NaN`. Normally, it should be set to false, but it may be useful in the cases where the power flow results are incomplete to go through the rest of the validation.

In this section, we go into more details about the checks performed by the validation feature of load-flow results available in PowSyBl.

Buses

If all values are present, or if only one value is missing, the result is considered to be consistent. Note that if the result contains only the voltages (phase and angle), the PowSyBl validation provides a load-flow results completion feature. It can be used to compute the flows from the voltages to ensure the result consistency, with the run-computation option of the PowSyBl validation.

Branches

The result on the branch is considered consistent if:

$$\max(\left|P_1^{calc} - P_1\right|, \left|Q_1^{calc} - Q_1\right|, \left|P_2^{calc} - P_2\right|, \left|Q_2^{calc} - Q_2\right|) \leq \epsilon$$

For a branch that is disconnected on one end (for example, end 2), then $P_2 = Q_2 = 0$. As a result, it is possible to recompute (V_2, θ_2) which are usually not returned by power flows and which are not stored in node-breaker *network* format. The quality checks are done when this is done.

In case of missing results (usually the powers P_1, Q_1, P_2, Q_2 which are not mandatory), the PowSyBl validation will consider the results as inconsistent, unless `ok-missing-values` was set to `true` by the user on purpose to make the consistency check more leniently.

In case the voltages are available but not the powers, the result completion feature of the PowSyBl validation can be used to recompute them using the validation equations (meaning that the branch validation tests will always be OK, so that it allows performing the bus validation tests).

Three-winding transformers

To be implemented, based on a conversion into 3 two-winding transformers.

Generators

Active power

The load-flow validation of PowSyBl checks whether the adjustment of balances has been done consistently by the power flow. The load-flow results do not include the adjustment mode used, nor the participation factors. They thus have to be inferred. If deviations are perfect, the proportion factor $\hat{K}$ estimated for the right mode will be the same for all the deviating units for which P is strictly P_{min} and P_{max}. Therefore, the inferred deviation is the one for which the standard deviation of the estimated proportion factor is the lowest.

Once the mode is determined, the new target can be computed for each unit. The following check is done:

$$\left|\max(P_{min}, \min(P_{max}, (1 + \hat{K}F(g)))targetP - P\right| < \epsilon$$

Voltage and reactive power

When the voltage regulation is disabled, the results' validity follows the condition below:

$$|targetQ - Q| < \epsilon$$

On the other hand, when the voltage regulation is enabled, depending on the generator's mode, one of the three conditions should be respected:

$$\begin{array}{rrcll} |V - targetV| \leq & \epsilon & \& & minQ & \leq Q \leq maxQ \\ V - targetV < & -\epsilon & \& & |Q - maxQ| & \leq \epsilon \\ targetV - V < & \epsilon & \& & |Q - minQ| & \leq \epsilon \end{array}$$

In the PowSyBl validation, there are a few tricks to handle special cases:

- if $minQ > maxQ$, then the values are switched to recover a meaningful interval if `noRequirementIfReactiveBoundInversion = false`
- in case of a missing value, the corresponding test is OK
- $minQ$ and $maxQ$ are function of P. If $targetP$ is outside $[minP, maxP]$, no test is done.

Loads

To be implemented, with tests similar to generators with voltage regulation.

Shunts

The two following conditions must be fulfilled in valid results:

$$|P| < \epsilon$$

$$\left|Q + \text{sections} * BV^2\right| < \epsilon$$

Static VAR Compensators

The following conditions must be fulfilled in valid results: $targetP = 0$ MW

- If the regulation mode is `OFF`, then $|targetQ - Q| < \epsilon$
- If the regulation mode is `REACTIVE_POWER`, same checks as a generator without voltage regulation
- If the regulation mode is `VOLTAGE`, same checks as a generator with voltage regulation with the following bounds: $minQ = -Bmax * V^2 and maxQ = -BminV^2$

HVDC lines

To be done.

VSC

Same checks as a generator. Besides, for stations paired by a cable: $\sum_{\text{stations}} P = \sum_{\text{stations}} Loss + Loss_{cable}$

LCC

To be done.

Transformers with a ratio tap changer

To check a steady-state has been reached, an upper bound of the deadband value is needed. Generally, the value of the deadband is not available in data models. Usual load flow solvers simply consider a continuous tap that is rounded afterward. As a result, one should compute an upper bound of the effect of the rounding. Under the usual situation where the low voltage (side one) is controlled, the maximum effect is expected if the high voltage is fixed (usually it decreases), and if the network connected to the low voltage is an antenna. If the transformer is perfect, the equations are:

- With the current tap `tap`, and if the regulated side is side `TWO`:

$$V_2(tap) = \rho_{tap} V_1$$

- With the next tap, the new voltage would be:

$$V_2(tap+1) = \rho_{tap+1} V_1 = \frac{\rho_{tap+1}}{\rho_{tap}} V_2(tap)$$

We can therefore compute approximately the voltage increments corresponding to $tap - 1$ and $tap + 1$.

- We then assume the *deadband* of the regulation to be equal to the voltage increase/decrease that can be performed with taps $tap - 1$ and $tap + 1$:

$$\text{up deadband} = -\min(V_2(tap+1) - V_2(tap), V_2(tap-1) - V_2(tap))$$
$$\text{down deadband} = \max(V_2(tap+1) - V_2(tap), V_2(tap-1) - V_2(tap))$$

Finally, we check that the voltage deviation $deviation = $ $V_2(tap) - targetV2$ stays inside the deadband.

- If $deviation < 0$, meaning that the voltage is too low, it should be checked if the deviation is smaller by increasing V2, i.e., the following condition should be satisfied: $|deviation| < downdeadband + threshold$
- If $deviation > 0, meaningthatthevoltageistoohigh, itshouldbecheckedifthedeviationissmallerbydecreasingV2, i.e., th updeadband + threshold$

The test is done only if the regulated voltage is on one end of the transformer, and it always returns OK if the controlled voltage is remote.

Examples

Example 1

The following example shows how to run a load flow validation on a UCTE network model:

```
$> itools loadflow-validation --case-file 20170322_1844_SN3_FR2.uct --output-folder /tmp/
↪results
```

The validation results, printed to the standard output:

```
Loading case 20170322_1844_SN3_FR2.uct
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type: TWTS -
↪ result: success
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type: FLOWS␣
↪- result: fail
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type: BUSES␣
↪- result: fail
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type: SVCS -
↪ result: success
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type:␣
↪SHUNTS - result: success
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type:␣
↪GENERATORS - result: fail
```

Eventually, you will find in your output-folder one csv file for each validation type.

Example 2

In this example, we are comparing results of two validations: before and after load flow computation. Two additional arguments are needed:

- `load-flow`
- `compare_results`: COMPUTATION

```
$> itools loadflow-validation --case-file 20170322_1844_SN3_FR2.uct --output-folder tmp/
↪loadFlowValidationResults
--verbose --output-format CSV --load-flow --compare-results COMPUTATION
```

The validation results, printed to the standard output:

```
Loading case 20170322_1844_SN3_FR2.uct
Running pre-loadflow validation on network 20170322_1844_SN3_FR2.uct.uct
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type: TWTS -
↪ result: success
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type:␣
↪GENERATORS - result: fail
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type: FLOWS␣
↪- result: fail
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type:␣
↪SHUNTS - result: success
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type: BUSES␣
↪- result: fail
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type: SVCS -
```

(continues on next page)

(continued from previous page)

```
↪ result: success
Running loadflow on network 20170322_1844_SN3_FR2.uct
Running post-loadflow validation on network 20170322_1844_SN3_FR2.uct
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type: TWTS -
↪ result: success
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type:␣
↪GENERATORS - result: fail
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type: FLOWS␣
↪- result: fail
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type:␣
↪SHUNTS - result: success
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type: BUSES␣
↪- result: fail
Validate load-flow results of network 20170322_1844_SN3_FR2.uct - validation type: SVCS -
↪ result: success
```

Eventually, you will find in your output-folder one csv file for each validation type, containing the data pre- and post-computation (load flow).

iTools plugins-info

The `plugins-info` command prints for each kind of plugin, the currently available implementations Ids. The available kind of plugins are:

- exporter
- *import-post-processor*
- importer
- *loadflow-validation computation*

Usage

```
$> itools plugins-info
Plugins:
+---------------------------------+-----------------------------------------------------+
| Plugin type name                | Available plugin IDs                                |
+---------------------------------+-----------------------------------------------------+
| exporter                        | AMPL, XIIDM                                         |
| import-post-processor           | groovyScript, javaScript, loadflowResultsCompletion |
| importer                        | CGMES, UCTE, XIIDM                                  ␣
↪|
| loadflow-validation computation | loadflow, loadflowResultsCompletion                 |
+---------------------------------+-----------------------------------------------------+
```

Maven configuration

To use the `plugins-info` command, add the following dependencies to the `pom.xml` file:

```
<dependency>
    <groupId>com.powsybl</groupId>
    <artifactId>powsybl-tools</artifactId>
```

(continues on next page)

(continued from previous page)

```
    <version>${powsybl.version}</version>
</dependency>
```

iTools run-script

The `run-script` command is used to run scripts based on PowSyBl.

Usage

```
$> itools run-script --help
usage: itools [OPTIONS] run-script --file <FILE> [--help]

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name

Available arguments are:
    --file <FILE>   the script file
    --help          display the help and quit
```

Required arguments

`--file` This option defines the path of the script to execute. Current, only Groovy scripts are supported.

Groovy extensions

The `run-script` command relies on a plugin mechanism to load extensions. Those extensions provide utility functions to make the usage of PowSyBl easier through the scripts. It prevents the user from writing boilerplate code and hides the technical complexity of the framework in more user-friendly functions. PowSyBl provides the following extensions to:

- *load a network from a file*
- *save a network to a file*
- *run a power flow simulation*
- *access to AFS*

Load a network

The `NetworkLoadSaveGroovyScriptExtension` extension adds a `loadNetwork` function to load a network from a file. This function has two parameters:

- the path of the file to load (mandatory)
- a list of properties to configure the importer (optional). The list of supported properties depends on the *grid format*.

To benefit from this feature, add `com.powsybl:powsybl-iidm-scripting` to your classpath.

Example:

```
network = loadNetwork(filename, parameters)
```

Save a network

The `NetworkLoadSave` extension adds a `saveNetwork` function to load a network from a file. This function has four parameters:

- the *format* of the output file (mandatory)
- the network object to save (mandatory)
- a list of properties to configure the exporter (optional). The list of supported properties depends on the *output grid format*.
- the path of the output file (mandatory)

To benefit from this feature, add `com.powsybl:powsybl-iidm-scripting` to your classpath.

Example:

```
saveNetwork(format, network, parameters, file)
```

Run a power flow

The `LoadFlow` extension adds a `loadflow` function to run a *load flow* simulation to a network. This function has two parameters:

- the network object (mandatory)
- the *load-flow parameters* (optional). If this parameter is not set, the parameters are loaded from the configuration.

To benefit from this feature, add `com.powsybl:powsybl-loadflow-scripting` to your classpath.

Example:

```
loadflow(network, parameters)
```

Access to AFS

The `Afs` extension adds a `afs` variable to the groovy binding that offers a facade to access data stored in AFS. This facade has two methods:

- `getFileSystemNames`: this method returns the names of the file system declared in the configuration
- `getRootFolder`: this method returns the root folder of the specified file system. From this root folder, it is possible to navigate in the different folders and open the different projects.

In order to benefit from this feature, add `com.powsybl:powsybl-afs-scripting` to your classpath.

Example

```
fileSystems = afs.getFileSystemNames()
for (String fs : fileSystems) {
    root = afs.getRootFolder(fs)
}
```

Examples

Example 1 - Hello World

The following example shows how to run a simple HelloWorld script. Note that the parameters pass to the command line can be accessed using the `args` array.

Content of the hello.groovy file:

```
print 'Hello ' + args[0]
```

To run this script, pass this file to the `--file` argument:

```
$> itools run-script hello.groovy John
Hello John
```

Example 2 - Run a power flow

The following example shows how to load a network from a file, run a *load flow* simulation and export the modified network to another file. This script is equivalent to the iTools *loadflow* command.

Content of the loadflow.groovy file:

```
import com.powsybl.loadflow.LoadFlowParameters.VoltageInitMode

input_file = args[1]
output_file = args[2]

// Load a network file
network = loadNetwork(input_file)

// Run a power flow, with custom parameters
parameters = new LoadFlowParameters()
parameters.voltageInitMode = VoltageInitMode.DC_VALUES
loadflow(network, parameters)

// Save the network to a file
saveNetwork("XIIDM", network, output_file)
```

To run the previous script, pass the input and output file names:

```
$> itools run-script loadflow.groovy XIIDM /tmp/case.xiidm /tmp/case-lf.xiidm
```

Going further

- Create a Groovy extension: Learn how to create a groovy extension to use it with the `run-script` command

iTools security-analysis

The `security-analysis` command loads a grid file and run a *security analysis* simulation, to detect security violations on pre- or post-contingencies states. At the end of the simulation, the results are printed or exported to a file.

Usage

```
$> itools security-analysis --help
usage: itools [OPTIONS] security-analysis --case-file <FILE>
       [--contingencies-file <FILE>] [--external] [--help] [-I <property=value>]
       [--import-parameters <IMPORT_PARAMETERS>] [--limit-types <LIMIT-TYPES>]
       [--log-file <FILE>] [--output-file <FILE>] [--output-format <FORMAT>]
       [--parameters-file <FILE>] [--with-extensions
```

(continues on next page)

(continued from previous page)

```
       <EXTENSIONS>]

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name

Available arguments are:
    --case-file <FILE>                          the case path
    --contingencies-file <FILE>                 the contingencies path
    --external                                  external execution
    --help                                      display the help and quit
 -I <property=value>                            use value for given importer
                                                parameter
    --import-parameters <IMPORT_PARAMETERS>     the importer configuation file
    --limit-types <LIMIT-TYPES>                 limit type filter (all if not set)
    --log-file <FILE>                           log output path (.zip)
    --output-file <FILE>                        the output path
    --output-format <FORMAT>                    the output format [JSON]
    --parameters-file <FILE>                    loadflow parameters as JSON file
    --with-extensions <EXTENSIONS>              the extension list to enable

Allowed LIMIT-TYPES values are [CURRENT, LOW_VOLTAGE, HIGH_VOLTAGE,
LOW_SHORT_CIRCUIT_CURRENT, HIGH_SHORT_CIRCUIT_CURRENT, OTHER]
Allowed EXTENSIONS values are []
```

Required arguments

`--case-file` This option defines the path of the case file on which the power flow simulation is run. The *supported formats* depend on the execution class path.

Optional arguments

`--contingencies-file` This option defines the path of the contingency files. If this parameter is not set, the security violations are checked on the base state only. This file is a groovy script that respects the *contingency DSL* syntax.

`--external` TODO: Use this argument to run the security analysis as an external process.

`--import-parameters` This option defines the path of the importer's configuration file. It's possible to overload one or many parameters using the `-I property=value` syntax. The list of supported properties depends on the *input format*.

`--limit-types` This option allows filtering certain types of violations. It overrides the default configuration defined in the *limit-violation-default-filter* configuration module. The supported types are the following: `CURRENT`, `LOW_VOLTAGE`, `HIGH_VOLTAGE`, `LOW_SHORT_CIRCUIT_CURRENT`, `HIGH_SHORT_CIRCUIT_CURRENT` and `OTHER`.

`--log-file` TODO

`--output-file` This option defines the path of the result file. If this option is not set, the results are printed to the console.

`--output-format` This option defines the format of the output file. This option is required if the `--output-file` is set. The only supported format is `JSON`.

`--parameters-file` This option defines the path of the *parameters* file of the simulation. If this option is not used, the simulation is run with the default parameters.

`--with-extensions` This option defines the list of extensions to complete the simulation results with additional data. The available extensions are listed in the usage of the command.

Simulators

TODO

Contingencies

TODO

Parameters

TODO

Results

TODO

Examples

Example 1

The following example shows how to run a security analysis simulation to detect only pre-contingency violations for a given network:

```
$> itools security-analysis --case-file 20170322_1844_SN3_FR2.uct
```

The analysis results are printed to the console:

```
Loading network '20170322_1844_SN3_FR2.uct'
Pre-contingency violations:
+--------+---------------------+---------+---------+--------------+----------------+-----
↪-----------+-----------+-----------+------------------+----------------+
| Action | Equipment (1)       | End     | Country | Base voltage | Violation type |␣
↪Violation name  | Value      | Limit     | abs(value-limit) | Loading rate % |
+--------+---------------------+---------+---------+--------------+----------------+-----
↪-----------+-----------+-----------+------------------+----------------+
|        | FFNGEN71 FFNHV111 1 | FFNHV17 | FR      |           27 | CURRENT        |␣
↪Permanent limit | 15350.0808 | 9999.0000 |        5351.0808 |         153.52 |
+--------+---------------------+---------+---------+--------------+----------------+-----
↪-----------+-----------+-----------+------------------+----------------+
```

Example 2

The following example shows how to run a security analysis simulation to detect the post-contingency violations status of a given network and a set of contingencies.

Content of the contingencies.groovy file:``

```
$> cat contingencies.groovy
contingency('HV_line_1') {
    equipments 'NHV1_NHV2_1'
}
```

(continues on next page)

(continued from previous page)

```
contingency('HV_line_2') {
    equipments 'NHV1_NHV2_2'
}
```

To run a post-contingencies security analysis, use the `--contingencies-file` argument:

```
$> itools security-analysis --case-file eurostag_example.xiidm --contingencies-file␣
↪contingencies.groovy
Loading network 'eurostag_example.xiidm'
Pre-contingency violations:
+--------+---------------+-----+---------+--------------+----------------+---------------
↪-+-------+-------+------------------+----------------+
| Action | Equipment (0) | End | Country | Base voltage | Violation type | Violation␣
↪name | Value | Limit | abs(value-limit) | Loading rate % |
+--------+---------------+-----+---------+--------------+----------------+---------------
↪-+-------+-------+------------------+----------------+
Post-contingency limit violations:
+-------------+----------+--------+---------------+-------+---------+--------------+-----
↪-----------+-----------------+-----------+-----------+------------------+--------------
↪--+
| Contingency | Status   | Action | Equipment (4) | End   | Country | Base voltage |␣
↪Violation type | Violation name  | Value     | Limit     | abs(value-limit) | Loading␣
↪rate % |
+-------------+----------+--------+---------------+-------+---------+--------------+-----
↪-----------+-----------------+-----------+-----------+------------------+--------------
↪--+
| HV_line_1   | converge |        | Equipment (2) |       |         |              |    ␣
↪           |                 |           |           |                  |             ␣
↪  |
|             |          |        | NHV1_NHV2_2   | VLHV1 | FR      |          380 |␣
↪CURRENT        | Permanent limit | 1008.9289 |  500.0000 |         508.9289 |        ␣
↪201.79 |
|             |          |        | NHV1_NHV2_2   | VLHV2 | FR      |          380 |␣
↪CURRENT        | Permanent limit | 1047.8260 |  500.0000 |         547.8260 |        ␣
↪209.57 |
| HV_line_2   | converge |        | Equipment (2) |       |         |              |    ␣
↪           |                 |           |           |                  |             ␣
↪  |
|             |          |        | NHV1_NHV2_1   | VLHV1 | FR      |          380 |␣
↪CURRENT        | Permanent limit | 1008.9289 | 1000.0000 |           8.9289 |        ␣
↪100.89 |
|             |          |        | NHV1_NHV2_1   | VLHV2 | FR      |          380 |␣
↪CURRENT        | Permanent limit | 1047.8260 | 1000.0000 |          47.8260 |        ␣
↪104.78 |
+-------------+----------+--------+---------------+-------+---------+--------------+-----
↪-----------+-----------------+-----------+-----------+------------------+--------------
↪--+
```

TODO: to be clean and completed with the following information

with-extensions

Use the `--with-extensions` parameter to activate a list of `com.powsybl.security.interceptors.SecurityAnalysisInterceptor` implementations.

iTools sensitivity-computation

Available commands

The `iTools` script relies on a plugin mechanism: the commands are discovered at runtime and depend on the jars present in the `share/java` folder.

Command	Theme	Description
action-simulator	Computation	Run a security analysis with remedial actions
cim-anonymizer	Data conversion	Anonymize CIM files
compare-security-analysis-results	Computation	Compare security analysis results
convert-network	Data conversion	Convert a grid file from a format to another
dynamic-security-analysis	Computation	Run a dynamic security analysis
dynamic-simulation	Computation	Run a dynamic simulation
list-dynamic-simulation-models	Computation	List all models used by the time domain simulation
loadflow	Computation	Run a power flow simulation
loadflow-validation	Computation	Validate load flow results on a network
plugins-info	Script	Print the currently available implementations for each kind of plugin
run-script	Script	Run a script on top of PowSyBl
security-analysis	Computation	Run a security analysis
sensitivity-computation	Computation	Run a sensitivity analysis

6.2.2 itools-packager

The `itools-packager` Maven plugin provides a way to assemble distribution bundles based on *itools*.

The layout of such distribution is the following:

```
<package-name>
├── bin
│   ├── itools
│   ├── itools.bat
│   └── powsyblsh
├── etc
│   ├── itools.conf
│   └── logback-itools.xml
├── lib
└── share
    └── java
        ├── <jars>
```

Goals overview

The `itools-packager` only has one goal.

- The `package-zip` goal creates a zip package based on `iTools`

Configuration

Properties

archiveName The `archiveName` property defines the basename of the archive. If this property is not set, `itools-packager` uses the packageName.

packageName The `packageName` property defines the name of the root folder of the distribution. If this property is not set, `itools-packager` uses the finalName of the maven project.

configName The `configName` property defines the the basename of the configuration file. The default value is `config`

javaXmx The `javaXmx` property defines the amout of the Java Heap memory. The default value is 8 Gb. This property is used to initialize the `java_xmx` property of the `itools.conf` file.

copyToBin The `copyToBin` property defines the list of files to copy in the `bin` directory of the distribution.

copyToLib The `copyToLib` property defines the list of files to copy in the `etc` directory of the distribution.

copyToEtc The `copyToEtc` property defines the list of files to copy in the `etc` directory of the distribution.

Example

```
<build>
    <plugins>
        <plugin>
            <groupId>com.powsybl</groupId>
            <artifactId>powsybl-itools-packager-maven-plugin</artifactId>
            <version>${powsybl.core.version}</version>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals>
                        <goal>package-zip</goal>
                    </goals>
                </execution>
            </executions>
            <configuration>
                <packageName>powsybl</packageName>
                <archiveName>powsybl-V${project.version}</archiveName>
                <javaXmx>512M</javaXmx>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Usage

The `itools-packager` plugin copies all the maven dependencies to the `share/java` folder of the distribution. To enable a feature, add a runtime dependency to the `pom.xml` file. Refer to the itools *commands list* documentation to learn more about existing commands.

The `itools-packager` tutorial gives the minimal configuration of an `itools` based distribution.

The `iTools` script provides a command-line interface to interact with PowSyBl, available under Linux and Windows (macOS is not supported yet).

An `iTools` package is constituted of:

- a `bin` directory containing the executable scripts and the binaries
- an `etc` directory containing the configuration and the `XIIDM` schemas
- a `lib` directory containing C++ libraries
- a `share/java` directory containing Java libraries

6.2.3 Installation

It is possible to install a basic PowSyBl distribution and to start running iTools commands from the binaries or from the sources

Install from the binaries

To use the executable tool provided with the `powsybl-distribution` repository, follow these steps:

- Download the zip folder from the latest released version of powsybl-distribution
- Unzip the downloaded package
- Add `<INSTALL_DIR>/powsybl-distribution-<LATEST_VERSION>/bin` to your environment variable `PATH`.

Install from the sources

To generate the executable tool from the sources, follow these steps:

- Download the powsybl-distribution sources
- Checkout the latest stable version by referencing the tag
- Generate the executable with the maven command

```
$ git clone https://github.com/powsybl/powsybl-distribution.git
$ cd powsybl-distribution
$ git checkout tags/<LATEST_RELEASE_TAG> -b latest-release
$ mvn clean package
```

The distribution is generated in the `target` folder.

- Add `<INSTALL_DIR>/powsybl-distribution-<LATEST_VERSION>/bin` to your environment variable `PATH`.

Test your installation

Launch the `itools --help` command in your terminal to check that everything went smoothly.

```
$> itools --help
usage: itools [OPTIONS] COMMAND [ARGS]

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name

```

(continues on next page)

(continued from previous page)

```
Available commands are:

Computation:
    compare-security-analysis-results        Compare security analysis results
    loadflow                                 Run loadflow
    loadflow-validation                      Validate load-flow results of a network
    security-analysis                        Run security analysis

Data conversion:
    convert-network                          convert a network from one format to another

Misc:
    plugins-info                             List the available plugins

Script:
    run-script                               run script (only groovy is supported)
```

6.2.4 Usage

The `iTools` script is available in the `bin` directory.

```
$> ./bin/itools
usage: itools [OPTIONS] COMMAND [ARGS]

Available options are:
    --config-name <CONFIG_NAME>   Override configuration file name

Available commands are:
```

`--config-name` Use this option to overload the default base name for the configuration file. It overrides the `powsybl_config_name` property defined in the `itools.conf` file.

6.2.5 Configuration

The `iTools` script reads its configuration from the `<ITOOLS_PREFIX>/etc/itools.conf` property file. The properties defined in this file are used to configure the Java Virtual Machine.

Example of itools.conf file:

```
# PowSyBl configuration directories
#powsybl_config_dirs=

# PowSyBl configuration base name
powsybl_config_name=config

# Maximum size of the Java memory allocation pool
java_xmx=8G
```

You can set a default configuration `config.yml` by copying the provided configuration file in your `<HOME>/.itools` repository (note that you will need to create this repository if it does not exist):

```
$ mkdir <HOME>/.itools
$ cp <INSTALL_DIR>/resources/config/config.yml <HOME>/.itools/config.yml
```

This step is not mandatory **if you already have a custom configuration file and the necessary configuration modules are filled**.

powsybl_config_dirs This is an optional property that defines the list of the folders where the configuration files are located. If this property is not set, the configuration files are read from `<USER_HOME>/.itools` and `<ITOOLS_PREFIX>/etc` folders. Note that the order of the folder is really import as the PowSyBl configuration is stackable.

powsybl_config_name This is an optional property that defines the base name of the configuration files. The default value for this property is `config`.

java_xmx This is an optional property that defines the maximum size of the memory allocation pool of the JVM. The default value for this property is 8 gigabytes.

At startup, Powsybl looks in the configuration directories defined in the `powsybl_config_dirs` for a YAML configuration file first, for a XML configuration file then and finally for properties files.

All the configuration files are stacked to allow a user to partially or totally overload the system configuration of a module.

Deprecated properties

itools_cache_dir The `itools_cache_dir` property is deprecated since V2.2.0. The `itools_cache_dir` property was an optional property that defined the path to the cache folder used by modules that required cache functionalities. The default value was `$HOME/.cache/itools`.

itools_config_dir The `itools_config_dir` property is deprecated since V2.2.0. Use the `powsybl_config_dirs` property instead.

itools_config_name The `itools_config_name` property is deprecated since V2.2.0. Use the `powsybl_config_name` property instead.

6.2.6 Logging

The `iTools` script uses logback as a logging framework. To configure the logging framework, edit the `<ITOOLS_HOME>/etc/logback-itools.xml` configuration file. Please refer to the logback manual for the available logging options.

Sometimes, it could be useful for a user to have its own logging configuration to filter unexpected logs or to have more details for some features. The simplest way to proceed is to copy the global configuration file in the `<USER_HOME>/.itools` folder and then customize it.

Example of logback-itools.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <Pattern>%d{yyyy-MM-dd_HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n
↪</Pattern>
        </encoder>
    </appender>
    <root level="ERROR">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

6.2.7 Examples

Converting a UCTE network file to XIIDM

In this example, you will convert a UCTE file describing the Belgian network to XIIDM format. The `beTestGridForMerging.uct` file available in `<POWSYBL-CORE_DIR>/ucte/ucte-util/src/test/resources` repository will be used.

Run the following command:

```
$ itools convert-network --input-file <POWSYBL-CORE_DIR>/ucte/ucte-util/src/test/
↪resources/beTestGridForMerging.uct --output-file <HOME>/beTestGridForMerging --output-
↪format XIIDM
Generating file <POWSYBL-CORE_DIR>/beTestGridForMerging.xiidm...
```

Once the command is completed, the XIIDM file describing the Belgian network will be present as `beTestGridForMerging.xiidm` in your `HOME` repository.

Update an XIIDM network file after running a load-flow

In this example, you will update an XIIDM file describing the example Eurostag network after running a load-flow using `powsybl-open-loadflow`. The `eurostag-tutorial-example1.xml` file available in `<POWSYBL-CORE_DIR>/iidm/iidm-serde/src/test/resources/V1_0` repository. Please note that this will permanently change the file. In order to keep it, you can start by copying it:

```
$ cp <POWSYBL-CORE_DIR>/iidm/iidm-serde/src/test/resources/V1_0/eurostag-tutorial-
↪example1.xml <HOME>/eurostag-tutorial-example1.xiidm
```

Run the following command:

```
$ itools loadflow --case-file <HOME>/eurostag-tutorial-example1.xiidm --output-case-file
↪<HOME>/eurostag-tutorial-example1.xiidm --output-case-format XIIDM
Loading network '/home/caronali/Téléchargements/eurostag-tutorial-example1.xiidm'
ERROR c.p.o.a.o.DistributedSlackOuterLoop - Failed to distribute slack bus active power␣
↪mismatch, -1.44 MW remains
Loadflow results:
+-------+--------+---------+
| Ok    | Status | Metrics |
+-------+--------+---------+
| false | FAILED | {}      |
+-------+--------+---------+
Components results:
+--------------------+----------------------+--------+---------------------------------
↪-------------------------------------------------------------+---------+----------------
↪+--------------+-------------------------+---------------------------------+
| Connected component | Synchronous component | Status | Status text                    ␣
↪                                                             | Metrics | Iteration count␣
↪| Slack bus ID | Slack bus mismatch (MW) | Distributed Active Power (MW) |
+--------------------+----------------------+--------+---------------------------------
↪-------------------------------------------------------------+---------+----------------
↪+--------------+-------------------------+---------------------------------+
| 0                  | 0                    | FAILED | Outer loop failed: Failed to␣
↪distribute slack bus active power mismatch, -1.44 MW remains | {}      | 3              ␣
↪  | VLHV1_0      | -1,44                  | 0,00000                         |
+--------------------+----------------------+--------+---------------------------------
```

(continues on next page)

(continued from previous page)

```
↪-----------------------------------------------------------+---------+-----------------
↪+--------------+-------------------------+--------------------------------+
```

Once the command is completed, the `eurostag-tutorial-example1.xiidm` file will be updated to contain post load-flow results, including calculated bus voltage, calculated bus angles and calculated flows.

6.2.8 Going further

The following links could also be useful:

- Bundle an iTools package: Learn how to use the `itools-packager` maven plugin
- Create an iTools command: Learn how to create your own `iTools` command in Java

6.3 Functional logging

6.3.1 Export

A report node can be either serialized to a JSON file or printed to a string / a text file.

JSON format

A report node can be serialized into a JSON file

- either by calling

```
var objectMapper = new ObjectMapper().registerModule(new ReportNodeJsonModule());
objectMapper.writeValue(outputStream, reportNode);
```

- or by calling

```
ReportNodeSerializer.write(reportNode, jsonFilePath);
```

An example of the current version 3.0 of the serialization is below:

```
{
  "version" : "3.0",
  "dictionaries" : {
    "default" : {
      "key1" : "template with typed ${value1}",
      "key2" : "template with several untyped ${value1}, ${value2}"
    }
  },
  "reportRoot" : {
    "messageKey" : "key1",
    "values" : {
      "value1" : {
        "value" : "value",
        "type" : "FILENAME"
      }
    },
    "children" : [ {
      "messageKey" : "key2",
```

(continues on next page)

(continued from previous page)

```
      "values" : {
        "value1" : {
          "value" : "first untyped value"
        },
        "value2" : {
          "value" : "second untyped value"
        }
      }
    } ]
  }
}
```

Display format

To get an overview of a report node, several print methods are provided in the API, with the possibility to provide your own `Formatter` - `Formatter` is a functional interface that specifies how to get a `String` from a `TypedValue`.

- To print to a `Path`:

```
reportNode.print(path);
reportNode.print(path, formatter);
```

- To print to a `Writer`:

```
reportNode.print(writer);
reportNode.print(writer, formatter);
```

In both cases, giving a custom formatter allows to do specific formatting based on types for instance. If no formatter is provided, the default one is used:

```
typedValue -> typedValue.getValue().toString()
```

The corresponding multiline string of above example is below. The + character and the indentation are used to show the tree hierarchy.

```
+ template with typed value
   template with several untyped first untyped value, second untyped value
```

6.3.2 Import

The report node JSON file obtained from `ReportNode` serialization can be deserialized

- from a file `Path`

```
ReportNodeDeserializer.read(path);
```

- from an `InputStream`

```
ReportNodeDeserializer.read(inputStream);
```

Version

Currently, the serialized versions supported are the versions 2.1 and 3.0.

Dictionaries

The two methods above look for a `default` dictionary in the `dictionaries` list. If no `default` dictionary is found, the first entry of the list is taken.

If several dictionaries are defined in the JSON file, we can choose which dictionary has to be used for deserialization by providing the dictionary name:

```
ReportNodeDeserializer.read(path, dictionary);
ReportNodeDeserializer.read(inputStream, dictionary);
```

Similarly, if the chosen dictionary cannot be found, the first entry of the list is taken.

6.3.3 Internationalization

ResourceBundle is used to handle the `ReportNode` messages internationalization. It enables to load locale translation from properties files containing locale-specific data (dictionary with `key = value`).

As usual for resource bundles:

- the default translation for any translation key must be present in the `reports.properties` file,
- a locale dictionary is represented by a `reports_<locale>.properties` file in the bundle with the locale as suffix (for instance: for the locale `en_US` the dictionary is in the `reports_en_US.properties`),
- if no dictionary is present for a given locale (for any of its suffixes) then the mechanism is to fall back to the JVM default locale dictionary,
- if a translation key is not present in a locale dictionary the mechanism is to fall back to a more general dictionary and if there is still no match the ultimate fallback is on the default dictionary.

Currently, only US English (default language) and French dictionaries are maintained in powsybl repositories. If other languages are needed, you need to add your resource bundle files to your classpath, with the appropriate base name and file name.

Best practices

Keys

The keys in the properties files are expected to be ordered in alphabetical order.

The keys are expected to be prefixed with a repository prefix, followed by a module prefix, possibly a submodule prefix, all separated with a `.` character. For instance, the key `core.iidm.modification.voltageLevelRemoved` corresponds to a report in powsybl-iidm-modification module, which is a submodule of powsybl-iidm inside the powsybl-core repository.

Bundles

To facilitate the translations, it is recommended to have a single resource bundle per repository. For instance, all the reports within powsybl-core are gathered into a single resource bundle, located in the `commons` module, its bundle base name being `com.powsybl.commons.reports`.

Services

To make your resource bundle easily available to the user with the `ReportNodeBuilder::withAllResourceBundlesFromClasspath` method, please add an implementation of `ReportResourceBundle` marked as a service for all the added bundles.

Unit test

If test message templates are needed, it is recommended to have the following test resource bundle basename: `"i18n.reports"`.

Going further

- If you need internationalization but ResourceBundle does not suit your needs, you can add a custom message template provider at the ReportNode root creation. Your custom message template provider needs to provide a string based on a message key and a `Locale`.
- The ReportNode provided serializer writes the dictionary of all the keys used in the `Locale` provided at root creation. If you need a multilingual JSON file, note that the JSON format supports a list of dictionaries. The ReportNode deserializer allows to choose one dictionary among the provided list (see *import*).

For functional logging powsybl provides an API called `ReportNode`, and a default implementation called `ReportNodeImpl`. In these functional logs we expect non-technical information useful for an end-user, about the various steps which occurred, what was unexpected, what caused an execution failure.

Each `ReportNode` contains

- several values, indexed by their keys,
- a functional message, given by a message template referenced by a key, which may contain references to those values or to the values of one of its ancestors,
- a list of `ReportNode` children.

Several `ReportNode` connected together through their children parameter define a tree. Each `ReportNode` of such a tree refers to the same `TreeContext`, which holds the context related to the tree.

Each message template is identified by a key. This key corresponds to an internationalized message template: a `ResourceBundle` usually links the key to the template in the desired language (see *i18n page* for more information). A dictionary key / message template is build in the `TreeContext`: this allows to serialize in an efficient way the tree, by not repeating the duplicated templates.

When a `ReportNode` has several children, the message of the corresponding `ReportNode` should summarize the children content. To that end, one or several values can be added after the `ReportNode` construction. The summarizing template should be succinct: 120 characters is an indicative limit for the message string length (once formatted).

6.3.4 Values

Only `float`, `double`, `int`, `float`, `boolean` and `String` values are supported in the API.

Each value can be either typed or untyped. A typed value is a value with a `String` parameter, which gives more insight to what the value is. This allows a GUI displaying the reports to, for instance,

- insert a hyperlink to the voltage level visualization when a voltage level is mentioned,
- insert all the parameters of the equipment which is mentioned,
- insert a link to the file mentioned,
- round some values to a given precision given their type,
- change the unit of the corresponding displayed values,
- ...

Some value types are provided by default, for instance:

- `TypedValue.TIMESTAMP`,
- `TypedValue.FILENAME`,
- `TypedValue.VOLTAGE_LEVEL`,
- `TypedValue.ACTIVE_POWER`,
- ...

6.3.5 Severity

A severity is a specific typed value. It is a `String` of type `TypedValue.SEVERITY`. The `ReportNode` builder API gives direct access to set this value.

The following default severity values are provided:

- `TypedValue.TRACE_SEVERITY`,
- `TypedValue.DEBUG_SEVERITY`,
- `TypedValue.INFO_SEVERITY`, for reports about a functional state which may be of interest for the end user,
- `TypedValue.WARN_SEVERITY`, for reports about an unwanted state that can be recovered from,
- `TypedValue.ERROR_SEVERITY`, for reports about a requested operation that has not been completed,
- `TypedValue.DETAIL_SEVERITY`, for reports which are children of a `ReportNode` of severity `WARN` or `ERROR`.

The number of `WARN` and `ERROR` reports should be as small as possible. That is, similar detailed reports about an unwanted state should be grouped, if possible, as children of the same `ReportNode`. This father `ReportNode` should carry the `WARN` or `ERROR` severity, whereas these `ReportNode` children should have a `DETAIL` severity. This allows to give fine-grained information about an unwanted state, without overwhelming the end-user with numerous `WARN` or `ERROR` reports and while keeping a succinct message for each report.

6.3.6 Builders / adders

The builder API is accessed from a call to `ReportNode::newRootReportNode` method. This API is used to build the root `ReportNode`.

The following methods are available in the builder API to define the corresponding `ReportNode` tree:

- `withAllResourceBundlesFromClasspath()`, to set the message template provider as based on all the `ResourceBundle` gathered by the `ServiceLoader` of `ReportResourceBundle` implementations,
- `withResourceBundles(String... bundleBaseNames)`, to set the message template provider as based on one or several `ResourceBundle`,
- `withDefaultTimestampPattern(pattern)`, for the pattern to be used in the tree when a timestamp is added,
- `withLocale(locale)`, for specifying the `Locale` to use in the whole tree:
  - for message templates (see *i18n page*),
  - for timestamps.

The adder API is accessed from a call to `reportNode.newReportNode()`. It is used to add a child to an existing `ReportNode`.

Both API share the following methods to provide the message template and the typed values:

- `withMessageTemplate(key)`, the key referring to a template in a `ResourceBundle` (see *i18n page*),
- `withUntypedValue(key, value)`,

- `withTypedValue(key, value, type)`,
- `withTimestamp()`, adding a typed value with node creation timestamp,
- `withSeverity(severity)`, severity being either a `String` or a `TypedValue`.

For further customization, the following methods are also available:

- `withMessageTemplateProvider(messageTemplateProvider)`, to specify how to get a message template from a given key and locale for all the descendents of the node to create, unless overridden,
- `withTimestamp(pattern)`, if a custom pattern has to be used instead of the default one specified at root construction,
- `withResourceBundles(String... bundleBaseNames)` is also shared by the adder API, to override the resource bundles to use as message template provider for all the descendents of the node to create (unless overridden).

6.3.7 Merging ReportNodes

Include

An `include` method is provided in the API in order to fully insert a given root `ReportNode` as a child of another `ReportNode`. The given root `ReportNode` is becoming non-root after this operation. This was meant for including the serialized reports obtained from another process.

AddCopy

An `addCopy` method is provided in the API to partly insert a `ReportNode`: unlike `include`, the given node does not need to be root. The given `ReportNode` is copied and inserted as a child of the `ReportNode`.

Two known limitations of this method:

1. the inherited values of copied `ReportNode` are not kept,
2. the resulting dictionary contains all the keys from the copied `ReportNode` tree, even the ones from non-copied `ReportNode`s.

6.3.8 Example

Resource bundle property file in `com/powsybl/commons/reports.properties` which is the default translation values file.

```
translationKey = Import file ${filename} in ${time} ms
task1 = Doing first task with double parameter ${parameter}
task2 = Doing second task, reading ${count} elements, among which ${problematicCount}␣
↪are problematic
problematic = Problematic element ${id} with active power ${activePower}
```

```
ReportNode root = ReportNode.newRootReportNode()
        .withAllResourceBundlesFromClasspath()
        .withMessageTemplate("translationKey")
        .withTypedValue("filename", "file.txt", TypedValue.FILENAME)
        .build();
long t0 = System.currentTimeMillis();

ReportNode task1 = root.newReportNode()
        .withMessageTemplate("task1")
```

(continues on next page)

(continued from previous page)

```
        .withUntypedValue("parameter", 4.2)
        .withSeverity(TypedValue.INFO_SEVERITY)
        .add();

ReportNode task2 = root.newReportNode()
        .withMessageTemplate("task2")
        .withUntypedValue("count", 102)
        .withUntypedValue("problematicCount", 2)
        .withSeverity(TypedValue.WARN_SEVERITY)
        .add();

// Supposing a list of problematic elements has been build, each containing an id and an␣
↪active power values
for (ProblematicElement element : problematicElements) {
    task2.newReportNode()
            .withMessageTemplate("problematic")
            .withTypedValue("id", element.getId(), TypedValue.ID)
            .withTypedValue("activePower", element.getActivePower(), TypedValue.ACTIVE_
↪POWER)
            .withSeverity(TypedValue.DETAIL_SEVERITY)
            .add();
}

// Putting a value afterward in a given reportNode
long t1 = System.currentTimeMillis();
root.addTypedValue("time", t1 - t0, "ELAPSED_TIME");
```

6.4 Frequent errors

6.4.1 Frequent error messages

No NetworkFactoryService providers found

This most likely happened because you tried to read a file, but didn't have any implementation of the IIDM API in your classpath. Use the implementation provided by PowSyBl / a third party, or write your own.

Fixing the issue by using powsybl-core implementation

If you wish to use powsybl-core in-memory implementation, add the following to your `pom.xml` in the `<dependencies>` section:

```
<dependency>
    <groupId>com.powsybl</groupId>
    <artifactId>powsybl-iidm-impl</artifactId>
    <version>$version$</version>
</dependency>
```

Using the correct version

Replace `$version$` by the version of this impl that will work with other PowSyBl packages. A comprehensive table of matching versions can be found at powsybl-dependencies; use the `powsybl-core` version that matches for

you.

You should also make sure that you have an implementation of `SerDe` if you are trying to read a file. See *adding SerDe*.

Unsupported file format or invalid file

You are most likely missing in your classpath the implementation of `com.powsybl.iidm.network.Importer` corresponding to the file you are trying to import (`XmlImporter`, `CgmesImporter`, etc.). Use the implementation provided by PowSyBl / a third party, or write your own.

Example for an IIDM file

If you wish to import an IIDM file using powsybl-core implementation, add the following to your `pom.xml` in the `<dependencies>` section:

```
<dependency>
    <groupId>com.powsybl</groupId>
    <artifactId>powsybl-iidm-serde</artifactId>
    <version>$version$</version>
</dependency>
```

Using the correct version

Replace `$version$` by the version of this impl that will work with other PowSyBl packages. A comprehensive table of matching versions can be found at powsybl-dependencies; use the `powsybl-core` version that matches for you.

You can find here some common errors encountered by people starting on PowSyBl, the reason and how to solve those problems.