Purpose of this Document

The purpose of this document is to describe a standard method for articulating technical functions and algorithms in such a way as to be understood by all interested parties. In the SmILES context, the documentation will support the translation of the control functions present in simulation models between simulation environments. This will enable the migration of simulation cases between partners and the dissemination of these cases to the public.

This document is an abridged version of the Scientific Algorithm Documentation Standard[1] developed by NOAA. It has been adapted to the needs of SmILES, and its scope has been broadened to cover the entire definition of "Control Functions" in the SmILES context.

*A **control function (CF)**, in the context of SmILES, is a description of an embedded control system (or an aspect thereof) which, together with the inherent physical properties of the system itself, defines the behaviour of the control system and its response to dynamic input. Control functions can be seen as a further detailing, or more formal description, of the mechanisms governing the behaviour of individual actors in a use case.*

Control functions may take the form of mathematical functions, such as e.g. transfer functions or descriptions of a stochastic process, or they may be best described by an algorithm. These different forms require different description options. As a result, some of the sections of the document are optional because their content does not apply to all descriptions. This is indicated at the beginning of each section. Optional sections should be left blank, rather than being deleted entirely, in order to preserve the consistency of section numbers. Sections marked as mandatory must be filled in every case.

Explanatory text and/or examples for each section are provided in gray. All gray text should be deleted when the section is filled in, or when optional sections are left blank.

1. Document History

This section is mandatory.

Date	Versions	Description	Author
21-04-20	1.0	Initial draft	Petra Raussi
04-06-21	1.1		Felipe Castro
02-11-21	1.2	Edit. comments	Gunter Arnold

2. Functional Description

[1] *National Oceanic and Atmospheric Administration (NOAA), National Weather Service, Office og Hydrologic Development: "Scientific Algorithm Documentation Standard", August 2006*

This section is mandatory.

The purpose here is to provide a brief introduction to the problem solved by the function or the algorithm. The technical background should be outlined by giving an overview of the function or algorithm. It will be important in the early introduction of the document to clearly inform the reader on the intent of the function or algorithm as well as the problem the function or algorithm is addressing. The author should keep in mind that this document is not a technical paper and should be written in such a way that it can be understood by an audience with limited expertise in specific simulation domains.

The statement of the problem should include a clear statement outlining the problem and why the problem is important and needs to be addressed. It should contain a sketch of the problem/situation and, if known and/or relevant, the background history that led to the development of the function or algorithm, including why the problem was investigated and how it relates to earlier work that has been done in the field. It should show that the author thoroughly understands the problem.

This control function implements a voltage control algorithm. This algorithm is responsible of calculating the inputs of the controllable elements of a distribution network that would result on the optimal operation point of the system. The optimal operation point is calculated based on three criteria: busbar voltage values within a certain user-defined range, reactive power flowing through the high voltage side of the transformer that connects the high and medium voltage networks within a user defined range, and the average value of the voltage of all the medium network busbars to remain as close to 1 p.u. as possible. The algorithm continuously calculates inputs for the tap position of the transformers, the step position of shunt elements (capacitors and/or reactors), and the reactive power contribution (or absorption) of the generating sources located within the distribution network.

3. Methodology or Theory (equations and computations)

This section is mandatory.

This section is used to provide the methodology or theory of the function or algorithm in detail. Relevant research results should be referenced. The description should not be as detailed as a peer-review journal paper. However, all information necessary for understanding specifics of the function or algorithm, especially for translating it into software, must be included. Required equations and computations should be described and explained term by term and be closely related to the detailed description of the function or algorithm later in the document.

In case of well-known or standard functions or algorithms, a simple reference to the theory (e.g. a link to the Wikipedia page for "PID controller") is sufficient.

The algorithm continuously receives as input the current tap position of the transformers of the power distribution network, the current operational status (on service/out of service) of the shunt elements, and their step position.

The algorithm executes a load flow for all possible combinations of the following operation scenarios:

- Tap position: considering the current tap position and the following **“X”** positions to both sides, bounded by the maximum and minimum tap position allowed by the transformer (denoted as **“TapMax”** and **“TapMin”**, respectively).
- Step position: considering all the step positions of the shunt elements, bounded by their maximum step position, denoted as **“StepMax”**.
- Power factor of the generating sources: considering, at the same time for all the generating sources, a power factor of **“PF”** lag, **“PF”** lead, and a power factor of “1”.

The user may define the variables **“X”** and **“PF”**:

- **“X”**: defines how many positions away from the current tap position will be considered for the combination of operational scenarios for the load-flow calculation (e.g. if the current tap position is 1, and “X” is set to “2”, then the tap positions considered by the algorithm are “-1”, ”0”, ”1”, ”2” and “3”)
- **“PF”**: defines the power factor that will be considered for all of the generating sources at the same time. For example, if “PF” is set to “0.95”, the algorithm will consider “0.95 lag”, “1” and “0.95 lead” as the power factor of all the generating sources, for the combination of the operational scenarios.

The variables **“TapMax”**, **“TapMin”** and **“StepMax”** must be set by the user considering the characteristics of the elements involved (the transformer and the shunt element).

With these inputs, the algorithm executes a load flow for each of the operational scenarios originated by the combination of these inputs. As an example, if the power distribution system counts with one HV/MV transformer, and “X” is set to “1”, then there are three possible values for the tap position of the transformer (the current value, one tap down and one tap up, if the bounds of the tap position of the transformers are not reached). In another hand, if there is one shunt element with 3 step positions, then there are four possible values (“0”,”1”,”2” and “3”). Finally, there are 3 possibilities of reactive power contribution for the generating sources of the distribution system “PF lag”, “1” and “PF lead”. Therefore, there is a total of (3 tap positions x4 step positions x3 power factor possibilities) 36 different operational scenarios. In case the operational status of the capacitor bank is set as “out of service”, only the step position “0” of the shunt element is assessed.

For each load flow calculation, the number of busbars for which the voltage value is outside a predefined range is calculated. This range is defined by the user by setting the value **“V_deviation”.** As an example, if this variable is set to 0.03, then the algorithm counts for each load flow the number of busbars that present a voltage outside of the range (0.97-1.03 p.u.). this amount is denoted as **“number of violations”.** Moreover, the average of the voltage of all the busbars located in the medium voltage network is calculated and denoted as **“mean voltage”**. Then the absolute deviation of this variable to the nominal value (1 p.u.) is calculated and denoted as **“mean deviation”**.

Additionally, for each load flow, the resulting reactive power flowing in the primary (HV) side of the transformer is obtained. This result is denoted as **“Q transformer”.**

Finally, for each load flow, the tap position, the step position, and the power factor of the generating sources are denoted as **"tap"**, **"step"** and **"PFgen"**, respectively.

The following results are stored in each row of a matrix denoted **"Performance"**:

"Number of violations" | "Q transformer" | "mean deviation" | "tap" | "Step" | "PFgen"

After executing each load flow calculation, the rows of the performance matrix are sorted according to the following criteria:

- The user defines a permissive reactive power window for the reactive power flowing in the HV side of the transformer by setting **"Q_permissive"**. The permissive operation range for this value is (-"Q_permissive" / "Q_permissive")
- The rows which have a value of "Q transformer" within the permissive reactive power range are separated in a matrix denoted **"Performance_1"**, and the rest of the rows are stored in a matrix denoted **"Performance_2"**.

The matrix "Performance_1" is sorted as follows:

- In a first level, the rows are sorted increasingly according to the "Number of violations"
- In a second level, the rows are sorted increasingly according to the "mean deviation"
- In this matrix, the variable "Q transformer" is irrelevant, because it already complies with its value being within the user-defined range.

The matrix "Performance_2" is sorted as follows:

- In a first level, the rows are sorted increasingly according to the "Q transformer"
- In a second level, the rows are sorted increasingly according to the "Number of violations"
- In a third level, the rows are sorted increasingly according to the "mean deviation"

The final output of the algorithm are the values "tap", "step" and "PFgen", which represent the inputs to the controllable elements of the power distribution grid for obtaining the optimal operation point of the assessed combination. These values are obtained from the first row of the sorted matrix "Performance_1". If this matrix is empty, then these values are obtained from the first row of the sorted matrix "Performance_2".

4. Limitations

This section is mandatory.

The Limitations section will describe prerequisites for the function or algorithm. These include limitations caused by input data, special types of data required, limitations inherent to the function or algorithm, situations where the function or algorithm cannot be applied, potential implementation difficulties, and any other problems known to be associated with the function or algorithm. This section should also describe both the implementation and operational impacts of these limitations.

The limitations of this code are the following:

- A tool for calculating load flow calculations is needed for this code. In the implementation of this benchmark, the python module PandaPower is used to execute the load flows in the algorithm.
- The model of the power system used to execute load flows within the algorithm has to be identical to the power system from where the inputs of this algorithm are sent.
- The algorithm is currently coded to support power system models in which just one transformer and one shunt element are controlled.
- The current power system model provided in this benchmark corresponds to a reduced version of the 15-busbar CIGRE MV Benchmark model.

5. Use Cases

This section is mandatory.

Use Cases are a way to express functional requirements. They bridge the gap between user needs and system functionality by directly stating the user's intention and system response for each step in a particular interaction. No single Use Case specifies the entire requirements of the system. A Use Case itself is an interaction that a User or another System has with the system that is being designed in order to achieve a goal. Different scenarios within a Use Case show how the goal succeeds or fails. A success scenario is one in which the goal is achieved; a failure scenario is one where the goal is not achieved. Because the goals summarize the intention of the various uses of the system, the users can see how they are supposed to use the system. Users can also spot when the system does not support all of their goals without having to wait for the first prototype or having to wait for the system to be developed.

This is a summary of how to write Use Cases:

- *The name of the use case should start with a strong verb.*
- *A Use Case is a set of scenarios. A scenario is a list of steps.*
- *Each step should state what the user does and/or how the system responds.*
- *The steps must not mention how the system does something.*
- *Each step needs to be analyzed in detail before it becomes code.*
- *Keep It Simple: use the simplest format you need.*
- *Keep track of different versions.*
- *Writing use cases is a team sport.*
- *Focus on a particular user (give them a name).*
- *Don't get bogged down in all the special ways it can go wrong until you've finished the main success story.*

Use Case Example	Analyzes Model input and output time series data (forecast and calibration mode)
Date created:	09/08/05
Actor:	Research Scientist, RFC Hydrologic Forecaster
Description:	DHM will use existing software to analyze input and output time series data

Preconditions:	User must have previously ran Model
Postconditions:	User must be satisfied with results and should also have knowledge on how to edit model data
Priority:	High
Frequency of use:	Daily or as often as every hour
Normal course:	Forecast Mode: Use existing procedures to get IFP to display input/output time series data (e.g. a PLOT-TUL or PLOTTS operation has been previously defined to show the desired data)
Alternative course:	Calibration Mode: Use existing procedures to get ICP to display input/output time series data (e.g. a PLOTTS operation has been previously defined to show the desired data), or use XDMS.
Exceptions:	
Assumptions:	User must have familiarity with the existing river forecasting operations workflow
Notes and Issues:	

6. Embedding (implicit functions or algorithms)

This section is optional. It should be filled in if the described function or algorithm is implicitly contained in a simulation model, solver, optimizer etc.

This section will describe in which way the documented functions or algorithms are implicitly contained or embedded in simulation models, solvers, optimization algorithms etc. This embedding may take the form of assumptions or abstractions. An example may be an optimization algorithm which assumes that an energy system is always in balance between supply and demand. However, no physical law guarantees this balance to be present at all times. Instead, if this system were to be implemented, a dedicated controller would have to maintain the balance. In order to reproduce the simulation results on another simulation tool, the function or algorithm which governs the behaviour of this controller would have to be extracted and documented. Otherwise, the reproduced simulation is likely to behave differently to the original simulation, especially if the two simulations are of a different type (e.g. transferring from an optimization tool to a timeseries simulator).

The documentation in this section should not focus on the description of the function or algorithm itself – this should be done in subsequent sections – but on describing how the function or algorithm is embedded into the simulation tool. This could be done by e.g. describing the underlying assumptions which imply the existence of the function of algorithm.

7. Inputs

This section is mandatory.

This section will describe all required data inputs to the documented function or algorithm. This includes static input data such as configuration or parametrization as well as dynamic input data such as measurement data or events. Each input data element should specify the data type (e.g. floating point, integer, character string), units (e.g. degrees, percentages), valid ranges (e.g. 0 – infinity; -1.0 – +1.0 inclusive), the expected update rate of the data (e.g. static; 10Hz; daily) and a short description of the information the variable contains (e.g. "Width of the radar beam in degrees …").

All data element names should be self-descriptive (i.e. the name should describe what the data element contains). Widely recognized symbols from textbooks and professional literature are acceptable.

Name	Current tap position
Type	Scalar
Unit	none
Range	Defined by user (in this model -3 /+3)
Expected update rate	20 seconds
Description	Contains the information of the current tap position of the transformer in the power system that the algorithm is controlling.

Name	Current step position
Type	Scalar
Unit	none
Range	Defined by user (in this model 0 - 3)
Expected update rate	20 seconds
Description	Contains the information of the current step position of the shunt element (capacitor) in the power system that the algorithm is controlling.

Name	Current status of the shunt element
Type	Scalar (binary)
Unit	None
Range	0 or 1
Expected update rate	20 seconds
Description	Contains the information of the current operation status (on service or out of service) of the shunt element (capacitor) in the power system that the algorithm is controlling.

8. Diagrams (data flow diagrams, sequence diagrams, logic diagrams, state diagrams, control hierarchy, etc.)

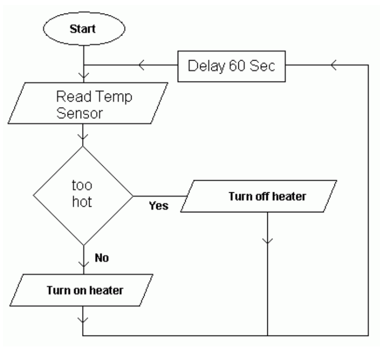
This section is mandatory.

This section will contain diagrams which capture the data flow occurring between a function or algorithm and the rest of the system. These diagrams should reveal relationships among and between the various components in a program or system.

In case of distributed controllers, i.e. multiple processes communicating to achieve an overall objective, sequence diagrams of the interaction between the individual processes must be provided here, in addition to the detailed descriptions of the individual processes in subsequent sections.

Where applicable, this section should also contain Logic Diagrams and/or State Diagrams. Logic Diagrams represent logical concepts and State Diagrams represent the behavior of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system.

Thermostat flowchart as example for a logic diagram.



9. Deterministic Functions

This section is optional; however, at least one of the three detailed description sections (Deterministic functions, stochastic functions, algorithms) must be filled in.

This section will contain a mathematical description of the deterministic (as opposed to stochastic) part of a function. Each mathematical formula should be followed by a brief explanation of the symbology.

The section can also be used to document the mathematical background of an algorithm for which pseudocode is provided in the "Algorithms" section below. In this case, the description should also cover the method used to realize the implementation in the Algorithms section.

10. Stochastic Functions

This section is optional; however, at least one of the three detailed description sections (Deterministic functions, stochastic functions, algorithms) must be filled in.

This section will contain a mathematical description of the stochastic (as opposed to deterministic) part of a function. Each mathematical formula should be followed by a brief explanation of the symbology.

The section can also be used to document the mathematical background of an algorithm for which pseudocode is provided in the "Algorithms" section below. In this case, the description should also cover the method used to realize the implementation in the Algorithms section.

11. Algorithms (pseudocode)

This section is optional; however, at least one of the three detailed description sections (Deterministic functions, stochastic functions, algorithms) must be filled in.

This section will contain the description of algorithms in a step-by-step fashion, preferably using pseudocode if applicable.

The term "pseudocode" is used here to describe an English-like language that articulates the steps carried out in an algorithm. It allows the author to focus on the logic of the algorithm without being distracted by details of any given programming language's syntax. For the sake of completeness and consistency, pseudocode should follow a general syntax convention (see the appendix). Pseudocode can be augmented with natural language where convenient.

In case of distributed controllers, i.e. multiple processes communicating to achieve an overall objective, sequence diagrams of the interaction between the individual processes must be provided in the "diagrams" section, in addition to the detailed descriptions of each of the individual processes here.…

12. Outputs

This section is mandatory.

This section will describe all data output produced by the documented function or algorithm, insofar as the output is relevant to the interaction between the function or algorithm and the rest if the system. This covers mostly dynamic output data such as setpoints or events. Each output data element should describe the format of the output data in sufficient detail for downstream algorithms to be easily interfaced. This should as a minimum include the data type (e.g. floating point, integer, character string), units (e.g. degrees, percentages), valid ranges (e.g. 0 – infinity; -1.0 – +1.0 inclusive), the expected update rate of the data (e.g. static; 10Hz; daily) and a short description of the information the variable contains (e.g. "Power setpoint in kW").

All data element names should be self-descriptive (i.e. the name should describe what the data element contains). Widely recognized symbols from textbooks and professional literature are acceptable.

Name	Optimal tap position
Type	Scalar
Unit	none
Range	Defined by user (in this model -3 /+3)
Expected update rate	20 seconds
Description	Contains the output of the optimal tap position of the transformer. To be sent to the power system

Name	Optimal step position
Type	Scalar
Unit	none
Range	Defined by user (in this model 0 - 3)
Expected update rate	20 seconds
Description	Contains the output of the optimal step position of the shunt element (capacitor). To be sent to the power system

Name	Optimal reactive power setpoint of the DER
Type	Vector of floating values. Dimension of the vector equal to the amount of controllable DER in the power system.
Unit	MVAr
Range	Depending on the active power input of each of the sources, the equivalent of a power factor set by the user. In this benchmark it is set to 0.95.
Expected update rate	20 seconds
Description	Contains the output of the optimal step position of the shunt element (capacitor). To be sent to the power system

13. References

This section is mandatory.

This section shall provide all the references which appear in this document. They need to be in one of the standard citation formats like IEEE, ACM, Harvard, …

A. Pseudocode Constructs

This appendix contains examples of pseudocode constructs to be used in describing a scientific algorithm. The convention is for keywords and commands to appear in uppercase, which variables and procedure names are either lowercase or have the first letter capitalized.

A.1 Algorithm Construct

The format of an algorithm construct shall be as follows:

```
BEGIN ALGORITHM (algorithm-name)
    Pseudocode statement(s)
    …
    Pseudocode statement(s)
END ALGORITHM (algorithm-name)
```

Example:

```
BEGIN ALGORITHM (Storm Segments)
    COMPUTE (Storm Segment)
END ALGORITHM (Storm Segments)
```

A.2 WHILE Construct

The second DO construct is DO WHILE. The format shall be as follows:

```
DO WHILE (predicate)
    Pseudocode statement(s)
END DO
```

The DO WHILE construct represents a loop of repeated consideration of the pseudocode statement while the value of the predicate is true.

Example:

```
DO WHILE (Reflectivity is above threshold)
    Add the current SAMPLE VOLUME to current segment.
    Update end SAMPLE VOLUME number.
    COMPUTE (maximum Reflectivity Factor (dbze))
    WRITE (maximum Reflectivity Factor (dbze))
END DO
```

A.3 DO UNTIL Construct

The third DO construct is DO UNTIL. The format shall be as follows:

```
DO UNTIL (predicate)
    Pseudocode statement(s)
END DO
```

This construct represents a loop of repeated consideration of Pseudocode statement until the value of the predicate is true. Pseudocode statement is considered at least once even if the predicate is true at the onset.

Example:

```
DO UNTIL (No more storms isolated)
    Correlate the cells along vertical.
```

```
        Find 3D characteristics of correlated cells.
        Add correlated cells as a new storm to the table of storms.
    END DO
```

A.4 IF-THEN-ELSE Construct

The IF-THEN-ELSE construct shall have the following format:

```
    IF (predicate) THEN
        Pseudocode statement(s)1
    ELSE <optional>
        Pseudocode statement(s)2
    END IF
```

If the predicate is true, Pseudocode statement(s)1 (the "THEN" portion) will execute. If the predicate is false, then Pseudocode statement(s)2 (the "ELSE" portion) will execute. Note that the "ELSE" portion of this construct is optional and may not be used in every instance.

Example:

```
    IF (REFLECTIVITY is greater than 35 dBZe) THEN
        Label current segment
        COMPUTE (maximum Reflectivity Factor (Reflectivity))
    ELSE
        Ignore current segment
    END IF
```

A.5 CASE Construct

The CASE construct shall have the following format:

```
    BEGIN CASE (variable-name)
        CASE (value-list-l)
            Pseudocode statement(s)1
        CASE (value-list-2)
            Pseudocode statement(s)2
            …
        CASE (value-list-n)
            Pseudocode statement(s)n
    END CASE
```

where "value-list-i" is a series of values "variable-name" can take. Pseudocode statement(s)i is performed if a variable-name has the same value as a member of value-list-i.

Example:

```
    BEGIN CASE (elapsed time in minutes)
        CASE (6, 12, 18, 24)
            Collect raw radar data
        CASE (7, 13, 19, 25)
            Preprocess raw data
        CASE (8, 14, 20, 26)
            Analyze preprocessed data
        CASE (9, 15, 21, 27)
```

```
                Track and forecast the storms
                COMPUTE (Storm Speed)
        END CASE
```

A.6 Logical Constructs

In creating pseudocode, it is often useful to use the logical operators AND, OR, and NOT. The format for using these shall be as follows:

- *(predicate1) AND (predicate2)*
- *(predicate1) OR (predicate2)*
- *NOT (predicate)*

Example:

```
IF (Velocity is greater than 10 knots) AND (Velocity is less than 20 knots) THEN
    Label current segment
ELSE
    Ignore current segment
END IF
```

A.7 I/O Constructs

The READ and WRITE constructs indicate when the algorithm will input or output data items and shall have the following formats:

```
READ (variable-name)
```

and:

```
WRITE (variable-name)
```

Examples:

```
READ (Maximum Reflectivity Factor (dbze))
```

and:

```
WRITE (Centroid Position)
```

A.8 COMPUTE Construct

The COMPUTE construct is a shorthand notation for a set of computations used to derive a value or set of values. The COMPUTE construct shall have the following format:

```
COMPUTE (variable-name)
```

This construct causes the value(s) for variable-name to be computed.

Example:

```
COMPUTE (Minimum Velocity (Doppler))
```

causes the algorithm to compute the minimum Doppler velocity....

A.9 EXIT Construct

The EXIT construct shall have the following format:

EXIT ALGORITHM (algorithm-name)

This key word forces an unconditional exit from the body of the algorithm, terminating its execution.