2025

MINT PCU

VERSION 2.0

EN

Table of Contents

1 Introduction.....5

1.1 General.....5

1.2 Architecture.....5

1.3 Supported hardware.....6

1.4 Supported software.....6

1.4.1 Code Creator.....6

1.4.2 Solarworkx.....6

1.5 Control.....6

1.5.1 Assets.....7

1.5.2 DataHub.....8

1.5.3 Injection Limiting.....8

1.5.4 Imbalance control.....8

2 Software.....9

2.1 Compatibility.....9

2.2 Control.....9

2.2.1 Master.....9

2.2.2 Slave.....10

2.3 Configuration.....10

2.3.1 Site.....10

2.3.2 Master/Slave.....11

2.3.3 RTU.....11

2.3.4 Collector.....12

2.3.5 Solar.....13

2.3.6 Wind.....13

2.3.7 CHP.....14

2.3.8 DataHub.....14

2.3.9 Final steps.....15

2.4 Input - Output.....16

2.5 Assets.....16

2.5.1 RTU.....16

2.5.2 Collector.....20

2.5.3 Solar.....21

2.5.4 Wind 21
2.5.5 CHP 22
2.6 Portal (option) 23
2.7 eHMI (option) 24
2.8 Debugging 25
3 Software Change 26
3.1 V3.0.0 26
3.2 V3.0.1 26
3.3 V3.0.2 26
3.4 V3.1.0 27

Table of Figures

Figure 1: PCU architecture 5
Figure 2: Control structure 7
Figure 3: Master controller 9
Figure 4: Slave controller........ 10
Figure 5: Site config method 10
Figure 6: Slave config method 11
Figure 7: RTU config method 12
Figure 8: Collector config method 12
Figure 9: Solar config method 13
Figure 10: Wind config method 13
Figure 11: CHP config method 14
Figure 12: DataHub config method 15
Figure 13: Ready and control enable method........ 15
Figure 14: Fluvius RTU Interface........ 17
Figure 15: ORES RTU Interface 18
Figure 16: ORES RTU Interface 19
Figure 17: Collector Interface 20
Figure 18: Solar Interface 21
Figure 19: Wind Interface........ 22
Figure 20: CHP Interface........ 23
Figure 21: DataHub method linking 24
Figure 22: eHMI 24

Table of Revisions

Date	Version	Description of Changes	Status	Author
30-08-2024	1.0	Initial release v3.0.0	Published	Anthony Nicasens
03-09-2024	1.1	Version changes v3.0.1	Published	Anthony Nicasens
18-09-2024	1.2	Version changes v3.0.2	Published	Anthony Nicasens
19-11-2024	1.3	Version changes v3.1.0	Draft	Anthony Nicasens
10-01-2025	2.0	New template	Release	Marco Cicarelli

1 Introduction

1.1 General

The PCU library (Power Control Unit) is made to easily integrate the control of power production assets (PV, Wind, CHP, ...) into a program, to ensure the demands of the power distribution company (Fluvius, ORES, Sibelga, ...) are met. This is done by using standardised code elements (Function blocks), in combination with methods to ensure maximum flexibility in coding and operational reliability. The PCU library has a set of standardised IO elements included, as well as standardised HMI elements for easy configuration and visualisation. For distributed power control (control via multiple electrical cabinets), multiple PLCs can be used in a master/slave configuration. To ensure system security, the PLCs in a master/slave solution can communicate over a secure TCP connection. An MQTT communication to the DataHub enables extended control options, like visualise all the power production in the Phoenix Contact MINT portal, or PCU control based on ENTSO-e day-ahead pricing.

1.2 Architecture

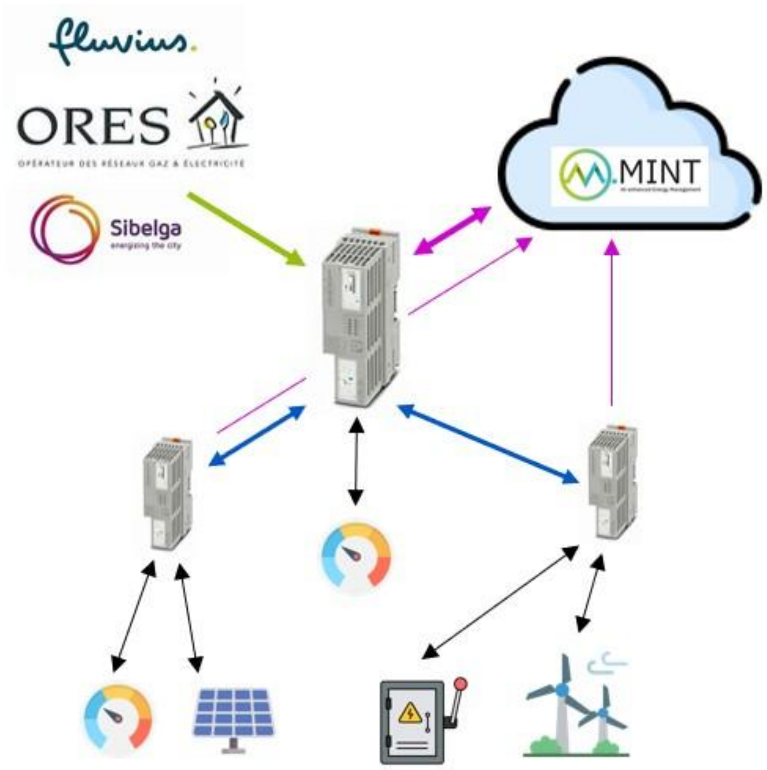


Figure 1: PCU architecture

The PCU architecture is made to be compatible with all kinds of devices and brands. If there is a way to communicate and control the assets, there is a way to make it work. The PLCs support up to 70 connections to assets in the field. The control software supports up to 20 slave controllers, so it is possible to control up to 120 assets in the field. The communication between the master and slave PLC is a secured TCP link (SSL). Each PLC can communicate to the DataHub on its own, this ensures reliability and a high bandwidth of data, it also allows for alarming in case one of the PLCs is down. Each PLC is equipped with IO cards that can read in real-time data from the field (statuses, serial and analog communication). The default way of communicating with field devices is Modbus TCP or RTU (depending on device compatibility), but other ways of communicating are also a possibility. The master PLC oversees the connection to the utility company and will also control the power of the site. In case of connection loss to the slave PLC, the slave will go back to a fallback scenario. This fallback scenario is putting all the power of the assets to 0 and in the case the asset keeps producing power after 30sec it will trigger a disconnect of the main switch. Each PLC sends its own measured data to the DataHub if the customer wants visualisation, in case of day-ahead/imbalance control the master PLC will request data from the DataHub.

1.3 Supported hardware

The PCU library is supported for all AXC F 2152 PLCs with firmware version 2024.0.6 and PLCnext engineer version 2024.0.4 (since PCU V3.1.0). Other PLCs might also be used but support will be limited. In case that there is a multi-cabinet solution (master/slave) multiple PLCs must be used. There can be no use of other Profinet devices.

1.4 Supported software

1.4.1 Code Creator

The PCU library is made to be universally compatible with libraries and other custom-made interface software. The Code Creator is an in-house tool that is used to generate code files for communication with a specific device. This device must have a device package, which has been created and verified (in the field). Once the code has been generated, the different measurement and command data points can be linked to the in- and outputs of the PCU interface function blocks.

1.4.2 Solarworkx

The Solarworkx library is a library created by Phoenix Contact Germany to communicate and control various PV inverters of different brands. This is a standardised software library utilising Modbus TCP or RTU to communicate with the inverters. This library requires an additional license that must be bought to function.

1.5 Control

The library makes use of PLCnext PLCs to control all the power production assets on-site. There can be multiple PLCs on-site in different cabinets who communicate their data to the master PLC. The master PLC oversees the entire site. In case of power failure or communication timeout to the master PLC the slave PLCs will give the command to the power production units to ramp their production down until communication to the master PLC is reestablished. The communication to the sub-PLCs runs over a TCP SSL link, so security of the communication is ensured. The master PLC controls the entire site power generation and makes sure the demands of the utility company are met. So will the PCU ramp the power down when commanded by the utility company when needed as well as provide reactive power control when requested.

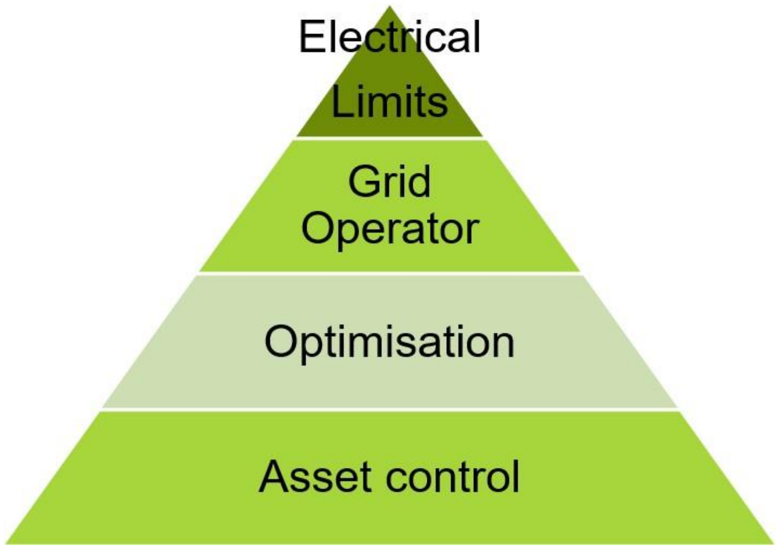


Figure 2: Control structure

The controller has a control hierarchy that is based on:

1. The electrical system limits of the site.
2. The demands of the utility company.
3. The power optimisation

- Load compensation
- Economic optimisation
- Reactive power compensation

4. Asset control limitations

This control structure is crucial to meet all the requirements of the grid operator and have a reliable system that can perform multiple roles. In the case the electrical limits are not respected, the installation will malfunction when there is high energy production. When the limits of the utility company are not respected, they will disconnect the installation from the grid. When an asset cannot perform in the way requested by the optimisation, the system will compensate this "loss" with another asset.

1.5.1 Assets

The PCU can control all kinds of assets and is compatible with all kinds of companies if there is a way to communicate and control the asset. The assets that are supported in the library (as of V3.0.x) are the following:

- RTU (communication with utility company)
    - Fluvius IEC104 (requires additional license)
    - ORES RS485 single RTU (RESA integrated)
    - ORES RS485 multi RTU (RESA integrated)
    - Sibelga (specific function can be created on request)
    - Other non-standardised RTU’s

- Collector (communication with measurement device)
- Production Assets
  - Photovoltaic inverters
  - Windmills
  - Combined Heat and Power
- Energy storage solutions (specific interface and control to be designed)

All the assets have their specific function. The RTU assets control the connection to the utility company who on its own controls the maximum power that may be delivered by the site. The collector asset is the asset responsible for measuring the effective power consumption/production on that fuse. There is at least one configuration of each asset necessary for the control to work.

1.5.2 DataHub

The optional DataHub feature of the PCU library can be useful when the data must be available over the API or to make visualisation or advice-based injection possible. To connect to the DataHub, a connection string should be requested to the Phoenix Contact Sales or Support person. Once verified that this option is included in the contract, a key will be given.

1.5.3 Injection Limiting

One of the functionalities of the control is limiting the injection of production of the site. This may be necessary to make sure you have no power injection on the grid or to have financial benefits of the system. The injection limiting is done by measuring the total power that is flowing on the injection point of the site. When the power flow is almost negative, the controller will start to reduce the power production on-site on the assets that are capable. This feature can be configured in the PLC, there is the option to always enable injection, never enable injection or control the injection based by day-ahead pricing. When choosing the option of day-ahead pricing there should be a connection to the DataHub.

1.5.4 Imbalance control

The PCU control can be extended by using imbalance prices to regulate the flow of energy to and from the grid, so the installation can be used to "generate" money. To optimise the imbalance control functionality of the installation, a battery is required to maximise the potential gains from the system. Working with imbalance control requires an imbalance party and has higher data costs due to the second data demand requested by the grid operator.

2 Software

2.1 Compatibility

The PCU library is built and designed for the ACX F2152 firmware version 2024.0.6 and PLCnext version 2024.0.4 Use of other firmware and software versions will not be supported. The PCU and DataHub task should be run on a cycle time of at least 25ms and a watchdog of 0 (disabled). The PLCnext base library service provider should be used to function properly, and a reboot of the system should be done to initiate the PLCnext base library. For more info refer to the PLCnext base library documentation.

The library is also compatible with the standard PCU cabinet, which has dedicated IO components for all kinds of assets and communication. For master/slave control the following cycle times are expected by the number of slaves:

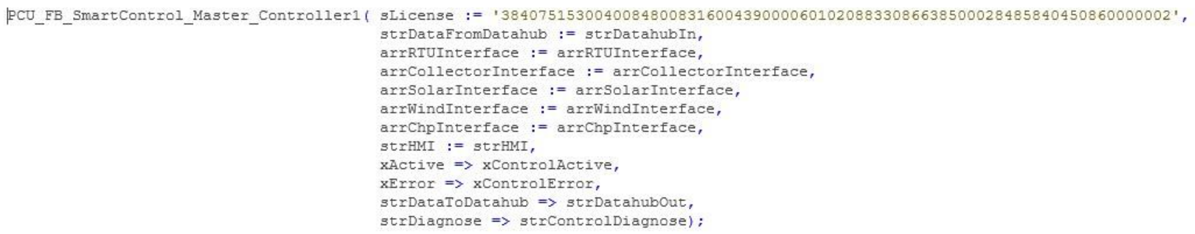
- 0-5 Slaves < 10ms
- 6-10 Slaves < 13ms
- 11-15 Slaves < 17ms
- 16-20 Slaves < 20ms

Note that these cycle times are for the master controller itself, without any other program task assigned under the same CPU core. For best practice, keep the PCU control and data interfaces on separate cores. When using MQTT connections to the DataHub the cycle time can also spike up to 4ms on top of the master control logic. When using more than 10 slaves it is also recommended to not have a lot of extra loads on the master controller (no extra data traffic or calculations) on the CPU.

2.2 Control

2.2.1 Master

The software allows control of many PLCs in a master/slave control functionality. There should always be at least one master configured in 1 PLC. This master functionality can be made by calling the function "PCU_FB_SmartControl_Master_Control". By calling this function you can call the configuration methods by which you can configure the entire site.



```
PCU_FB_SmartControl_Master_Controller1( sLicense := '384075153004008480083160043900006010208833086638500028485840450860000002',
                                        strDataFromDatahub := strDatahubIn,
                                        arrRTUInterface := arrRTUInterface,
                                        arrCollectorInterface := arrCollectorInterface,
                                        arrSolarInterface := arrSolarInterface,
                                        arrWindInterface := arrWindInterface,
                                        arrChpInterface := arrChpInterface,
                                        strHMI := strHMI,
                                        xActive => xControlActive,
                                        xError => xControlError,
                                        strDataToDatahub => strDatahubOut,
                                        strDiagnose => strControlDiagnose);
```

Figure 3: Master controller

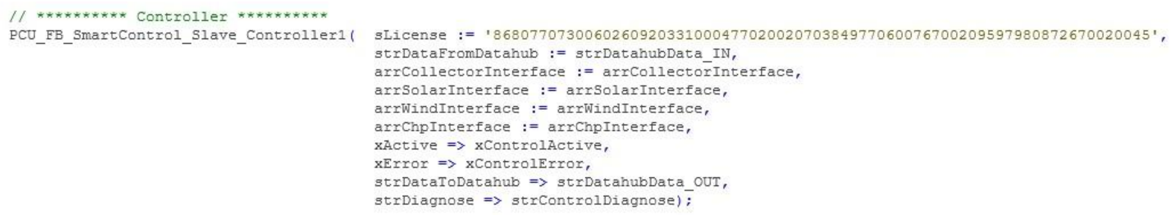
The master control function consists of several variables:

- License: The license of the master (specific to device).
- RTU interface: the RTU interface to the RTU function blocks.
- Collector interface: the collector interface to the collector function blocks.

- Solar interface: the solar interface to the solar function blocks
- Wind interface: the wind interface to the wind function blocks.
- CHP interface: the CHP interface to the CHP function blocks.
- DataHub data in: data from the DataHub.
- DataHub data out: data to the DataHub.
- HMI structure: linked to the HMI page (option).
- Active: if the control is active.
- Error: if there is an error active in the control.
- Diagnose structure: diagnose structure for error debugging.

2.2.2 Slave

The slave PLC has less inputs than the master PLC. There is no need for a configuration of the assets on the slave PLC, this is all managed on the master PLC. There can be up to 10 slaves in the project. The slave controller has only a limited amount of configuration (DataHub and connection to master). It also does not have the ability to connect to RTU and battery interfaces, because this is all managed in the master PLC.



```
// ********** Controller **********
PCU_FB_SmartControl_Slave_Controller1(  sLicense := '868077073006026092033100047702002070384977060076700209597980872670020045',
                                        strDataFromDatahub := strDatahubData_IN,
                                        arrCollectorInterface := arrCollectorInterface,
                                        arrSolarInterface := arrSolarInterface,
                                        arrWindInterface := arrWindInterface,
                                        arrChpInterface := arrChpInterface,
                                        xActive => xControlActive,
                                        xError => xControlError,
                                        strDataToDatahub => strDatahubData_OUT,
                                        strDiagnose => strControlDiagnose);
```

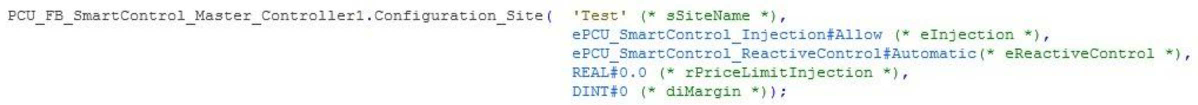
Figure 4: Slave controller

- License: The license of the slave (specific to device).
- Collector interface: the collector interface to the collector function blocks.
- Solar interface: the solar interface to the solar function blocks
- Wind interface: the wind interface to the wind function blocks.
- CHP interface: the CHP interface to the CHP function blocks.
- DataHub data in: data from the DataHub.
- DataHub data out: data to the DataHub.
- Active: if the control is active.
- Error: if there is an error active in the control.
- Connected: shows if the slave is connected to the master.
- Diagnose structure: diagnose structure for error debugging.

2.3 Configuration

2.3.1 Site

Every PCU has its site configuration, this site configuration has all the fundamental details of the site.



```
PCU_FB_SmartControl_Master_Controller1.Configuration_Site(  'Test' (* sSiteName *),
                                                            ePCU_SmartControl_Injection#Allow (* eInjection *),
                                                            ePCU_SmartControl_ReactiveControl#Automatic(* eReactiveControl *),
                                                            REAL#0.0 (* rPriceLimitInjection *),
                                                            DINT#0 (* diMargin *));
```

Figure 5: Site config method

In the site configuration method, you can set up the following details:

1. SiteName: give the site a name (for visualisation).
2. InjectionControl: control of the injection.

- Allow (Default): allow injection.
- Disallow: never allow injection on the grid.
- AdviceBased: allow injection based on day-ahead pricing.

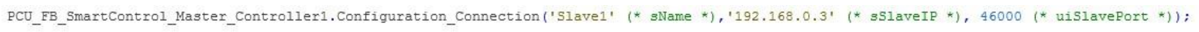
3. ReactiveControl:

- RuleBased (Default): only follow RTU command, otherwise power factor 1.0.
- Manual: Define a power factor with the "Input_ManualPowerFactor" to follow if RTU requests no reactive power.
- Automatic: The controller will try to compensate the reactive power consumption with production. So, if the active power on the grid collector is 0, the reactive power will also be 0.

4. PriceLimitInjection: when Advice-based injection is selected, you should write down the price threshold by which the PCU should stop injecting into the grid.
5. Margin: extra margin by which you should stop injecting into the grid when injection limiting is active.

2.3.2 Master/Slave

When making use of multiple PLCs, multiple slave connections should be configured to communicate to the slave PLCs.



```
PCU_FB_SmartControl_Master_Controller1.Configuration_Connection('Slave1' (* sName *),'192.168.0.3' (* sSlaveIP *), 46000 (* uiSlavePort *));
```

Figure 6: Slave config method

There are 3 parameters that must be configured to communicate to the slave/master PLC.

1. Slave name: Give the connection a name (for visualisation).
2. IP address: the address to which the slave/master PLC can be reached.
3. Port: the port by which the connection can connect over.

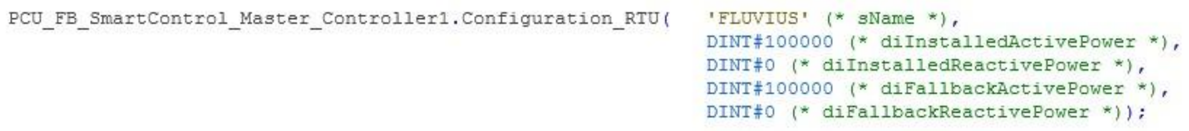
A maximum of 20 connections to slaves can be made. If this limit is exceeded in the configuration, an error will occur in the initialisation of the project.

Important note

The firewall (of routers) should be configured so the connection ports can reach each other. The SSL certificates should be uploaded in the PLC via the WBM, and the PLC time should be synchronised to the same NTP server. Refer to the MINT certificates documentation for more info.

2.3.3 RTU

Each project should have at least 1 RTU connection configured in the configuration. There are 3 possible options at this point to communicate with the utility company. These options will be explained in the Assets topic. The RTU configuration has 5 parameters to configure.



```
PCU_FB_SmartControl_Master_Controller1.Configuration_RTU(  'FLUVIUS' (* sName *),
                                                           DINT#100000 (* diInstalledActivePower *),
                                                           DINT#0 (* diInstalledReactivePower *),
                                                           DINT#100000 (* diFallbackActivePower *),
                                                           DINT#0 (* diFallbackReactivePower *));
```

Figure 7: RTU config method

1. Name: The name of the connection (for visualisation).
2. InstalledActivePower: The maximum active power that may be produced according to the contract with the utility company (this value should match the value in the utility company control).
3. InstalledReactivePower: The maximum reactive power that may be produced according to the contract with the utility company (this value should match the value in the utility company control).
4. FallBackActivePower: The fallback active power when the RTU watchdog has an alarm state.
5. FallBackReactivePower: The fallback reactive power when the RTU watchdog has an alarm state.

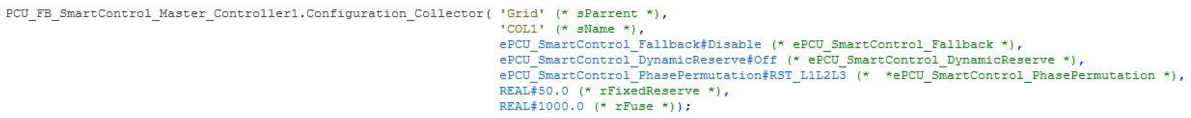
A maximum of 4 RTU control loops can be created in the PCU software, in most cases only one will be used.

Important note

ORES communication scales between negative 0-120%, use the 120% active power value in the configuration. For reactive power, the range is between -50 and +50% make sure to use these values in the configuration. Fluvius communication scales between -100% and +100% make sure the value of 100% matches the one configured by Fluvius. Typically for Fluvius you count all installed power of the site to get the 100% value.

2.3.4 Collector

A collector is the name for the measurement device that is actively measuring the power going through the electrical system. For each project at least one collector is recommended. The collector configuration method has 7 parameters that must be configured.



```
PCU_FB_SmartControl_Master_Controller1.Configuration_Collector( 'Grid' (* sParrent *),
                                                                'COL1' (* sName *),
                                                                ePCU_SmartControl_Fallback#Disable (* ePCU_SmartControl_Fallback *),
                                                                ePCU_SmartControl_DynamicReserve#Off (* ePCU_SmartControl_DynamicReserve *),
                                                                ePCU_SmartControl_PhasePermutation#RST_L1L2L3 (*  *ePCU_SmartControl_PhasePermutation *),
                                                                REAL#50.0 (* rFixedReserve *),
                                                                REAL#1000.0 (* rFuse *));
```

Figure 8: Collector config method

1. Parent: the collector that is electrically connected above this one. If this is the collector on the connection point to the utility company, 'Grid' should be used.
2. Name: name of the collector.
3. Fallback: fallback scenario in case that the collector is disconnected.
4. Dynamic reserve: not used (for future compatibility).
5. PhasePermutation: how the collector is electrically connected (refer to the electrical drawing/wiring).
6. FixedReserve: the amount of reserve on top of the fuse limit (10% as default).
7. Fuse: The fuse limit on the collector.

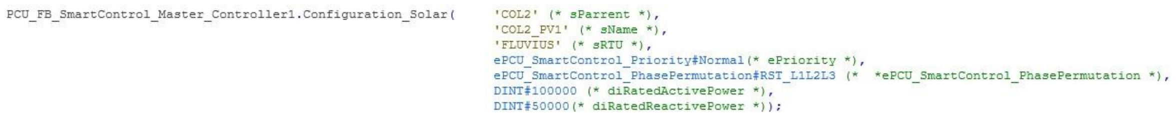
A maximum of 20 collectors can be used per slave, in total a site can have 200 collector measurements.

Important note

There should be at least one collector configured that has as parent 'Grid'. When configuring multiple collectors, the parent collector must already exist to get the configuration working.

2.3.5 Solar

The solar configuration is used to parameterise the values of the solar inverter/logger to control the inverter. The solar configuration has 7 parameters that must be configured to work properly.



```
PCU_FB_SmartControl_Master_Controller1.Configuration_Solar(      'COL2' (* sParrent *),
                                                                 'COL2_PV1' (* sName *),
                                                                 'FLUVIUS' (* sRTU *),
                                                                 ePCU_SmartControl_Priority#Normal(* ePriority *),
                                                                 ePCU_SmartControl_PhasePermutation#RST_L1L2L3 (*  *ePCU_SmartControl_PhasePermutation *),
                                                                 DINT#100000 (* diRatedActivePower *),
                                                                 DINT#50000(* diRatedReactivePower *));
```

Figure 9: Solar config method

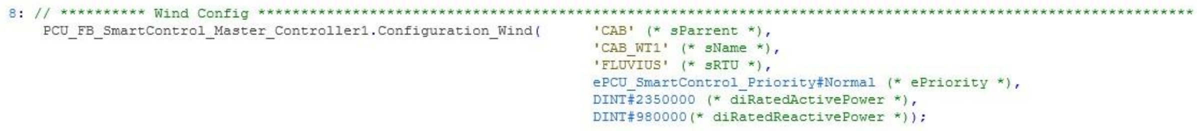
1. Parent: on what collector the solar is connected to.
2. Name: name of the solar.
3. RTU: what RTU control loop the solar is being controlled by.
4. Priority: what priority the asset has (green certificates high priority, default normal).
5. PhasePermutation: how the solar is electrically connected (refer to the electrical drawing/wiring).
6. RatedActivePower: the maximum active power that can be delivered by the invertor/logger (refer to the manual of the invertor/logger or the settings of the logger).
7. RatedReactivePower: the maximum reactive power that can be delivered by the invertor/logger (refer to the manual of the invertor/logger or the settings of the logger).

Important note

The parent collector and RTU unit must already exist for the configuration to be successful. If there is no parent collector or RTU, the field should be empty. At least an RTU or parent collector should be present in the config.

2.3.6 Wind

The wind configuration is used to parameterise the values of a wind controller. The wind configuration has 6 parameters that must be configured to work properly.



```
// ********** Wind Config ***************************************************************************************
    PCU_FB_SmartControl_Master_Controller1.Configuration_Wind(      'CAB' (* sParrent *),
                                                                    'CAB_WT1' (* sName *),
                                                                    'FLUVIUS' (* sRTU *),
                                                                    ePCU_SmartControl_Priority#Normal (* ePriority *),
                                                                    DINT#2350000 (* diRatedActivePower *),
                                                                    DINT#980000(* diRatedReactivePower *));
```

Figure 10: Wind config method

1. Parent: on what collector the wind is connected to.
2. Name: name of the wind.
3. RTU: what RTU control loop the wind is being controlled by.

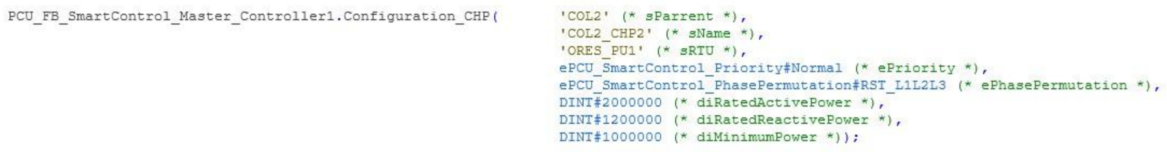
4. Priority: what priority the asset has (green certificates high priority, default normal).
5. RatedActivePower: the maximum active power that can be delivered by the invertor/logger (refer to the manual of the invertor/logger or the settings of the logger).
6. RatedReactivePower: the maximum reactive power that can be delivered by the invertor/logger (refer to the manual of the invertor/logger or the settings of the logger).

Important note

The parent collector and RTU unit must already exist for the configuration to be successful. If there is no parent collector or RTU, the field should be empty. At least an RTU or parent collector should be present in the config.

2.3.7 CHP

The CHP configuration is used to parameterise the values of a CHP unit. The CHP configuration has 8 parameters that must be configured to work properly.



```
PCU_FB_SmartControl_Master_Controller1.Configuration_CHP(        'COL2' (* sParrent *),
                                                                 'COL2_CHP2' (* sName *),
                                                                 'ORES_PU1' (* sRTU *),
                                                                 ePCU_SmartControl_Priority#Normal (* ePriority *),
                                                                 ePCU_SmartControl_PhasePermutation#RST_L1L2L3 (* ePhasePermutation *),
                                                                 DINT#2000000 (* diRatedActivePower *),
                                                                 DINT#1200000 (* diRatedReactivePower *),
                                                                 DINT#1000000 (* diMinimumPower *));
```

Figure 11: CHP config method

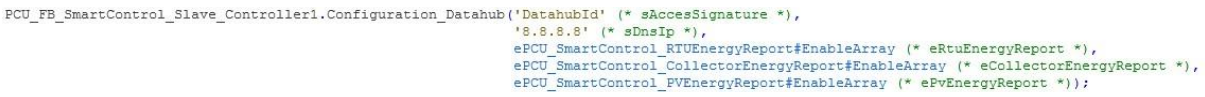
1. Parent: on what collector the CHP is connected to.
2. Name: name of the CHP.
3. RTU: what RTU control loop the CHP is being controlled by.
4. Priority: what priority the asset has (green certificates high priority, default normal).
5. PhasePermutation: how the CHP is electrically connected (refer to the electrical drawing/wiring).
6. RatedActivePower: the maximum active power that can be delivered by the invertor/logger (refer to the manual of the invertor/logger or the settings of the logger).
7. RatedReactivePower: the maximum reactive power that can be delivered by the invertor/logger (refer to the manual of the invertor/logger or the settings of the logger).
8. MinimumPower: the minimum power level that the CHP can be run on (typically between 70 and 50% of full power).

Important note

The parent collector and RTU unit must already exist for the configuration to be successful. If there is no parent collector or RTU, the field should be empty. At least an RTU or parent collector should be present in the config.

2.3.8 DataHub

When the PCU is an Advanced version, the DataHub must be configured. The DataHub connection allows to push data to the MQTT program instance. There are only a few parameters that must be configured to enable the connection to the DataHub.



```
PCU_FB_SmartControl_Slave_Controller1.Configuration_Datahub('DatahubId' (* sAccesSignature *),
                                                          '8.8.8.8' (* sDnsIp *),
                                                          ePCU_SmartControl_RTUEnergyReport#EnableArray (* eRtuEnergyReport *),
                                                          ePCU_SmartControl_CollectorEnergyReport#EnableArray (* eCollectorEnergyReport *),
                                                          ePCU_SmartControl_PVEnergyReport#EnableArray (* ePvEnergyReport *));
```

Figure 12: DataHub config method

1. AccesSignature: The specific PLC access signature to the DataHub.
2. DNS IP: the DNS IP address (default 8.8.8.8, change if necessary).
3. RTU energy report

- Disable: disables the reporting to the DataHub.
- Enable Individual: enables reporting (sends 1 message for each RTU asset).
- Enable Array: enables reporting (sends 1 message for all RTU assets).

4. Collector energy report

- Disable: disables the reporting to the DataHub.
- Enable Individual: enables reporting (sends 1 message for each collector asset).
- Enable Array: enables reporting (sends 1 message for all collector assets).

5. Solar energy report

- Disable: disables the reporting to the DataHub.
- Enable Individual: enables reporting (sends 1 message for each solar asset).
- Enable Array: enables reporting (sends 1 message for all solar assets).

6. Wind energy report

- Disable: disables the reporting to the DataHub.
- Enable Individual: enables reporting (sends 1 message for each wind asset).
- Enable Array: enables reporting (sends 1 message for all wind assets).

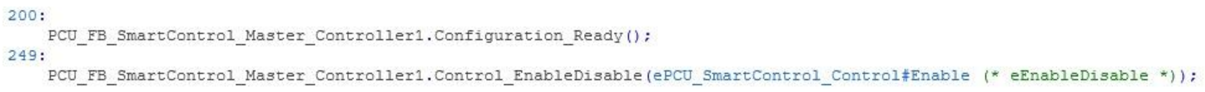
7. CHP energy report

- Disable: disables the reporting to the DataHub.
- Enable Individual: enables reporting (sends 1 message for each CHP asset).
- Enable Array: enables reporting (sends 1 message for all CHP assets).

When the DataHub functionality is used you should also call the "Input_Datahub" and "Output_Datahub" methods to connect the data to the MQTT program instance.

2.3.9 Final steps

For the configuration to be loaded in the methods, "Configuration_Ready" and "Control_Enable" should be called.



```
200:
    PCU_FB_SmartControl_Master_Controller1.Configuration_Ready();
249:
    PCU_FB_SmartControl_Master_Controller1.Control_EnableDisable(ePCU_SmartControl_Control#Enable (* eEnableDisable *));
```

Figure 13: Ready and control enable method

By doing this, the program will start to link all the assets internally. If an error occurs, refer to the diagnostic structure on the output of the "PCU_FB_SmartControl_Master_Control" or "PCU_FB_SmartControl_Slave_Control" function block. If everything is configured properly, the output xActive should become TRUE. If the "disable" parameter in the "Control_Enable" method is used, the PLC will put all solar assets to 100%, and all wind and CHP assets will be set to 0% and will not be controlled. The PLC can still control the switches.

2.4 Input - Output

In the PCU V3 library there are in- and output structures defined to easily integrate the communication protocols of the Belgian grid utility operators. These input and outputs are based on the IO standardisation that is used in the PCU cabinet that can be delivered by Phoenix Contact Belgium.

There are 4 standardised IO functions that can be used in the program.

1. PCU_FC_MasterInputs: Structuring the Master cabinet inputs.
2. PCU_FC_MasterOutputs: Structuring the Master cabinet Outputs.
3. PCU_FC_SlaveInputs: Structuring the Slave cabinet inputs.
4. PCU_FC_SlaveOutputs: Structuring the Slave cabinet Outputs.

By reading in the input word of the input card and coupling it to the input of the function, the structure of the IO will be mapped automatically. This is also the case for the outputs, but the other way around, couple the structure to the function and the return will give the word output that can be coupled to the output card.

If the standardised IO is not used, you can make the coupling to the In/Out structure yourself in order to have a structured program (not mandatory, but preferred).

To visualise the IO on the HMI, you can make use of the "PCU_FC_Status" function to assign names to the input/output word and show them on the HMI. This function and its HMI object ("Status") are dynamic, meaning, when string inputs are not used, they will not be shown on the HMI.

2.5 Assets

2.5.1 RTU

2.5.1.1 General

The RTU asset in the PCU library is the asset that is responsible for the power control of the site. There can be up to 4 RTU power control loops (and a minimum of one) in the control logic. For each control loop you must select one of the following RTU interface function blocks:

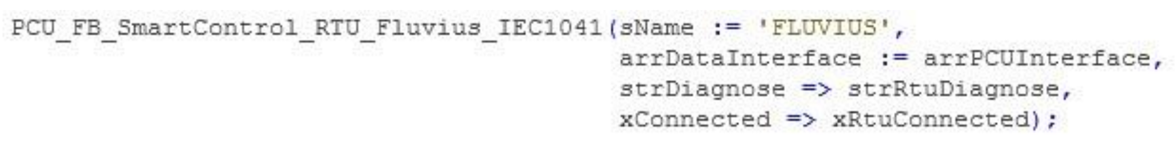
- PCU_FB_SmartControl_RTU_Fluvius_IEC104: For communication with Fluvius over IEC104 protocol.
- PCU_FB_SmartControl_RTU_ORES_RS485_Single: For communication with ORES over RS485 protocol for a single control loop.
- PCU_FB_SmartControl_RTU_ORES_RS485_Multi: For communication with ORES over RS485 protocol for multiple control loops.
- PCU_FB_SmartControl_RTU_Other: To control the control loop when no RTU is present on-site or when a modification of the RTU signal is done or when the specific RTU interface is not made for the PCU library.

In these RTU interface function blocks all the necessary communication parameters and functions are integrated so it can easily be integrated in the control function. Just drag and drop the right interface in the program, connect the data interface to the control logic, assign the configuration in the configuration of the control logic and it is done. Each RTU function block has some basic methods, as well as some specific methods that can be used for controlling the control loop. The basic methods are:

- Input_DisconnectCmd: When used and the input is TRUE, this will send a disconnect command to all production units, when they do not disconnect their main disconnect, the refuse switch will also disconnect in the time that is configured for the specific production unit.
- Input_StatusGridRelay: When used and the input is FALSE, this will trigger the control logic to ramp down the power to 0.
- Output_InstallationOk: When the return is TRUE, all communication to the production units of that control loop.
- Output_DisconnectClosed: When the return is TRUE, at least one of the main disconnect switches of the production units is closed.
- Output_DisconnectOpen: When the return is TRUE, at least one of the main disconnect switches of the production units is opened.
- Output_RefusesClosed: When the return is TRUE, at least one of the main back-ups. (refuse) switches of the production units is closed.
- Output_RefusedOpen: When the return is TRUE, at least one of the main back-ups. (refuse) switches of the production units is opened.

2.5.1.2 Fluvius IEC104

Fluvius communicates with the PLC using the IEC104 protocol. This protocol is standardised in different European countries to perform telecommunication with the grid operator. To work with this protocol, an extra license must be ordered. The function block itself in the PCU library has all parameters assigned internally to perform the necessary control. Fluvius itself communicates its setpoints in a range between -100 and +100%. If the configuration is done properly, it will be scaled to the right values internally. On top of this, there is a reactive control by which Fluvius can determine whether they want control of the reactive power or if this can be chosen locally. All of this is managed inside the function block of the RTU. The interface block has some specific Fluvius methods that can be used to control the installation.



```
PCU_FB_SmartControl_RTU_Fluvius_IEC1041(sName := 'FLUVIUS',
                                        arrDataInterface := arrPCUInterface,
                                        strDiagnose => strRtuDiagnose,
                                        xConnected => xRtuConnected);
```

Figure 14: Fluvius RTU Interface

- Output_GetActivePowerPU: Returns the active power production value of the selected power production unit (PU) that is being received by Fluvius (this must be set up by Fluvius in their system).

- Output_GetActivePowerPU: Returns the reactive power production value of the selected power production unit (PU) that is being received by Fluvius (this must be set up by Fluvius in their system).
- Output_GetVoltages: Returns the voltage of the selected phase that is being received by Fluvius (this must be set up by Fluvius in their system).

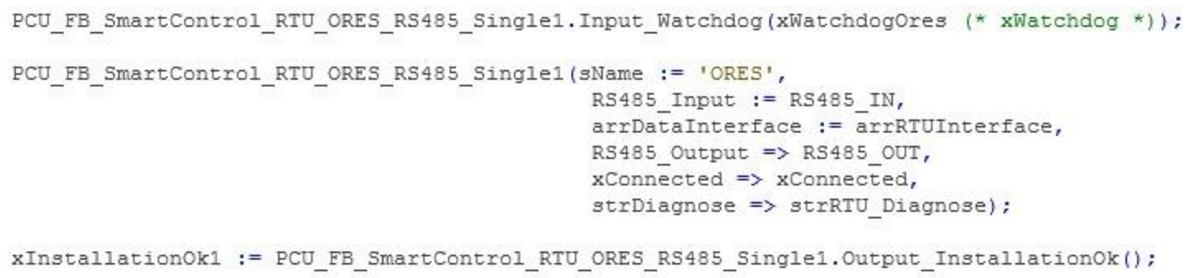
When connection to Fluvius is achieved, the output xConnected on the interface will turn TRUE. If an error occurs, look at the diagnose structure that is provided to see what the fault is.

Important note

One of the PLC ethernet adapters must be set up with an IP address of 172.31.254.102 and subnet mask 255.255.255.252 with no default gateway. In the case that the PLC firewall is configured, make sure that port 2404 is inside the configuration. Use of an ethernet extension card is recommended.

2.5.1.3 ORES RS485 Single

ORES makes use of serial communication to communicate the setpoints to the PLC, this function block can only receive one setpoint of one control loop with ORES. To make sure all communication is internal, the necessary scaling is done to work with the control logic.



```
PCU_FB_SmartControl_RTU_ORES_RS485_Single1.Input_Watchdog(xWatchdogOres (* xWatchdog *));

PCU_FB_SmartControl_RTU_ORES_RS485_Single1(sName := 'ORES',
                                            RS485_Input := RS485_IN,
                                            arrDataInterface := arrRTUInterface,
                                            RS485_Output => RS485_OUT,
                                            xConnected => xConnected,
                                            strDiagnose => strRTU_Diagnose);

xInstallationOk1 := PCU_FB_SmartControl_RTU_ORES_RS485_Single1.Output_InstallationOk();
```

Figure 15: ORES RTU Interface

Make sure all parameters in the configuration match the 120% value of ORES. ORES sends data setpoints in MW, by which the scaling to Watts is done internally. This also happens the other way around, so ORES receives our production figures. The O-One logic that ORES uses for controlling the installation is also included in the interface function block. To modulate power, the watchdog of ORES must be TRUE, otherwise the fallback value (in the configuration) will be used. ORES has some specific methods that can be used:

- Output_UndelayedActivePower: Returns the active power setpoint from ORES without the O-One delay.
- Output_UndelayedReactivePower: Returns the reactive power setpoint from ORES without the O-One delay.

When communication with ORES is established, the output xConnected will be TRUE. If an error occurs, check the diagnose output structure.

Important note

Serial card "AXL SE RS485" is supported, "AXL F RSUNI" could lead to issues.

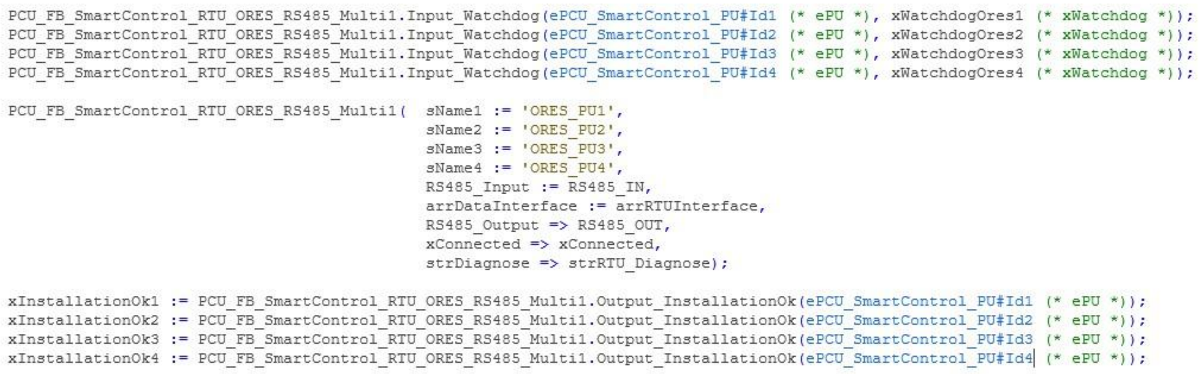
Parameters in the serial card must be set to RS485, Baud: 38400, parity bits: 8, parity: even, 1 stop bit.

Make use of the Phoenix Contact standardised serial interface parameters for ORES to make sure that all data is sent and received on the right registers, the setpoints coming from ORES are standard negative as well as the feedback to them.

It is preferred that the methods "Input_xxxxxx" are called before the block call and "Output_xxxx" after the block call to have the most recent process data in the interfaces.

2.5.1.4 ORES RS485 Multi

ORES makes use of serial communication to communicate the setpoints to the PLC, this function block can send and receive the 4 setpoints of all control loops at once in the case that multiple control loops are used. To make sure all communication is internal, the necessary scaling is done to work with the control logic.



```
PCU_FB_SmartControl_RTU_ORES_RS485_Multi1.Input_Watchdog(ePCU_SmartControl_PU#Id1 (* ePU *), xWatchdogOres1 (* xWatchdog *));
PCU_FB_SmartControl_RTU_ORES_RS485_Multi1.Input_Watchdog(ePCU_SmartControl_PU#Id2 (* ePU *), xWatchdogOres2 (* xWatchdog *));
PCU_FB_SmartControl_RTU_ORES_RS485_Multi1.Input_Watchdog(ePCU_SmartControl_PU#Id3 (* ePU *), xWatchdogOres3 (* xWatchdog *));
PCU_FB_SmartControl_RTU_ORES_RS485_Multi1.Input_Watchdog(ePCU_SmartControl_PU#Id4 (* ePU *), xWatchdogOres4 (* xWatchdog *));

PCU_FB_SmartControl_RTU_ORES_RS485_Multi1(  sName1 := 'ORES_PU1',
                                            sName2 := 'ORES_PU2',
                                            sName3 := 'ORES_PU3',
                                            sName4 := 'ORES_PU4',
                                            RS485_Input := RS485_IN,
                                            arrDataInterface := arrRTUInterface,
                                            RS485_Output => RS485_OUT,
                                            xConnected => xConnected,
                                            strDiagnose => strRTU_Diagnose);

xInstallationOk1 := PCU_FB_SmartControl_RTU_ORES_RS485_Multi1.Output_InstallationOk(ePCU_SmartControl_PU#Id1 (* ePU *));
xInstallationOk2 := PCU_FB_SmartControl_RTU_ORES_RS485_Multi1.Output_InstallationOk(ePCU_SmartControl_PU#Id2 (* ePU *));
xInstallationOk3 := PCU_FB_SmartControl_RTU_ORES_RS485_Multi1.Output_InstallationOk(ePCU_SmartControl_PU#Id3 (* ePU *));
xInstallationOk4 := PCU_FB_SmartControl_RTU_ORES_RS485_Multi1.Output_InstallationOk(ePCU_SmartControl_PU#Id4 (* ePU *));
```

Figure 16: ORES RTU Interface

Make sure all parameters in the configuration match the 120% value of ORES. ORES sends data setpoints in MW, by which the scaling to Watts is done internally. This also happens the other way around, so ORES receives our production figures. The O-One logic that ORES uses for controlling the installation is also included in the interface function block. To modulate power, the watchdog of ORES must be TRUE, otherwise the fallback value (in the configuration) will be used. ORES has some specific methods that can be used:

- Output_UndelayedActivePower: Returns the active power setpoint from ORES without the O-One delay.
- Output_UndelayedReactivePower: Returns the reactive power setpoint from ORES without the O-One delay.

When communication with ORES is established the output xConnected will be TRUE. If an error occurs, check the diagnose output structure. You must assign the 4 names that are also used in the configuration of the smart control logic. Each control loop has its own production unit number in the method of the function block that corresponds with the name of that production unit (sName1 corresponds with ePCU_SmartControl_PU1, sName2 with the enumeration PU2, ...).

Important note

Serial card "AXL SE RS485" is supported, "AXL F RSUNI" could lead to issues.

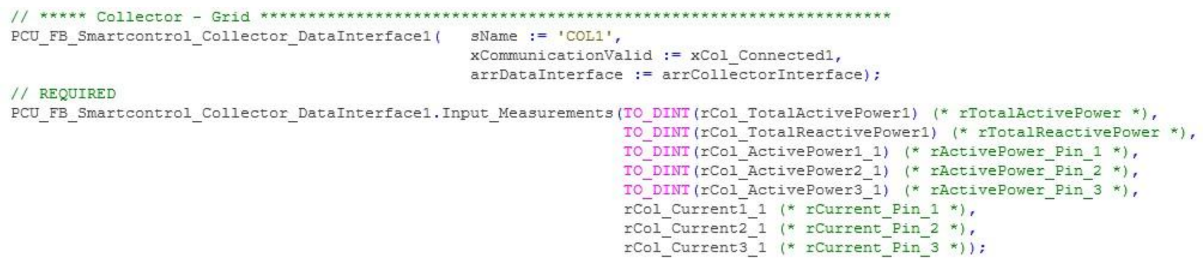
Parameters in the serial card must be set to RS485, Baud: 38400, parity bits: 8, parity: even, 1 stop bit.

Make use of the Phoenix Contact standardised serial interface parameters for ORES to make sure that all data is sent and received on the right registers, the setpoints coming from ORES are standard negative as well as the feedback to them.

It is preferred that the methods "Input_xxxxxx" are called before the block call and "Output_xxxx" after the block call to have the most recent process data in the interfaces.

2.5.2 Collector

The collector interface is made to control the site power production and perform load balancing on the electrical circuits. The collector interface needs a name that is linked to the configuration. The status of the collector must also be known. If the collector is connected to the PLC, the input xConnected must be TRUE.



```
// ***** Collector - Grid ********************************************************************
PCU_FB_Smartcontrol_Collector_DataInterface1(   sName := 'COL1',
                                                xCommunicationValid := xCol_Connected1,
                                                arrDataInterface := arrCollectorInterface);
// REQUIRED
PCU_FB_Smartcontrol_Collector_DataInterface1.Input_Measurements(TO_DINT(rCol_TotalActivePower1) (* rTotalActivePower *),
                                                                TO_DINT(rCol_TotalReactivePower1) (* rTotalReactivePower *),
                                                                TO_DINT(rCol_ActivePower1_1) (* rActivePower_Pin_1 *),
                                                                TO_DINT(rCol_ActivePower2_1) (* rActivePower_Pin_2 *),
                                                                TO_DINT(rCol_ActivePower3_1) (* rActivePower_Pin_3 *),
                                                                rCol_Current1_1 (* rCurrent_Pin_1 *),
                                                                rCol_Current2_1 (* rCurrent_Pin_2 *),
                                                                rCol_Current3_1 (* rCurrent_Pin_3 *));
```

Figure 17: Collector Interface

Each collector can be configured so the available data can be read in using the methods:

- Input_Measurements: (Requirement for control) This method is the minimum requirement for the control system to work.
- Input_ReactivePower: (Recommended for visualisation) Adds reactive power data to the API.
- Input_TotalEnergy: (Recommended for visualisation) Adds energy data to the API.
- Input_Energy: Adds phase energy values to the API.
- Input_Voltage: Adds phase voltage values to the API.
- Input_ConductorVoltage: Adds line voltage data to the API.
- Input_Powerfactor: Adds phase power factor values to the API.
- Input_TotalPowerfactor: Adds power factor values to the API.
- Input_Frequency: Adds frequency values to the API.

A maximum of 20 collector interface function blocks can be used per PLC. If this limit is exceeded, the values will not be used.

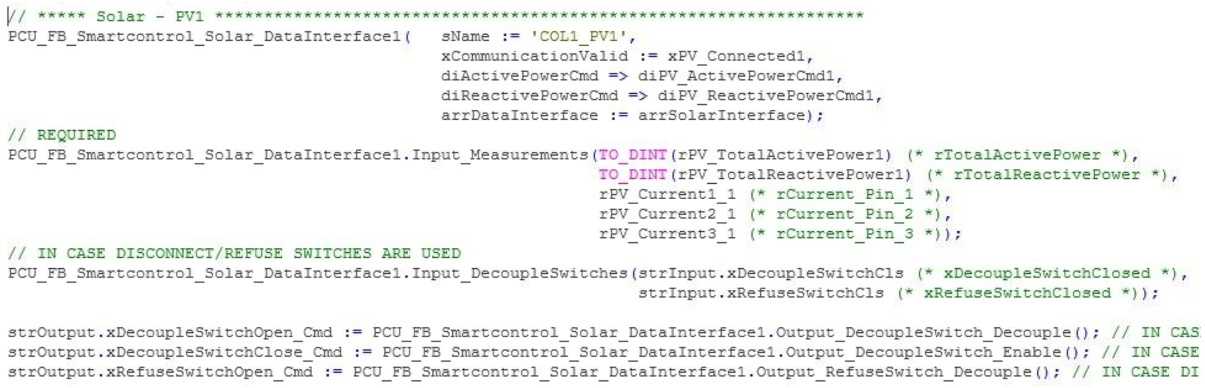
Important note

At least 1 collector must be configured, if there are no physical measurement devices in the field there must be a virtual collector. It is preferred to have 1 collector (measurement device) per slave cabinet.

2.5.3 Solar

The solar interface is responsible for interfacing with the solar inverter/logger and the control logic. In this interface, all data that is available must be coupled and the outputs must be linked to the control logic of the inverter. The solar interface needs a name that is linked to the one in the configuration to work properly. The connection status to the inverter/logger must also be linked to the xConnected input and the solar interface to the one on the control logic. Additional data can be coupled via the methods:

- Input_Measurements: (Requirement for control) This method is the minimum requirement for the control system to work.



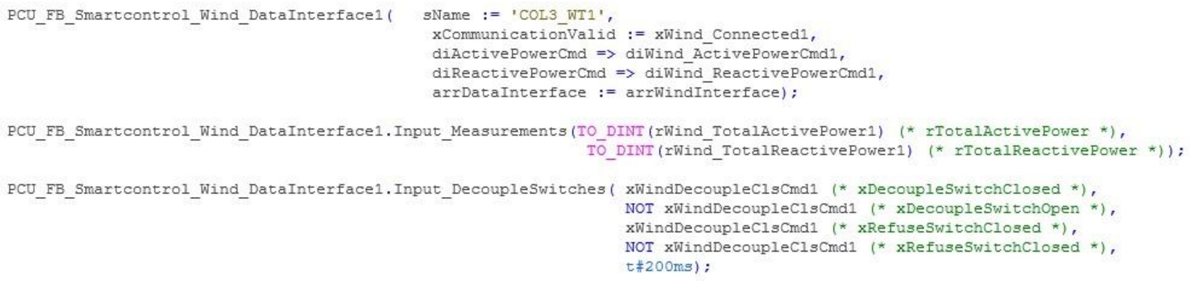
```
// ***** Solar - PV1 ******************************************************************
PCU_FB_Smartcontrol_Solar_DataInterface1(   sName := 'COL1_PV1',
                                            xCommunicationValid := xPV_Connected1,
                                            diActivePowerCmd => diPV_ActivePowerCmd1,
                                            diReactivePowerCmd => diPV_ReactivePowerCmd1,
                                            arrDataInterface := arrSolarInterface);
// REQUIRED
PCU_FB_Smartcontrol_Solar_DataInterface1.Input_Measurements(TO_DINT(rPV_TotalActivePower1) (* rTotalActivePower *),
                                                            TO_DINT(rPV_TotalReactivePower1) (* rTotalReactivePower *),
                                                            rPV_Current1_1 (* rCurrent_Pin_1 *),
                                                            rPV_Current2_1 (* rCurrent_Pin_2 *),
                                                            rPV_Current3_1 (* rCurrent_Pin_3 *));
// IN CASE DISCONNECT/REFUSE SWITCHES ARE USED
PCU_FB_Smartcontrol_Solar_DataInterface1.Input_DecoupleSwitches(strInput.xDecoupleSwitchCls (* xDecoupleSwitchClosed *),
                                                                strInput.xRefuseSwitchCls (* xRefuseSwitchClosed *));

strOutput.xDecoupleSwitchOpen_Cmd := PCU_FB_Smartcontrol_Solar_DataInterface1.Output_DecoupleSwitch_Decouple(); // IN CAS
strOutput.xDecoupleSwitchClose_Cmd := PCU_FB_Smartcontrol_Solar_DataInterface1.Output_DecoupleSwitch_Enable(); // IN CASE
strOutput.xRefuseSwitchOpen_Cmd := PCU_FB_Smartcontrol_Solar_DataInterface1.Output_RefuseSwitch_Decouple(); // IN CASE DI
```

Figure 18: Solar Interface

- Input_Activepower: Adds phase active power values to the API.
- Input_Reactivepower: Adds phase reactive power values to the API.
- Input_TotalEnergy: (Recommended for visualisation) Adds energy data to the API.
- Input_Energy: Adds phase energy values to the API.
- Input_Voltage: Adds phase voltage values to the API.
- Input_Powerfactor: Adds phase power factor values to the API.
- Input_TotalPowerfactor: Adds power factor values to the API.
- Input_DecoupleSwitches: Adds the ability to read in the status of the main and backup disconnect switches (when used and status is open, setpoint will be set to 0).
- Output_DecoupleSwitch_Decouple: Adds the ability to disconnect the solar from the grid when the command is given by the RTU ("Input_DecoupleSwitches" must be used). When command is given, this output will give a 1sec pulse.
- Output_DecoupleSwitch_Enable: Adds the ability to reconnect the solar to the grid ("Input_DecoupleSwitches" must be used). When command is given, this output will be continuous.
- Output_RefuseSwitch_Decouple: Adds the ability to disconnect the solar from the grid with the backup switch in case of failure to open the main in 200ms ("Input_DecoupleSwitches" must be used). When the command is given, this output will give a 1sec pulse.

2.5.4 Wind

The wind interface is responsible for interfacing with the wind controller and the control logic. In this interface, all data that is available must be coupled and the outputs must be linked to the control logic of the wind controller. The wind interface needs a name that is linked to the one in the configuration to work properly.



```
PCU_FB_Smartcontrol_Wind_DataInterface1(   sName := 'COL3_WT1',
                                            xCommunicationValid := xWind_Connected1,
                                            diActivePowerCmd => diWind_ActivePowerCmd1,
                                            diReactivePowerCmd => diWind_ReactivePowerCmd1,
                                            arrDataInterface := arrWindInterface);

PCU_FB_Smartcontrol_Wind_DataInterface1.Input_Measurements(TO_DINT(rWind_TotalActivePower1) (* rTotalActivePower *),
                                                            TO_DINT(rWind_TotalReactivePower1) (* rTotalReactivePower *));

PCU_FB_Smartcontrol_Wind_DataInterface1.Input_DecoupleSwitches( xWindDecoupleClsCmd1 (* xDecoupleSwitchClosed *),
                                                                NOT xWindDecoupleClsCmd1 (* xDecoupleSwitchOpen *),
                                                                xWindDecoupleClsCmd1 (* xRefuseSwitchClosed *),
                                                                NOT xWindDecoupleClsCmd1 (* xRefuseSwitchClosed *),
                                                                t#200ms);
```

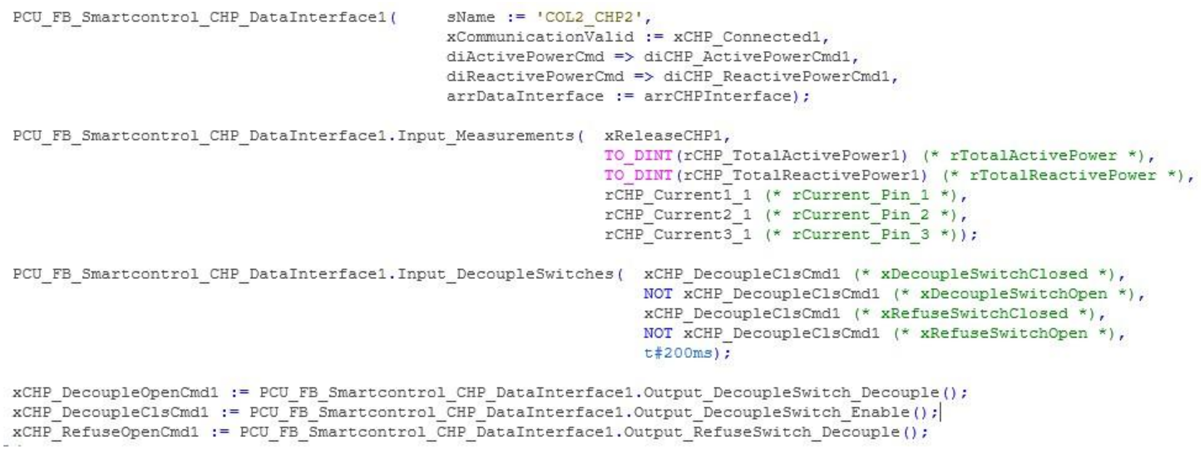
Figure 19: Wind Interface

The connection status to the controller must also be linked to the xConnected input and the wind interface to the one on the control logic. Additional data can be coupled via the methods:

- Input_Measurements: (Requirement for control) This method is the minimum requirement for the control system to work.
- Input_Activepower: Adds phase active power values to the API.
- Input_Reactivepower: Adds phase reactive power values to the API.
- Input_TotalEnergy: (Recommended for visualisation) Adds energy data to the API.
- Input_Energy: Adds phase energy values to the API.
- Input_Voltage: Adds phase voltage values to the API.
- Input_Currents: Adds currents values to the API.
- Input_Powerfactor: Adds phase power factor values to the API.
- Input_TotalPowerfactor: Adds power factor values to the API.
- Control_Release: Adds the ability to shut down the unit when input is FALSE.
- Input_DecoupleSwitches: Adds the ability to read in the status of the main and backup disconnect switches (when used and status is open, setpoint will be set to 0).
- Output_DecoupleSwitch_Decouple: Adds the ability to disconnect the solar from the grid when command is given by the RTU ("Input_DecoupleSwitches" must be used). When command is given, this output will give a 1sec pulse.
- Output_DecoupleSwitch_Enable: Adds the ability to reconnect the wind to the grid ("Input_DecoupleSwitches" must be used). When command is given, this output will be continuous.
- Output_RefuseSwitch_Decouple: Adds the ability to disconnect the wind from the grid with the backup switch in case of failure to open the main in 200ms ("Input_DecoupleSwitches" must be used). When command is given, this output will give a 1sec pulse.

2.5.5 CHP

The CHP interface is responsible for interfacing with the CHP unit controller and the control logic. In this interface, all data that is available must be coupled, and the outputs must be linked to the control logic of the unit. The CHP interface needs a name that is linked to the one in the configuration to work properly.



```
PCU_FB_Smartcontrol_CHP_DataInterface1(      sName := 'COL2_CHP2',
                                             xCommunicationValid := xCHP_Connected1,
                                             diActivePowerCmd => diCHP_ActivePowerCmd1,
                                             diReactivePowerCmd => diCHP_ReactivePowerCmd1,
                                             arrDataInterface := arrCHPInterface);

PCU_FB_Smartcontrol_CHP_DataInterface1.Input_Measurements(  xReleaseCHP1,
                                                            TO_DINT(rCHP_TotalActivePower1) (* rTotalActivePower *),
                                                            TO_DINT(rCHP_TotalReactivePower1) (* rTotalReactivePower *),
                                                            rCHP_Current1_1 (* rCurrent_Pin_1 *),
                                                            rCHP_Current2_1 (* rCurrent_Pin_2 *),
                                                            rCHP_Current3_1 (* rCurrent_Pin_3 *));

PCU_FB_Smartcontrol_CHP_DataInterface1.Input_DecoupleSwitches(  xCHP_DecoupleClsCmd1 (* xDecoupleSwitchClosed *),
                                                                NOT xCHP_DecoupleClsCmd1 (* xDecoupleSwitchOpen *),
                                                                xCHP_DecoupleClsCmd1 (* xRefuseSwitchClosed *),
                                                                NOT xCHP_DecoupleClsCmd1 (* xRefuseSwitchOpen *),
                                                                t#200ms);

xCHP_DecoupleOpenCmd1 := PCU_FB_Smartcontrol_CHP_DataInterface1.Output_DecoupleSwitch_Decouple();
xCHP_DecoupleClsCmd1 := PCU_FB_Smartcontrol_CHP_DataInterface1.Output_DecoupleSwitch_Enable();
xCHP_RefuseOpenCmd1 := PCU_FB_Smartcontrol_CHP_DataInterface1.Output_RefuseSwitch_Decouple();
```

Figure 20: CHP Interface

The connection status to the controller must also be linked to the xConnected input and the CHP interface to the one on the control logic. Additional data can be coupled via the methods:

- Input_Measurements: (Requirement for control) This method is the minimum requirement for the control system to work.
- Input_Activepower: Adds phase active power values to the API.
- Input_Reactivepower: Adds phase reactive power values to the API.
- Input_TotalEnergy: (Recommended for visualisation) Adds energy data to the API.
- Input_Energy: Adds phase energy values to the API.
- Input_Voltage: Adds phase voltage values to the API.
- Input_Powerfactor: Adds phase power factor values to the API.
- Input_TotalPowerfactor: Adds power factor values to the API.
- Input_DecoupleSwitches: Adds the ability to read in the status of the main and backup disconnect switches (when used and status is open, setpoint will be set to 0).
- Output_DecoupleSwitch_Decouple: Adds the ability to disconnect the solar from the grid when command is given by the RTU ("Input_DecoupleSwitches" must be used). When command is given, this output will give a 1sec pulse.
- Output_DecoupleSwitch_Enable: Adds the ability to reconnect the CHP to the grid ("Input_DecoupleSwitches" must be used). When command is given, this output will be continuous.
- Output_RefuseSwitch_Decouple: Adds the ability to disconnect the CHP from the grid with the backup switch in case of failure to open the main in 200ms ("Input_DecoupleSwitches" must be used). When command is given, this output will give a 1sec pulse.

2.6 Portal (option)

When the client chooses to visualise data on the portal you must use the DataHub program instance ("MQTT_MINT") delivered in the PCU library. This program instance is made so no extra input is required, and all settings and communication parameters are set inside the control logic. This program instance should be called every 25ms with a watchdog of 0. In the program instance of the control function the input and output "strDataFromDatahub" and "strDataToDatahub" must be used. The input data of the input method must be an in port and be linked to the MQTT program instance. The return value of the output method must be configured as out port and linked to the DataHub instance.

```
PCU_FB_SmartControl_Master_Controller1( sLicense := '384075153004008480083160043900006010208833086638500028485840450860000002',
                                        strDataFromDatahub := strDatahubIn,
                                        arrRTUInterface := arrRTUInterface,
                                        arrCollectorInterface := arrCollectorInterface,
                                        arrSolarInterface := arrSolarInterface,
                                        arrWindInterface := arrWindInterface,
                                        arrChpInterface := arrChpInterface,
                                        strHMI := strHMI,
                                        xActive => xControlActive,
                                        xError => xControlError,
                                        strDataToDatahub => strDatahubOut,
                                        strDiagnose => strControlDiagnose);
```

Figure 21: DataHub method linking

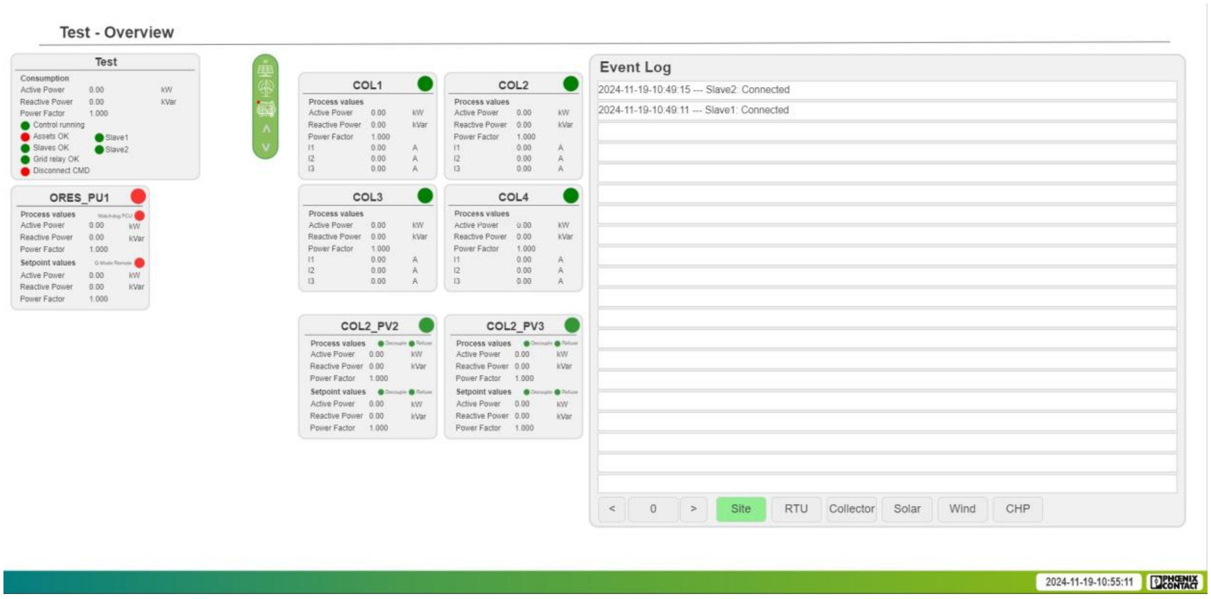
Both datatypes are compatible so they can be linked to each other. Once the connection of the ports is made, the MQTT broker can connect to the MINT DataHub, and data can start to flow between the systems.

Make sure the credentials are filled in to the configuration.

Check the DNS IP address in case the client has the DNS linked otherwise.

Check that the certificate of the MQTT communication is in the PLC memory.

2.7 eHMI (option)



Test - Overview

Test
Consumption
Active Power 0.00 kW
Reactive Power 0.00 KVar
Power Factor 1.000
● Control running
● Assets OK ● Slave1
● Slaves OK ● Slave2
● Grid relay OK
● Disconnect CMD

ORES_PU1 ●
Process values Watchdog PCU ●
Active Power 0.00 kW
Reactive Power 0.00 KVar
Power Factor 1.000
Setpoint values Q Mode Remote ●
Active Power 0.00 kW
Reactive Power 0.00 KVar
Power Factor 1.000

COL1 ●
Process values
Active Power 0.00 kW
Reactive Power 0.00 kVar
Power Factor 1.000
I1 0.00 A
I2 0.00 A
I3 0.00 A

COL2 ●
Process values
Active Power 0.00 kW
Reactive Power 0.00 kVar
Power Factor 1.000
I1 0.00 A
I2 0.00 A
I3 0.00 A

COL3 ●
Process values
Active Power 0.00 kW
Reactive Power 0.00 kVar
Power Factor 1.000
I1 0.00 A
I2 0.00 A
I3 0.00 A

COL4 ●
Process values
Active Power 0.00 kW
Reactive Power 0.00 kVar
Power Factor 1.000
I1 0.00 A
I2 0.00 A
I3 0.00 A

COL2_PV2 ●
Process values ● Decouple ● Refuse
Active Power 0.00 kW
Reactive Power 0.00 kVar
Power Factor 1.000
Setpoint values ● Decouple ● Refuse
Active Power 0.00 kW
Reactive Power 0.00 kVar
Power Factor 1.000

COL2_PV3 ●
Process values ● Decouple ● Refuse
Active Power 0.00 kW
Reactive Power 0.00 kVar
Power Factor 1.000
Setpoint values ● Decouple ● Refuse
Active Power 0.00 kW
Reactive Power 0.00 kVar
Power Factor 1.000

Event Log

```
2024-11-19-10:49:15 --- Slave2: Connected
2024-11-19-10:49:11 --- Slave1: Connected
```

Figure 22: eHMI

For the PCU library, there is a template HMI page on which all data is displayed. The eHMI is set up dynamically, so by clicking buttons and scrolling all assets can be displayed. The HMI is a feature made for commissioning and will deactivate after midnight. The eHMI can only be temporarily enabled by password unless the feature is ordered. In the case that the eHMI is ordered the data will always be displayed. The time displayed on the eHMI is the local time zone, in the logs the local time is also used.

2.8 Debugging

For debugging you can use the diagnose structures available on the function blocks. If an error occurs, it will always be displayed in the diagnose structure. In case of unclear messaging, contact your Phoenix Contact project engineer.

3 Software Change

3.1 V3.0.0

- Initial release version

3.2 V3.0.1

PCU_FB_RTU_ORES_RS485

- Changes to the control logic (inversion of values).

PCU_FB_SmartControl_ORES_RS485

- Changes to the control logic (inversion of values).

MQTT_MINT

- Changed certificate for the MQTT connection (future compatibility).
- Added DNS IP in data structure.

PCU_FB_SmartControl_Master_Controller

- Added DNS IP to the DataHub config method.

PCU_FB_SmartControl_Slave_Controller

- Added DNS IP to the DataHub config method.

3.3 V3.0.2

PCU_FB_Smartcontrol_Solar_DataInterface

- Changes in switch's logic.
- Added configurable time in the switch's method.

PCU_FC_MasterInputs

- Name change of function from RTUInputs.

PCU_FC_MasterOutputs

- Name change of function from RTUOutputs.

PCU_FC_SlaveInputs

- Name change of function from NobInputs.

PCU_FC_SlaveOutputs

- Name change of function from NobOutputs.
- Changed mapping Out1 = Disconnect close, Out2 = Refuse pulse.

PCU_FB_SmartControl_Master_Controller

- Changed internal logic for switch commands.
- Changed behaviour of setpoint based on the feedback of the switches.

3.4 V3.1.0

New

- Added PCU_FB_RTU_ORES_RS485_Multi, legacy communication with up to 4 RTU’s combined in 1.
- Added PCU_FB_SmartControl_ORES_RS485_Multi, for integration in the controller with up to 4 control loops.
- PCU_FB_SmartControl_Interface_Wind, for the control of wind assets.
- PCU_FB_SmartControl_Interface_CHP, for the control of CHP assets.

PCU_FB_SmartControl_Master_Control

- Added Wind and CHP interfaces.
- Added Wind and CHP in the control loop.
- Added Wind and CHP configuration methods.
- Added Wind and CHP data for the DataHub.
- Changed the DataHub methods to in-/output parameters.
- Improved disconnect CMD reaction time to < 150ms (@25ms cycle time) to all slaves.
- Changed license model.
- Changed eHMI functionality (added default dynamic eHMI page).
- Changed communication structure to slaves.

PCU_FB_SmartControl_Slave_Control

- Added Wind and CHP interfaces.
- Added Wind and CHP data for the DataHub.
- Changed the DataHub methods to in-/output parameters.
- Changed license model.
- Changed communication structure to master.

PCU_FB_RTU_ORES_RS485

- Changed name to PCU_FB_RTU_ORES_RS485_Single.
- Changed xConnected parameter to include checking for receiving of new data.
- Changed in-/output parameters to methods.
- Changed configuration input parameters to methods.

PCU_FB_RTU_Fluvius_IEC104

- Added license check in diagnose.
- Changed in-/output parameters to methods.
- Changed configuration input parameters to methods.
  PCU_FB_SmartControl_RTU_ORES_RS485.
- Added method Output_DisconnectOpen.
- Added method Output_RefuseOpen.
- Changed name to PCU_FB_SmartControl_ORES_RS485_Single.
- Changed xConnected parameter to include checking for receiving of new data.
- Added compatibility for positive and negative ORES setpoints/feedback.

PCU_FB_SmartControl_RTU_Fluvius_IEC104

- Added method Output_DisconnectOpen.
- Added method Output_RefuseOpen.
- Added license check in diagnose.

PCU_FB_SmartControl_Solar

- Added xDisconnectOpen and xRefuseOpen to method Input_StatusSwitches.