Modbus Asherah (MBANS)

Deployment Guide

Release 06 JULY 2023 (Linux Only)

R. P. Marques, University of São Paulo
ricardomarques@usp.br

To be distributed only to authorized parties.

MBANS is open-source and freely distributable, with the exception of the Asherah NPP Simulator Simulink code, which can only be used with permission of the International Atomic Energy Agency.

1 Introduction

This is the deployment guide for the Asherah NPP Simulator with Modbus interface (MBANS). It is based on the 15jul21 version distributed for IAEA with some extra features:

- Modbus support, instead of OPC UA, as the interface protocol.
- Versions with the Reactor Protection System enabled and disabled (recommended for training courses and exercises – the system never trips).
- Ready for deployment in virtual environments, such as containers.

For the moment, MBANS supports only Linux, having been tested with Ubuntu versions 20.04 and 22.04. Windows support would require rewriting part of the C-code routines, although the changes should be quite simple.

2 Distribution Contents

Archive **MBANS_ddmmyy.7z** contains a folder **MBANS** and the following subfolders:

2.1 Apps

The **Apps** folders contains C-code routines and libraries for Modbus implementation and integration with the simulator:

2.1.1 Modbus/mbclient

mbclient is a shared library that allows a simple interface to implement Modbus client functionality in the simulator as well as other eventual applications. The shared library also allows independent applications and routines to share a single Modbus connection, in order to optimize resource usage.

The shared library **libmclient.so** is used by the client functions implemented in the Asherah simulator and is based on the open-source Modbus implementation libmodbus[1].

The source code and all files required for compilation are included, including a script (**build_mbclient.sh**) for building the shared library.

Note that the **libmclient.so** has to be in folder **/usr/lib** in order to be accessible to the client applications.

2.1.2 Modbus/Server

The application modbus_server implements a simple Modbus server for a limited number of signals and simultaneous connections, adequate for the MBANS simulator and similar small-scale applications.

[1] Available in https://libmodbus.org/

Source code and all files required for compilation are included, including configuration files for use with Visual Studio Code[2].

modbus_server uses files

modbus_init_coil.txt
modbus_init_discinp.txt
modbus_init_holdreg.txt
modbus_init_inpreg.txt

to load initial conditions for all served values.

Appendix A presents the general structure of each file.

2.1.3 Modbus/Simulink Client

This folder contains the source code (***.c** files) and pre-compiled libraries (***.mexa64** files) for integration with the Simulink-based simulator in the form of Simulink C-coded S-functions.

A Matlab script (**mex_sfun_modbus_linux.m**) for compilation of the routines is also included.

The available routines are:

*For writing locally served read-only signals, using internal access to the server (note that this requires a RAM folder **/ramdisk** properly configured in the host):*

sfun_local_modbus_write_discinp.*

For writing discrete inputs (Modbus binary read-only signals) to the server. Text file **/ramdisk/discrete_inputs.txt** is used for data exchange.

sfun_local_modbus_write_inpreg.*

For writing 16-bit register inputs (Modbus integer read-only signals) to the server. Text file **/ramdisk/input_registers.txt** is used for data exchange.

*For reading/writing Modbus signals, using the Modbus protocol (the server is specified by files **modbus_srv_ip.txt** and **modbus_srv_port.txt**)*

sfun_ modbus_read_coil.*

For reading coils (Modbus binary read/write signals) from the server. This routine uses the Modbus protocol with the server specified by the text files.

sfun_ modbus_read_discinp.*

For reading discrete inputs (Modbus binary read-only signals) from the server. This routine uses the Modbus protocol with the server specified by the text files.

sfun_ modbus_read_holdreg.*

[2] Available in https://code.visualstudio.com/

For reading holding registers (Modbus integer read/write signals) from the server. This routine uses the Modbus protocol with the server specified by the text files.

sfun_ modbus_read_inpreg.*

For reading input registers (Modbus integer read-only signals) from the server. This routine uses the Modbus protocol with the server specified by the text files.

sfun_ modbus_write_coil.*

For writing coils (Modbus binary read/write signals) on the server. This routine uses the Modbus protocol with the server specified by the text files.

sfun_ modbus_write_holdreg.*

For writing holding registers (Modbus integer read/write signals) on the server. This routine uses the Modbus protocol with the server specified by the text files.

2.1.4 Container

This folder contains a complete example of a Docker Container[3] implementation of MBANS. The context is simplified, with the simulator not connected to any virtual network and the Modbus server listening to port 502 of the host. A proper implementation will require adaptations.

File **Dockerfile** should be used for building the container image. Folder **files** contains all files required for the docker container implementation.

Scripts **mbans_build.sh** (image building) and **mbans_deploy.sh** (container deployment) can be used as templates for proper implementations.

Files **ramdisk.txt** and **readme.txt** contain more information and further instructions.

2.1.5 Docs

General documentation for MBANS, including this document.

2.1.6 Docs/IO List

I/O list for MBANS. This folder contains the original ANS I/O list (**IO List ANS (USP) - Rev 2020-12-02.xlsx**) and the Modbus version, implemented in MBANS (**MBANS - Modbus IO List - 26may2023 - rev1.xlsx**) as Excel spreadsheets.

There is also a Matlab script (**create_modbus_files.m**) that creates the Modbus initial condition text files directly from the spreadsheets.

2.1.7 MBANS Runtime

Folder for MBANS compilation. Contains all files needed to build the compiled runtime version of MBANS.

[3] See https://www.docker.org

3 Deployment instructions

MBANS can be deployed in a docker container virtual environment or directly on the host (bare metal or virtual machine).

3.1 Container Deployment

This is the recommended means of deployment, eliminating dependency version compatibility problems and providing optimal soft-real-time performance.

3.1.1 Compatibility

All code was compiled in an updated (June/2023) Ubuntu 20.04 and also tested in Ubuntu 22.04. Deployment in older Ubuntu versions or other Linux distributions may require availability of compatible libraries and other dependencies or even code recompilation.

Compatibility is not an issue with the container deployment, for the Ubuntu 20.04 environment is loaded for the container.

3.1.2 Requirements

- Ubuntu 20.04 or 22.04 (this is a recommendation only - may work in different versions or Linux distributions);
- Docker container installed;

3.1.3 Deployment

Open a terminal in the **Container** folder and run the following command

```
$ ./mbans_build.sh
```

to build the container image. This has to be done only once, but notice that any change to the **Dockerfile** or to **files** folder will require that this script is executed again to update the container image.

The simulator version with RPS disabled is built by default. To change, edit **Dockerfile** and rerun the build script.

Execute the following command

```
$ ./mbans_deploy.sh
```

to deploy the environment. Notice that the supplied environment is just a template and should be adapted for use in a real system.

With the deployment, the Modbus server and the simulator start automatically and the simulator I/O signals are available on port 502.

3.2 Host Deployment

The environment can be executed on a host, whether bare-metal or a virtual machine, with superior soft-real-time performance on bare-metal hosts.

3.2.1 Compatibility

All code was compiled in an updated (June/2023) Ubuntu 20.04 and also tested in Ubuntu 22.04. If the host operational system has a different version or is part of another distribution, there might appear issues related to incompatible versions of libraries and other dependencies.

3.2.2 Requirements

- Ubuntu 20.04 or 22.04 (this is a recommendation only - may or may not work in different versions or Linux distributions);
- Ramdisk implemented in **/ramdisk** folder. See file **ramdisk.txt** on **Container** folder for more information.
  Alternatively, a conventional disk folder **/ramdisk** can be used, with potentially inferior performance.
- **libmbclient.so** file, available in the distribution folder **mbclient**, should be copied to folder **/usr/lib** of the host.

3.2.3 Deployment

First, make sure that file **libmbclient.so** is available at **/usr/lib** folder.

Open a terminal in the **Modbus/Server** folder and execute command

```
$ sudo ./modbus_server
```

with response

```
## MB_SRV:  Local Modbus server.

## MB_SRV:  Listening in all IP addresses at port 502.
```

Notice that elevated privileges are required for running modbus_server at ports under 1000.

Open another terminal in folder **MBANS Runtime** and execute command

```
$ ./mbans_runtime_ddmmmyy_rpsoff
```

or

```
$ ./mbans_runtime_ddmmmyy_rpson
```

depending of the choice of RPS (off or on), with response

```
## mb_client_connect: Connected to server 127.0.0.1:502.

** starting the model **
```

In case of errors, explanatory messages should appear on each terminal. The simulator and the Modbus server should be running.

4 Troubleshooting

4.1 Troubleshooting for Container Deployment

Troubleshooting for container deployment is easier with a tool such as portainer.io[4], but can be also done from the command line.

1. Open a terminal in container mbans (using portainer.io or command "docker exec -it mbans /bin/bash"). In this container terminal, execute command

```
# ps -aux
```

The response should be like

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0   4116  3384 pts/0    Ss+  19:27   0:00 /bin/bash
root           9  0.0  0.0   2616   592 ?        Ss   19:27   0:00 /bin/sh ./mbans.sh
root          17  0.0  0.0   2524   636 ?        S    19:27   0:00 ./modbus_server
root          19  0.0  0.0   9556  3996 ?        S    19:27   0:01 ./mbans_runtime_29may23
root          20  0.0  0.0   4116  3516 pts/1    Ss+  19:27   0:00 bash
root          29  0.0  0.0   2616   528 pts/2    Ss   19:31   0:00 /bin/sh
root          36  0.0  0.0   5900  2844 pts/2    R+   19:48   0:00 ps -aux
```

Make sure that both **./modbus_server** and **./mbans_runtime_ddmmmyy** are running.

If one or both apps are not running, try running them manually in the container terminal, with commands

```
# ./modbus_server
```

or

```
# ./mbans_runtime_ddmmmyy
```

Take notice of any error message displayed and take actions accordingly.

2. To check if the Modbus server and simulator are working properly, run a Modbus client in the host or any computer that has direct network access to the host. Recommended Modbus clients are

- Radzio! Modbus Master Simulator[5] (for Windows – the best option), or
- QT Modbus Master[6] (for Linux).

Using the client, check the real-time behavior of the signals of the I/O list of Appendix B in the Modbus server. A suggested maneuver is to change the power setpoint (CTRL_RXPowerSetpoint) and then watch the behavior of the other signals, especially reactor power (RX_ReactorPower).
Keep in mind that the data is served as 16-bit unsigned integers (0 to 65535) and not in engineering units.

[4] Available in https://www.portainer.io/
[5] Available in https://en.radzio.dxp.pl/modbus-master-simulator/
[6] See https://doc.qt.io/qt-5/qtserialbus-modbus-master-example.html

4.2 Troubleshooting for Host Deployment

Troubleshooting for container deployment is similar, but more straightforward than with container deployment.

1. Check the terminals where the applications were deployed for any error messages and act accordingly.
2. Use a Modbus client in the same way as for container deployment.

5 Compilation Instructions

The distribution includes completely functional pre-compiled versions of all applications and libraries, but they can be modified and recompiled if needed.

5.1 libmbclient.so shared library

To compile **libmbclient.so**, open a terminal in folder **Apps/Modbus/mbclient** and execute command

```
$ ./build_mbclient.sh
```

The simulator is configured to look for the library in folder **/usr/lib**, so copy the generated **libmbclient.so** file to that folder.

Files **modbus_srv_ip.txt** and **modbus_srv_port.txt**, which should be located on the same folder of the application that uses the library, are used to specify the Modbus server IP address and ports for the connections. Default value is 127.0.0.1:502.

This C-code library implements simplified functions for connecting, disconnecting, reading and writing to a Modbus server that share a same connection, in order to minimize resource usage both on client and server sides. See files **mbclient.c** and **mbclient.h** for more details. It is based on the libmodbus library.

5.2 modbus_server

The recommended way to compile the C-coded Modbus server is to open folder **Apps/Modbus/Server** in Visual Studio Code and build the application from there. Configuration files are included in the folder.

The server uses file **modbus_srv_port.txt**, which should be in the same folder as the server application, to specify the Modbus port (default is 502).

Initial values for the served variables are loaded from files **modbus_init_*.txt**, which should be also located in the same folder as the server application. Refer to Appendix A for more details.

5.3 Simulink client functions

Modbus access (read and write) from the simulator is performed by C-code S-functions interfacing with Simulink. To compile these functions, open Matlab, move to folder **Apps/Modbus/Simulink Client** and, from Matlab, execute command

```
>> mex_sfun_modbus_linux.m
```

The compiled S-functions, which are actually shared libraries, have extension **mexa64**. Notice that files **mbclient.h**, **modbus_srv_ip.txt** and **modbus_srv_port.txt** should be in the same folder as the source code for the S-functions.

Files **modbus_srv_ip.txt** and **modbus_srv_port.txt**, which should be located on the same folder of the Simulink application that uses the S-functions, are used to specify the Modbus server IP address and ports for the connections. Default value is 127.0.0.1:502.

5.4 MBANS Runtime

MBANS is originally a graphically-coded Simulink simulation diagram produced in Matlab/Simulink 2020a. To produce a compiled C-code runtime version, the package *Simulink Coder* is required.

Notice that before producing the runtime version, Matlab checks the original diagram, so all requirements for it to run from Simulink should be also met (an active Modbus server, **mexa64** files in the same folder, etc.)

To compile it, follow the sequence below.

1. Make sure that file **libmbclient.so** is in **/usr/lib** folder and that the following files are in the **MBANS Runtime** folder:

   Modbus dependencies:

   - libmodbus.a
   - mbclient.h
   - modbus.h

   Modbus server executable and configuration files:

   - modbus_server
   - modbus_srv_ip.txt
   - modbus_srv_port.txt
   - modbus_init_coil.txt
   - modbus_init_discinp.txt
   - modbus_init_holdreg.txt
   - modbus_init_inpreg.txt

   Real time module (source code and **mexa64** file, required for diagram checking):

   - sfun_pacer.c
   - sfun_pacer.mexa64

   Simulink client functions (both source code and **mexa64** files):

   - sfun_localmodbus_write_discinp.c
   - sfun_localmodbus_write_discinp.mexa64
   - sfun_localmodbus_write_inpreg.c
   - sfun_localmodbus_write_inpreg.mexa64
   - sfun_modbus_read_coil.c
   - sfun_modbus_read_coil.mexa64
   - sfun_modbus_read_discinp.c
   - sfun_modbus_read_discinp.mexa64

- sfun_modbus_read_holdreg.c
- sfun_modbus_read_holdreg.mexa64
- sfun_modbus_read_inpreg.c
- sfun_modbus_read_inpreg.mexa64
- sfun_modbus_write_coil.c
- sfun_modbus_write_coil.mexa64
- sfun_modbus_write_holdreg.c
- sfun_modbus_write_holdreg.mexa64

Data files for the simulator:

- mbans_runtime_ddmmmyy_RPS_OFF.mat
- mbans_runtime_ddmmmyy_RPS_ON.mat

Simulation diagram (Simulink file):

- mbans_runtime_ddmmmyy.xls

2. Open a terminal in folder **MBANS Runtime** and start the Modbus server. This is required for diagram checking before compilation.

```
$ sudo ./modbus_server
```

Notice that sudo is required for modbus_server to have access to port 502.

3. Open Matlab, move to the **MBANS Runtime** folder and execute

```
>> load mbans_runtime_ddmmmyy_RPS_OFF.mat
```

or

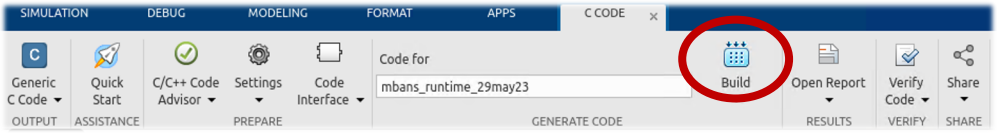
```
>> load mbans_runtime_ddmmmyy_RPS_ON.mat
```

Depending on the choice of RPS on or off (off is recommended for training courses and exercises).

4. In Matlab, open the Simulink diagram:

```
>> mbans_runtime_ddmmmyy
```

5. Once the diagram is open, press the “Build” button in the Simulink interface (the button is in APPS Tab > Simulink Coder Icon > C CODE Tab):



Take notice of the messages in the Simulink diagnostics window.

At the end of the process, an executable file **mbans_runtime_ddmmmyy** will be created in the **MBANS Runtime folder**.

6. Rename generated file **mbans_runtime_ddmmmyy** to **mbans_runtime_ddmmmyy_rpsoff** or **mbans_runtime_ddmmmyy_rpson**, depending on the initial choice.

5.5 Compiling for Windows

MBANS can potentially run in Windows, requiring only minimal changes to the code, such as:

- Replacing Linux network dependencies in the source code for their Windows counterparts.
- Rewriting Linux file names in the source code for their Windows equivalents (e.g. replacing "/" for "\".
- Compiling mbclient for Windows, producing a **dll** file instead of a **so** one.

6 Notes and Known Limitations

- Current version of the routines and applications only accept contiguous addresses (00000, 00001, 00002, etc.) for Modbus addressing.
  On the init files, only the first address is read. All subsequent variables are contiguously allocated, even if a different address is assigned in the init files.
  This should be corrected in future versions.

- Current version of the client routines and applications accept connections to only one single Modbus server, specified by files **modbus_srv_ip.txt** and **modbus_srv_port.txt**.

- The simulator adopts a movement max rate of 10 step/s for the reactor control rods (a typical real-world value would be 1 step/s). This is done to accelerate the transients so the exercises are more expedited.
  Experienced NPP operators might note that the transients are too fast and slightly more oscillatory than in the real world.

- Temperature values are presented in Kelvin (as produced by the simulator), but can be easily converted to Celsius for more convenient viewing in HMI screens. Just subtract 273.15 from the range limits.

Appendix A – Modbus Init Files

The Modbus init files are used to load initial values to all served variables. For this distribution, the whole ANS I/O list is included, but modifications can be easily done.

The discrete variables (coils and discrete inputs) follow the structure below, used in files **modbus_init_coil.txt** and **modbus_init_discinp.txt**

[address] [tag] [init value (0/1)]

For example:

```
00000 RC1_PumpOnOffCmd      1
00001 RC2_PumpOnOffCmd      1
00002 CR_SCRAMCmd           0
```

These files can be automatically generated from the I/O list spreadsheet (recommended) or manually edited.

The integer variables (holding registers and input registers) follow the structure below, used in files **modbus_init_holdreg.txt** and **modbus_init_inpreg.txt**

[address] [tag] [init value (eng. units)] [min (eng. units)] [max (eng. units)]

For example:

```
00000 RC1_PumpSpeedCmd          100.000000 0.000000 100.000000
00001 RC2_PumpSpeedCmd          100.000000 0.000000 100.000000
00002 CR_PosCmd                 833.000000 0.000000 1000.000000
```

In this example, variable RC1_PumpSpeedCmd receives an initial value of 100% for the range {0, … ,100}, converted to a 16-bit unsigned integer, that is, the server will be loaded with the value 65535. To inject a speed command of 50%, address 00000 of the holding register memory area in the server should receive a value of 32768, and so on.

These files can be automatically generated from the I/O list spreadsheet (recommended) or manually edited.

Adding signals to the Modbus server does not require recompilation, but the simulator expects the assigned variables to be present in the precise addresses of the I/O list. Modifications of this behavior require recoding and recompilation of the simulator.

Appendix B – I/O List

COILS

Address	Tag	Rated Value	Comment
00000	RC1_PumpOnOffCmd	1	
00001	RC2_PumpOnOffCmd	1	
00002	CR_SCRAMCmd	0	
00003	PZ_BackupHeaterPowCmd	0	
00004	AF_MakeupPumpCmd	1	
00005	SD_SafetyValveCmd	0	
00006	TB_IsoValveCmd	1	
00007	CC_PumpOnOffCmd	1	
00008	FW_Pump1OnOffCmd	1	
00009	FW_Pump2OnOffCmd	1	
00010	FW_Pump3OnOffCmd	0	
00011	CE_Pump1OnOffCmd	1	
00012	CE_Pump2OnOffCmd	1	
00013	CE_Pump3OnOffCmd	0	
00014	INT_SimulationStopCmd	0	(not a process signal, 1 stops simulation)

DISCRETE INPUTS (read-only)

Address	Tag	Rated Value	Comment
00000	CR_SCRAM	0	SCRAM confirmation
00001	AF_MakeupPumpOnOff	1	
00002	SD_SafetyValvePos	0	
00003	TB_IsoValvePos	1	
00004	GN_GenBreak	1	(normal situation = 1; circuit disconnected = 0)
00005	CC_PumpOnOff	1	
00006	FW_Pump1OnOff	1	
00007	FW_Pump2OnOff	1	
00008	FW_Pump3OnOff	0	
00009	CE_Pump1OnOff	1	
00010	CE_Pump2OnOff	1	
00011	CE_Pump3OnOff	0	

HOLDING REGISTERS

Address	Tag	Rated Value	Min Range	Max Range	Units	Comment
00000	RC1_PumpSpeedCmd	100	0	100	%	Generated by the Controllers, not sent from SCADA
00001	RC2_PumpSpeedCmd	100	0	100	%	Generated by the Controllers. not sent from SCADA
00002	CR_PosCmd	833	0	1000	step	Generated by the Controllers, not sent from SCADA
00003	PZ_MainHeaterPowCmd	0	0	100	%	Generated by the Controllers, not sent from SCADA
00004	PZ_CL1SprayValveCmd	0	0	100	%	Generated by the Controllers, not sent from SCADA
00005	PZ_CL2SprayValveCmd	0	0	100	%	Generated by the Controllers, not sent from SCADA
00006	AF_MakeupValveCmd	0	0	100	%	Generated by the Controllers, not sent from SCADA
00007	AF_LetdownValveCmd	0	0	100	%	Generated by the Controllers, not sent from SCADA
00008	SD_CtrlValveCmd	0	0	100	%	Generated by the Controllers, not sent from SCADA
00009	TB_SpeedCtrlValveCmd	100	0	100	%	Generated by the Controllers, not sent from SCADA
00010	CC_PumpSpeedCmd	100	0	100	%	Generated by the Controllers, not sent from SCADA
00011	FW_Pump1SpeedCmd	100	0	100	%	Generated by the Controllers, not sent from SCADA
00012	FW_Pump2SpeedCmd	100	0	100	%	Generated by the Controllers, not sent from SCADA
00013	FW_Pump3SpeedCmd	0	0	100	%	Generated by the Controllers, not sent from SCADA
00014	CE_Pump1SpeedCmd	100	0	100	%	Generated by the Controllers, not sent from SCADA
00015	CE_Pump2SpeedCmd	100	0	100	%	Generated by the Controllers, not sent from SCADA
00016	CE_Pump3SpeedCmd	0	0	100	%	Generated by the Controllers, not sent from SCADA
00017	CTRL_RXPowerSetpoint	100	0	110	%	Sent from SCADA
00018	CTRL_PZPressSetPoint	15.106	0	18	MPa	Sent from SCADA
00019	CTRL_CDLevelSetpoint	0	0	0	m	(just placeholders, not implemented)
00020	CTRL_CDPressSetpoint	0	0	0	Pa	(just placeholders, not implemented)
00021	CTRL_PZLevelSetPoint	0	0	0	m	(just placeholders, not implemented)
00022	CTRL_RC1FlowSetpoint	0	0	0	kg/s	(just placeholders, not implemented)
00023	CTRL_RC2FlowSetpoint	0	0	0	kg/s	(just placeholders, not implemented)
00024	CTRL_SG1LevelSetpoint	0	0	0	kg/s	(just placeholders, not implemented)
00025	CTRL_SG1PressSetpoint	0	0	0	MPa	(just placeholders, not implemented)
00026	CTRL_SG2LevelSetpoint	0	0	0	m	(just placeholders, not implemented)
00027	CTRL_SG2PressSetpoint	0	0	0	MPa	(just placeholders, not implemented)
00028	CTRL_MeanCoolTempSetpoint	0	0	0	K	(just placeholders, not implemented)
00029	CTRL_ColdLegTempSetpoint	0	0	0	K	(just placeholders, not implemented)
00030	CTRL_TBSpeedSetpoint	0	0	0	%	(just placeholders, not implemented)

INPUT REGISTERS (read-only)

Address	Tag	Rated Value	Min Range	Max Range	Units	Comment
00000	RC1_PumpDiffPress	1.051	0	5	Mpa	
00001	RC1_PumpSpeed	100	0	100	%	
00002	RC1_PumpFlow	8206.6	0	10000	kg/s	
00003	RC1_PumpTemp	338.15	0	1000	K	
00004	RC2_PumpDiffPress	1.051	0	5	Mpa	
00005	RC2_PumpSpeed	100	0	100	%	
00006	RC2_PumpFlow	8206.6	0	10000	kg/s	
00007	RC2_PumpTemp	338.15	0	1000	K	
00008	RX_MeanCoolTemp	576.75	0	1000	K	
00009	RX_InCoolTemp	562.94	0	1000	K	
00010	RX_OutCoolTemp	590.62	0	1000	K	
00011	RX_CladTemp	948.28	0	1000	K	
00012	RX_FuelTemp	948.28	0	1000	K	
00013	RX_TotalReac	6.1835E-06	-1.00E-05	1.00E-05	$	
00014	RX_ReactorPower	100	0	120	%	100% corresponds to 2772 MWt / 830 MWe
00015	RX_ReactorPress	15.166	0	20	MPa	
00016	RX_CL1Press	15.365	0	20	MPa	
00017	RX_CL2Press	15.365	0	20	MPa	
00018	RX_CL1Flow	8801.4	0	10000	kg/s	
00019	RX_CL2Flow	8801.4	0	10000	kg/s	
00020	CR_Position	833	0	1000	step	
00021	PZ_Press	15.106	0	20	MPa	
00022	PZ_Temp	586.95	0	1000	K	
00023	PZ_Level	6	0	10	m	
00024	SG1_InletTemp	590.48	0	1000	K	
00025	SG1_OutletTemp	562.9	0	1000	K	
00026	SG2_InletTemp	590.48	0	1000	K	
00027	SG2_OutletTemp	562.9	0	1000	K	
00028	AF_MakeupValvePos	0	0	100	%	
00029	AF_LetdownValvePos	0	0	100	%	
00030	AF_MakeupFlow	0	0	1000	kg/s	
00031	AF_LetdownFlow	0	0	1000	kg/s	
00032	SG1_InletWaterTemp	495.77	0	1000	K	
00033	SG1_OutletSteamTemp	553.08	0	1000	K	

Address	Tag	Rated Value	Min Range	Max Range	Units	Comment
00034	SG1_InletWaterFlow	745,14	0	1000	kg/s	
00035	SG1_OutletSteamFlow	745,14	0	1000	kg/s	
00036	SG1_WaterTemp	553,08	0	1000	K	
00037	SG1_SteamTemp	553,08	0	1000	K	
00038	SG1_Press	6,41	0	20	Mpa	
00039	SG1_Level	15	0	20	m	
00040	SG2_InletWaterTemp	495,77	0	1000	K	
00041	SG2_OutletSteamTemp	553,08	0	1000	K	
00042	SG2_InletWaterFlow	745,14	0	1000	kg/s	
00043	SG2_OutletSteamFlow	745,14	0	1000	kg/s	
00044	SG2_WaterTemp	553,08	0	1000	K	
00045	SG2_SteamTemp	553,08	0	1000	K	
00046	SG2_Press	6,41	0	20	Mpa	
00047	SG2_Level	15	0	20	Mpa	
00048	SD_CtrlValvePos	0	0	100	%	
00049	TB_Speed	157,08	0	250	rad/s	
00050	TB_InSteamPress	6,41	0	20	MPa	
00051	TB_OutSteamPress	5200	0	10000	Pa	
00052	TB_SpeedCtrlValvePos	100	0	100	%	
00053	TB_InSteamFlow	1490,28	0	2000	kg/s	
00054	GN_GenElecPow	789	0	1000	MW	
00055	GN_GridFreq	50	0	100	Hz	
00056	GN_GenFreq	50	0	100	Hz	
00057	CD_Level	1	0	2	m	
00058	CD_SteamTemp	306,46	0	1000	K	
00059	CD_CondTemp	306,46	0	1000	K	
00060	CD_Press	5200	0	10000	Pa	
00061	CD_InSteamFlow	1490,28	0	2000	kg/s	
00062	CD_OutCondFlow	1490,28	0	2000	kg/s	
00063	CC_PumpInletTemp	298,15	0	1000	K	
00064	CC_PumpOutletTemp	302,48	0	1000	K	
00065	CC_PumpSpeed	100	0	100	%	
00066	CC_PumpFlow	1,73E+05	0	2,00E+05	kg/s	
00067	CC_PumpTemp	338,15	0	1000	K	
00068	FW_TankPress	1	0	2	MPa	
00069	FW_TankLevel	4	0	10	m	

Address	Tag	Rated Value	Min Range	Max Range	Units	Comment
00070	FW_Pump1DiffPress	5,41	0	20	MPa	
00071	FW_Pump1Flow	745,14	0	1000	kg/s	
00072	FW_Pump1Speed	100	0	120	%	
00073	FW_Pump1Temp	343,15	0	1000	K	
00074	FW_Pump2DiffPress	5,41	0	20	MPa	
00075	FW_Pump2Flow	745,14	0	1000	kg/s	
00076	FW_Pump2Speed	100	0	120	%	
00077	FW_Pump2Temp	343,15	0	1000	K	
00078	FW_Pump3DiffPress	5,41	0	20	MPa	
00079	FW_Pump3Flow	0	0	1000	kg/s	
00080	FW_Pump3Speed	0	0	120	%	
00081	FW_Pump3Temp	343,15	0	1000	K	
00082	CE_Pump1DiffPress	0,9948	0	20	MPa	
00083	CE_Pump1Speed	100	0	120	%	
00084	CE_Pump1Flow	745,14	0	1000	kg/s	
00085	CE_Pump1Temp	343,15	0	1000	K	
00086	CE_Pump2DiffPress	0,9948	0	20	MPa	
00087	CE_Pump2Speed	100	0	120	%	
00088	CE_Pump2Flow	745,14	0	1000	kg/s	
00089	CE_Pump2Temp	343,15	0	1000	K	
00090	CE_Pump3DiffPress	0,9948	0	20	MPa	
00091	CE_Pump3Speed	0	0	120	%	
00092	CE_Pump3Flow	0	0	1000	kg/s	
00093	CE_Pump3Temp	343,15	0	1000	K	
00094	INT_SimulationTime	0	0	65535	s	(not a process signal)