SD02624G/00/EN/04.26-00
71757483
2026-02-18

Valid as of version
18.3.4

Special Documentation
Tankvision Professional NXA85
Tankvision LMS NXA86B
Terminalvision NXS85

Modbus Slave Configuration
Inventory Gauging

Endress+Hauser
People for Process Automation

Table of contents

1 Introduction

This manual documents the interface the Tankvision Professional and Terminalvision provides to enable third-party vendors to access internal system data. 2 interfaces are available:
- Modbus
- OPC: The system includes a UA based OPC interface which is documented in the document BA02061G.

A full list of all available data items is available on request. Please contact your supplier for details.

1.1 Modbus Slave Versions

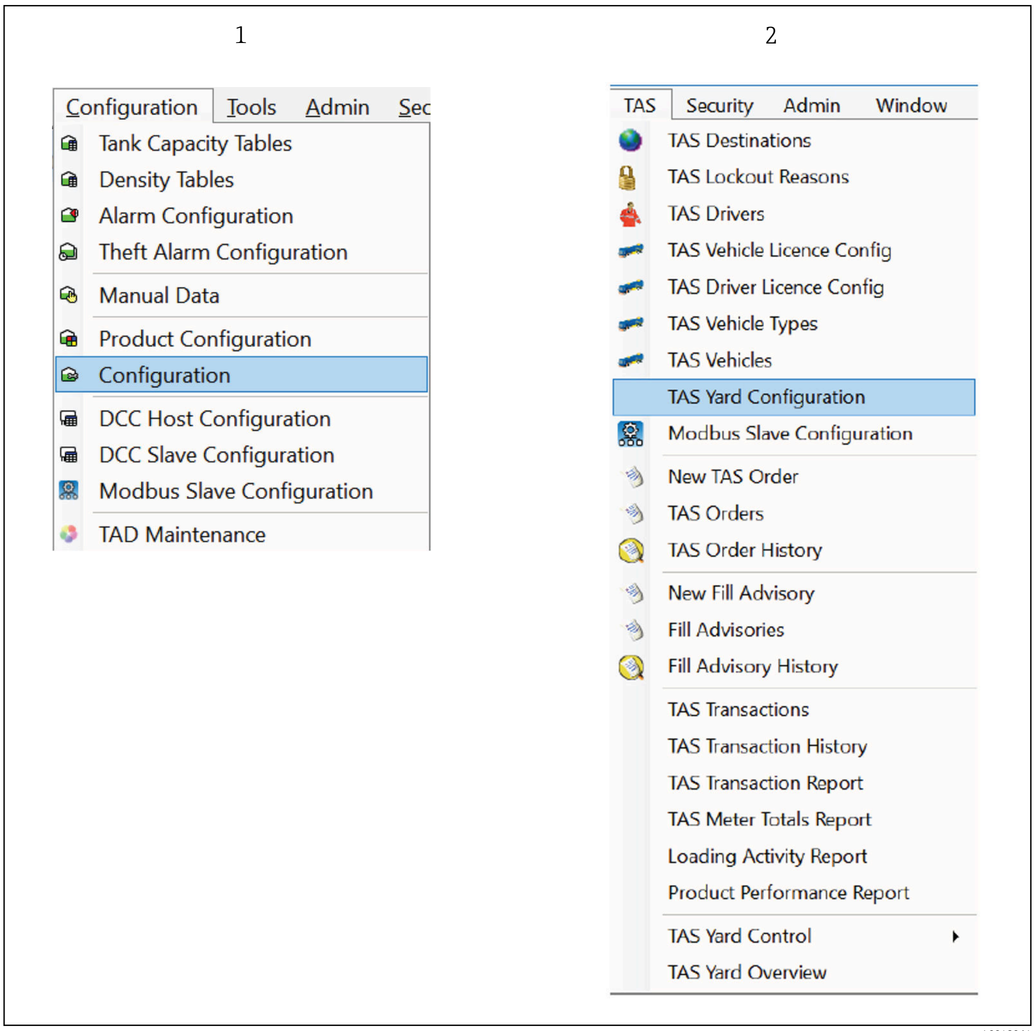
As of version 18.3.1, the software introduces a new Modbus slave service which contains extra functionality compared to the previous legacy Modbus slave implementation. Some of the main new features are as follows:

- Support for terminal automation scenarios
- Many more data points can be accessed
- Configuration is synchronised across redundant servers
- New configuration tool

If data tunneling is required then the legacy Modbus slave must be used. See the DCC Configuration manual for details.

2 Configuring a physical port for Modbus

Before a Modbus slave can be configured, a physical port must be configured to support Modbus. The configuration tool for this can be opened by selecting **Configuration** > **Configuration** in tank gauging versions of the software and **TAS** > **TAS Yard Configuration** in terminal automation versions → 🖼 1, 📄 5.



🖼 1 *Menu option*

1 *Tank gauging versions*
2 *Terminal automation versions*

1. Add a new port if necessary.
2. Configure the port.
3. Select **Modbus** as protocol.

📖 Detailed information on adding and configuring new ports, Operating Instructions BA00391G.

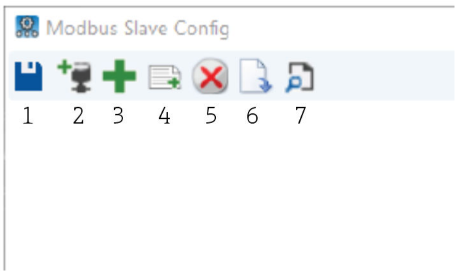
Important points should be considered for the port to be used as a Modbus slave:

Serial ports may only have one Modbus slave port associated with them. Network ports may support multiple Modbus slave ports on the same IP address providing the TCP port setting of the physical port selected is different for each Modbbus slave port to be configured.

The protocol selected must be **Modbus** if the port is intended to be used as a Modbus slave port. (As noted in the previous chapter data tunneling is only supported on the legacy Modbus slave. This scenario is not covered by this manual.)

3 Configuring a Modbus slave

The Modbus slave configuration tool may be opened from the main home screen by selecting the **Modbus Slave Configuration** option in the **Configuration** menu on tank gauging versions of the software or the **TAS** menu in terminal automation versions.



2 *Slave configuration tool*

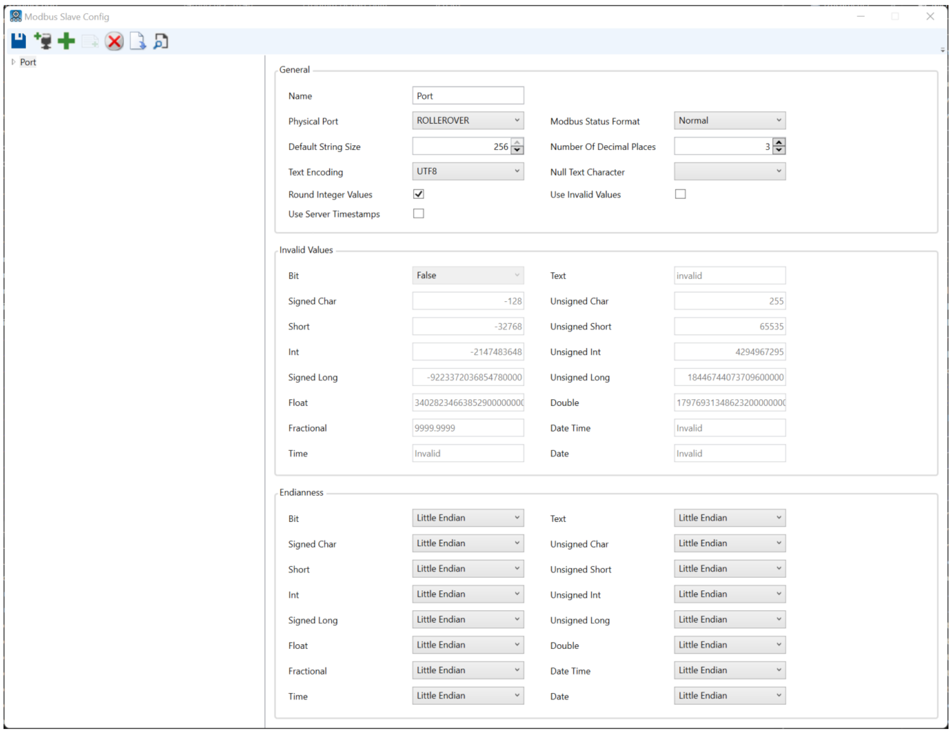
1 *Save icon: This is only available if all configuration items are valid. Items that are invalid will be indicated with a red box around the invalid data.*
2 *Add Modbus Port icon: Add a top level slave port.*
3 *Add Item icon: Add to the currently selected item. (The user may also use the Alt + Numeric+ key combination or by right-clicking with the mouse on an item).*
4 *Add Preconfigured Template icon: Some templates are supplied with preconfigured maps available. This is only enabled if a preconfigured map is available for the currently selected template. (See Modbus Template Index for the available preconfigured templates).*
5 *Delete icon: Deletes the currently selected item (and any items it contains).*
6 *Export icon: Exports the currently configured map to an xml file.*
7 *Preview icon: Preview the currently configured map.*

▸ To begin configuration click the **Add Modbus Port** icon (2).

The **Add Item** icon (3) is context sensitive and will add different items depending on which item is currently selected. The default item to add is a Modbus port if no other item is selected. It is also possible to use a right-click menu or the Alt + Numeric + key combination to add an item beneath the currently selected item.

On all screens in the configuration tool, it is not possible to progress with further changes if any invalid configuration is detected on the current screen.

3.1 Configure the port



General

Name	Port		
Physical Port	ROLLEROVER	Modbus Status Format	Normal
Default String Size	256	Number Of Decimal Places	3
Text Encoding	UTF8	Null Text Character	
Round Integer Values	✓	Use Invalid Values	☐
Use Server Timestamps	☐		

Invalid Values

Bit	False	Text	invalid
Signed Char	-128	Unsigned Char	255
Short	-32768	Unsigned Short	65535
Int	-2147483648	Unsigned Int	4294967295
Signed Long	-9223372036854780000	Unsigned Long	18446744073709600000
Float	34028234663852900000000	Double	17976931348623200000000
Fractional	9999.9999	Date Time	Invalid
Time	Invalid	Date	Invalid

Endianness

Bit	Little Endian	Text	Little Endian
Signed Char	Little Endian	Unsigned Char	Little Endian
Short	Little Endian	Unsigned Short	Little Endian
Int	Little Endian	Unsigned Int	Little Endian
Signed Long	Little Endian	Unsigned Long	Little Endian
Float	Little Endian	Double	Little Endian
Fractional	Little Endian	Date Time	Little Endian
Time	Little Endian	Date	Little Endian

3 *Modbus port configuration*

The configuration items are discussed below:

3.1.1 General

- **Name**
  Define a descriptive name for this Modbus slave port
  A Modbus port must have a name and the name must begin with an alphabetic character
- **Physical Port**
  The previously configured physical port to define a Modbus slave on
  In this example the physical port was named **Port**
  A physical port must always be selected
- **Default String Size**
  The default size of string data
  This can be configured individually for data items
  The maximum string size is 256 characters (see → 3, 8).
- **Text Encoding**
  Text values may be encoded as 8 or 16-bit characters
  16-bit characters allow the use of unicode characters
  In the case of 8-bit characters, two 8-bit characters will be encoded in each 16-bit register
- **Round Integer Values**
  If this item is ticked, any floating point value to be displayed as an integer will be rounded rather than truncated
  Truncation is the default mode of operation

- **Modbus Status Format**
  Defaults to **Normal**
  This sets the formatting for the parameter status fields.
  The current options are:
  - **Normal**
    A decimal number where:
    −1 = data is valid
    0 to 255 = Diagnostic code for invalid data (ie the DN number displayed on screen)
  - **Bit**
    Bit-mapped, where:
    Bit 1 Set = data is valid
    Bit 2 Set = data is manual
    Bit 9 Set = no reply from device (ie DN00)
    Bit 10 Set = data not ready (ie DN01)
    Bit 11 Set = data invalid (ie DN04)
  - **Datacon**
    Bit-mapped, where:
    Bit 7 Set = data invalid (ie DN04)
    Bit 8 Set = no reply from device (ie DN00)
  - **Enraf**
    Formatted as per the Enraf CIU Prime Modbus specification
  - **Saab**
    Bit-mapped, where:
    Bit 16 Set = data is valid
- **Number Of Decimal Places**
  Specifies the number of decimal places to consider for the **Fractional** data type
- **Null Text Character**
  By default the system uses the character code zero (ASCII null) to denote null characters in a text string
  The user may configure another ASCII charcter to use for this purpose by selecting from the drop down list (this character will be disregarded when decoding string values)
- **Use Invalid Values**
  If ticked, the values in the **Invalid Values** section will be published if the data item concerned is discovered to have a status of invalid
  For example, if the Use Invalid Values box was ticked in figure → 🗎 8 then any signed char data point which has a status of invalid will be published −128 regardless of the actual value of the data point.
- **Use Server Timestamps**
  If ticked the system will convert date and time values to local (server) time. Otherwise date and time values will be output in UTC format.

3.1.2 Invalid Values

The invalid values that will be published for each data type if the corresponding status signifies that the value is invalid (data points that do not have an associated status value will remain unchanged).

3.1.3 Endianness

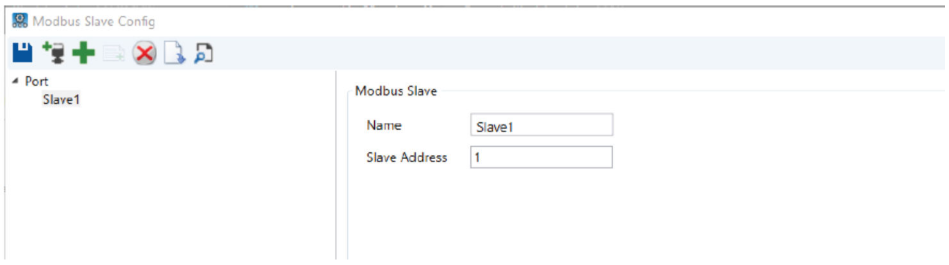
Defines the byte ordering of each output data type.

The byte ordering can be one of:
- Little Endian: Most significant register first
- Big Endian: Least significant register first
- Little Logical: Little Endian with the two bytes within the register swapped
- Big Logical: Big Endian with the two bytes within the register swapped

When writing strings, the string written must be exactly the size configured when adding an item in the **Add a data point** section. Any unused characters should be written as the null character configured in **Configure the port**.

3.2 Add a Slave

Using the **Add Item** icon (3) (or optionally the right-click menu or the keyboard shortcut) adds a Modbus slave to the currently selected port.



Name: Slave1
Slave Address: 1

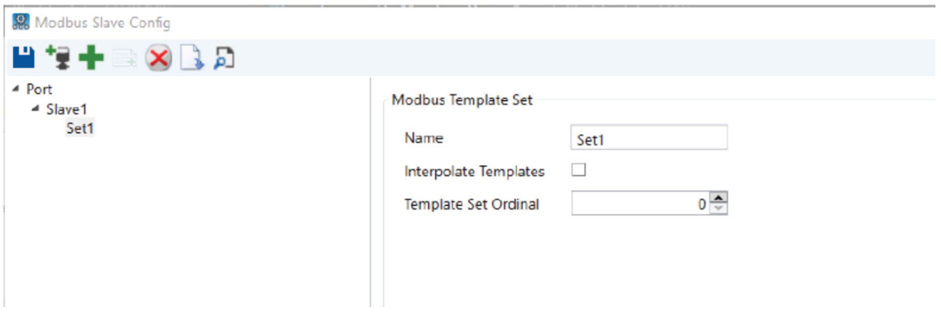
4 *Modbus slave*

In the case of network connected modbus slave ports, a single Modbus slave port may have one or more Modbus slaves configured on it. Each slave must be named and have a slave address that is unique on the Modbus slave port on which it is configured. Each slave address must be in the range 1 to 247.

Each slave configured is a container for a collection of Modbus template sets.

3.3 Add a template set

Using the **Add Item** icon (3) (or optionally the right-click menu or the keyboard shortcut) adds a Modbus template set to the currently selected Modbus slave.



Name: Set1
Interpolate Templates: ☐
Template Set Ordinal: 0

5 *Slave Template Set*

Each Modbus template set must be named.

The **Interpolate Templates** option determines how the templates in the set will be applied to system data objects.

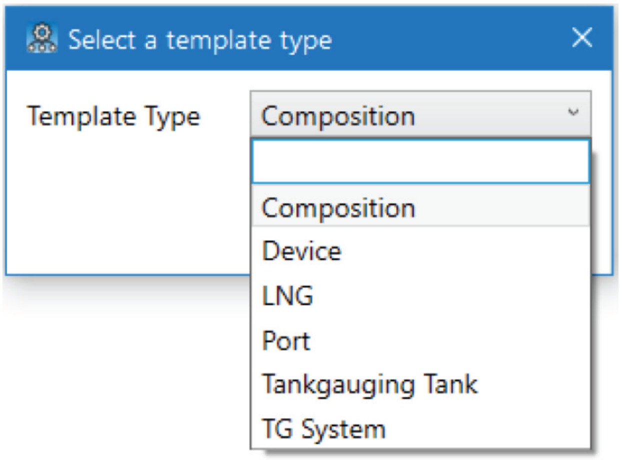
- Ticked: For each template configured, each system data object will be applied in turn
- Unticked: For each system data object, each template will be applied in turn

If the check box **Interpolate Templates** is ticked, any configured System templates will be applied first regardless of the template ordinal assigned to them since they do not relate to system data objects.

Each Modbus template set is a container for a collection of Modbus templates.

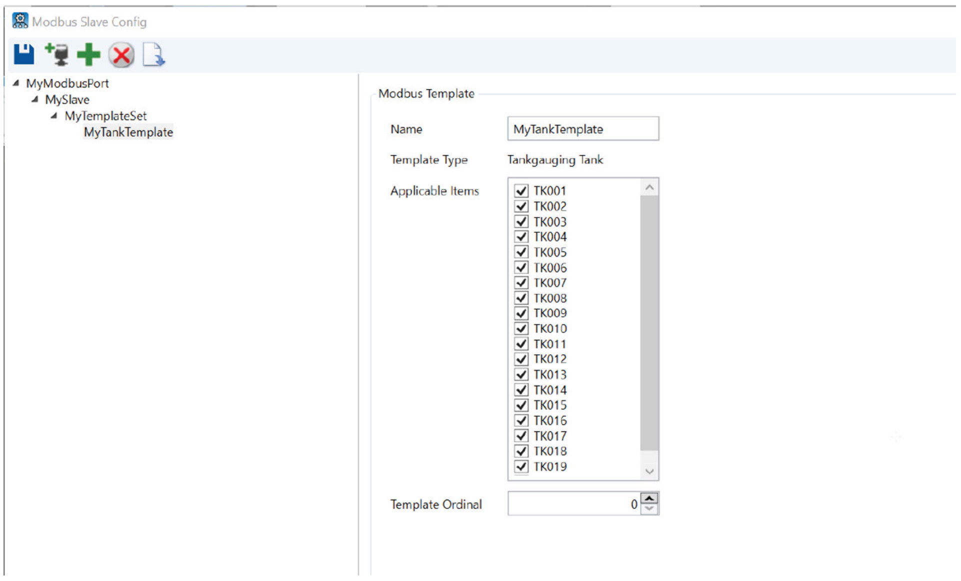
3.4 Add a Modbus Template

Using the **Add Item** icon (3) (or optionally the right-click menu or the keyboard shortcut) adds a Modbus template to the currently selected Modbus template set.



6 *Template type selection*

Each template represents a type of data item available in the system. When adding a template, the template type must be selected as shown in figure → 🖹 10.



Name: MyTankTemplate

Template Type: Tankgauging Tank

Applicable Items: ✓ TK001, ✓ TK002, ✓ TK003, ✓ TK004, ✓ TK005, ✓ TK006, ✓ TK007, ✓ TK008, ✓ TK009, ✓ TK010, ✓ TK011, ✓ TK012, ✓ TK013, ✓ TK014, ✓ TK015, ✓ TK016, ✓ TK017, ✓ TK018, ✓ TK019

Template Ordinal: 0

7 *Tankgauging Tank Template*

Figure → 7, 🖹 11 demonstrates the result of choosing a **Tankgauging Tank** template.

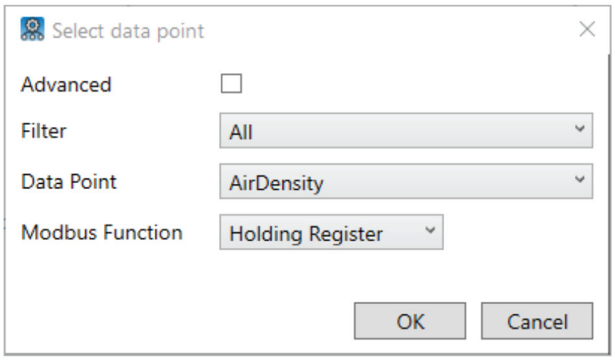
Name
A valid name must be entered for each template

Applicable Items
- By default, all applicable system items of type supported by the template are selected
- It is possible to deselect individual system items
- Only selected items will be considered by the system
- At least one item must be selected
- In the case of System or Composition templates, no selectable items will appear

Template Ordinal
This value determines the order in which templates are applied within the parent template set and must be unique with the current template

3.5 Add a data point

Using the **Add Item** icon (3) (or optionally the right-click menu or the keyboard shortcut) adds a Modbus item to the currently selected Modbus template.



Advanced	☐
Filter	All
Data Point	AirDensity
Modbus Function	Holding Register

8 Choose data point

Upon selecting to add a data point the dialog shown in figure → 8, 12 is displayed.

Advanced
By default, status values are displayed as single bit-mapped values. If the **Advanced** option is selected each bit of the status values becomes available as individual data points.

Please note that some status values only support certain status bits so not all options available will be applicable to every status value.

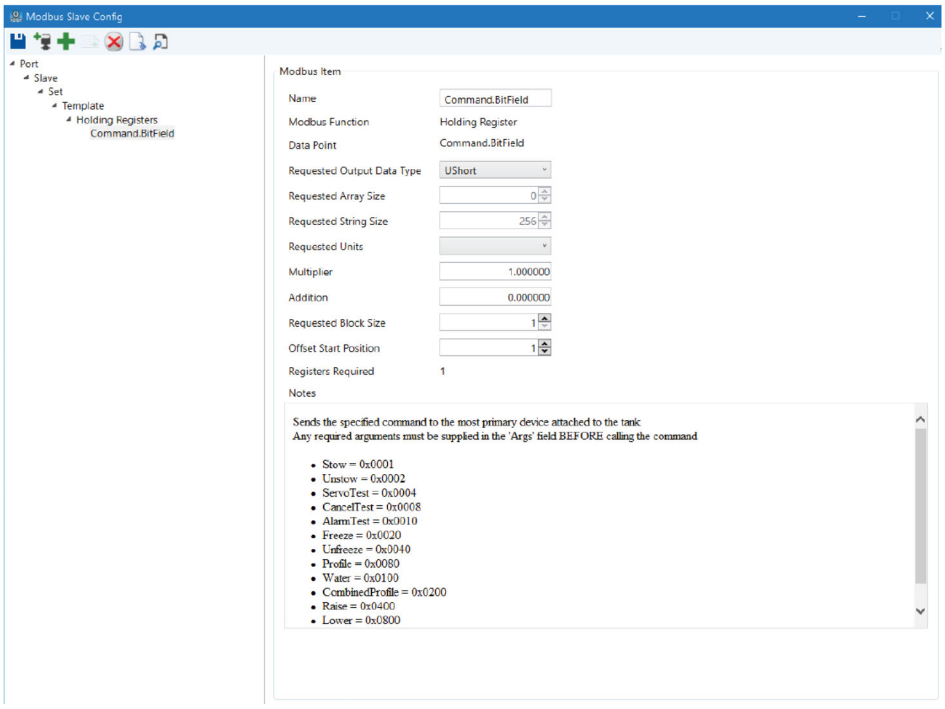
Filter
By default All data points are shown in the Data Point selection
For some templates it is possible to filter the list of available data points by their purpose
For example, it is possible to filter Tankgauging Tank data points to show only those applicable to live data

Data Point
This selection box contains all the data items that are available in the system and applicable to the currently selected Modbus template

Modbus function
The options here are one of:
- Coils
- Holding registers
- Input registers
- Input statuses

The configuration tool will allow the user to add any available data point to any Modbus function (even if that assignment does not make sense).

3.5.1 Configure the data point



Name	Command.BitField
Modbus Function	Holding Register
Data Point	Command.BitField
Requested Output Data Type	UShort
Requested Array Size	0
Requested String Size	256
Requested Units	
Multiplier	1.000000
Addition	0.000000
Requested Block Size	1
Offset Start Position	1
Registers Required	1

Notes

Sends the specified command to the most primary device attached to the tank
Any required arguments must be supplied in the 'Args' field BEFORE calling the command

- Stow = 0x0001
- Unstow = 0x0002
- ServoTest = 0x0004
- CancelTest = 0x0008
- AlarmTest = 0x0010
- Freeze = 0x0020
- Unfreeze = 0x0040
- Profile = 0x0080
- Water = 0x0100
- CombinedProfile = 0x0200
- Raise = 0x0400
- Lower = 0x0800

9 *Configure the data point*

Name
Each data point must be named. By default the name is the name of the system data point.

Requested Output Data Type
This defaults to the data type that best suits the type of the data in the system. Every data item can be configured to be output as any data type. The system will attempt to honour the configured data type. If such a data type conversion is not possible the data will be returned as its default type. See Appendix A for notes on Fractional and Date/Time related output types.

Requested Array Size
If the data item represents an array of values then it is possible to specify the number of array elements to publish. Arrays always start from position zero. The maximum number of array items is 512

Requested String Size
It is possible to override the default string size on a per data item basis.

Requested Units
Data items that support units can be set to automatically be converted to other relevant units. This defaults to the system base units. Any unit conversion is made before multiplier and addition values are applied.

Multiplier / Addition
If the data point value supports multiplication and addition then after any unit conversion is made, the data point value is multiplied by the multiplier before having the addition value added on. The multiplier may be less than 1 and the addition may be negative to facilitate division and subtraction respectively.

Requested Block Size
The requested block size is always at least the minimum number of registers required to publish the data point using the requested output type. It is possible to increase the number of registers taken by adjusting this value. This may be desired for example to allow space in the modbus map for additional tanks.

Offset start position
Determines the order in which this data item appears within its containing template. This must be unique within the containing template.

3.6 Saving

If the current configuration is validated the **Save** icon (1) will become available.

Clicking the **Save** icon has the following consequences:
- Configuration is saved to the database
- Configuration is synchronised to any redundant server
- The Modbus slave service on each server is restarted with the new configuration

3.7 Deleting Items

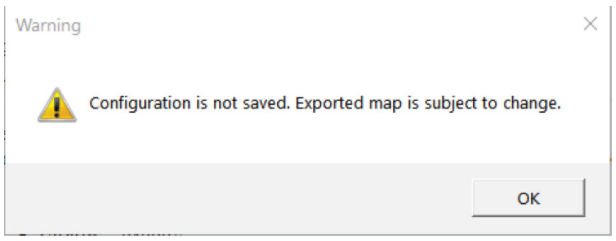
Any item may be deleted by clicking the **Delete** icon (5)

If the item contains 'child' items they will also be deleted.

3.8 Exporting the map

If the currently configured items are validated the **Export** icon (6) will become available. Clicking the export icon will result in the user being prompted to save an xml representation of the Modbus slave configuration. An example of the configuration from the previous sections is shown in the appendix.

If the current configuration has not yet been saved a warning as shown in → 10, 14 will be displayed to warn the user that the exported map may not represent the final saved configuration.

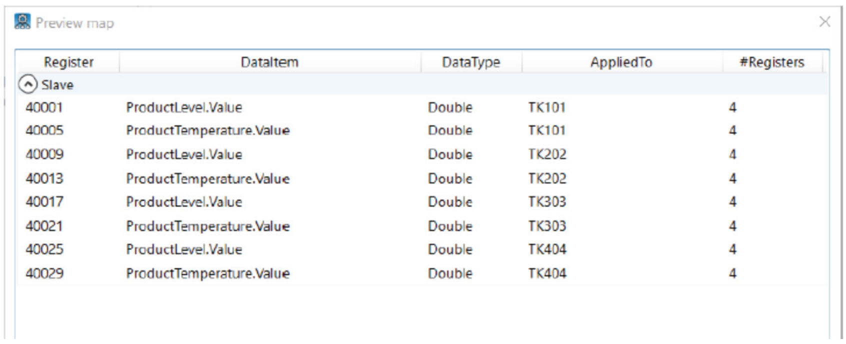

Warning

Configuration is not saved. Exported map is subject to change.

10 *Export warning*

3.9 Previewing the map

If the currently configured items are valid the preview icon (G) will become available. Clicking the preview icon will result in a preview of the currently configured map being displayed to the user as shown in fig → 11, 15. If the current configuration has not yet been saved a warning as shown in fig → 10, 14 will be displayed to warn the user that the previewed map may not represent the final saved configuration.



Register	DataItem	DataType	AppliedTo	#Registers
Slave				
40001	ProductLevel.Value	Double	TK101	4
40005	ProductTemperature.Value	Double	TK101	4
40009	ProductLevel.Value	Double	TK202	4
40013	ProductTemperature.Value	Double	TK202	4
40017	ProductLevel.Value	Double	TK303	4
40021	ProductTemperature.Value	Double	TK303	4
40025	ProductLevel.Value	Double	TK404	4
40029	ProductTemperature.Value	Double	TK404	4

11 Map Preview

4 Updating Chemical Composition via Modbus

The Modbus server includes the ability to update existing or create new chemical compositions. The procedure for this is detailed as follows:

		Start		
		↓		
		User Sets Composition Name		
		↓		
Yes	←	Does Composition Exist?	→	No
↓				↓
System Loads matching composition				System Creates New Composition
↓				↓
User Enters or Updates Composition Details	←	←	←	←
↓		↑		
Mole Fraction and Composition Valid?	→	No		
↓				
Yes				
↓				
User Writes to Save Composition				
↓				
System Attemts To Save data				
↓				
Save succeeded?				
↓				
Yes				
↓				
System Sets SaveSuccess to 1				

Saving composition changes is a computationally expensive operation. It may be necessary to increase any response timeout threshold to prevent spurious timeout errors.

5 Appendix A: Notes on output data types

Although most data types available as output data types are self-explanatory, there are are some data types with specific formats as shown below.

5.1 Fractional

The fractional data type exposes numeric data in two distinct parts - the integral portion and the fractional portion. When this data type is selected the integral portion of the value is exposed in one register and the fractional part (as an integral value) in the next register. For example: The value 10234:546 would be exposed as one register containing the value 10234 and the next register containing the value 546.

5.2 DateTime

The **DateTime** data type exposes a date and time in the following format as a string value.

ddMMyyyy:HHmmss.fffff zzz

Where:

- **dd** two-digits representing the day
- **MM** two-digits representing the month
- **yyyy** four-digits representing the year
- **HH** two-digits representing the hour
- **mm** two-digits representing minutes
- **ss** two-digits representing seconds
- **fffff** five-digits representing fractional seconds
- **zzz** six characters representing the time zone offset (eg. +01:00)

5.3 DateTimeMulti (NXA85 and NXS85 only)

The **DateTimeMulti** data type exposes a date and time in the following format as a collection of registers.

The following list covers the Register Offset and its respective value using the scheme:
- **Register Offset** its respective value
- **0** Day
- **1** Month
- **2** Year
- **3** Hour
- **4** Minutes
- **5** Seconds
- **6** Milliseconds
- **7** TimeZone Offset (in hours)

5.4 Short dates/times

In addition to the above **dateTime** format, date-times can also be exposed in a shorthand form.

5.4.1 Date

The **Date** data type exposes a date and time in the following format as a string value.

ddMMyyyy

Where:

- **dd** two-digits representing the day
- **MM** two-digits representing the month
- **yyyy** four-digits representing the year

5.4.2 DateMulti (NXA85 and NXS85 only)

The **DateMulti** data type exposes a date and time in the following format as a collection of registers.

The following list covers the Register Offset and its respective value using the scheme:

- **Register Offset** its respective value
- **0** Day
- **1** Month
- **2** Year

5.4.3 Time

The **Time** data type exposes a date and time in the following format as a string value.

HHmmss.fffff zzz

Where:

- **HH** two-digits representing the hour
- **mm** two-digits representing minutes
- **ss** two-digits representing seconds
- **fffff** five-digits representing fractional seconds
- **zzz** six characters representing the time zone offset (eg. +01:00)

5.4.4 TimeMulti (NXA85 and NXS85 only)

The **DateTimeMulti** data type exposes a date and time in the following format as a collection of registers.

The following list covers the Register Offset and its respective value using the scheme:

- **Register Offset** its respective value
- **0** Hour
- **1** Minutes
- **2** Seconds
- **3** Milliseconds
- **4** TimeZone Offset (in hours)

6 Appendix B: Status values

6.1 Gauge status

Bit Position	Value	Description
0	1	ConfigComplete
1	2	ConfigExecuting
2	4	Complete
3	8	Executing
4	16	Initialising
5	32	FastScan
6	64	ManualScan
7	128	OffScan

6.2 Servo status

Bit Position	Value	Description
0	1	ServoUp
1	2	ServoDown
2	4	ServoTopLimit/Stowed
3	8	ServoBottomLimit
4	16	OffLevel
5	32	WaterInterfaceMode
6	64	Testing/Stowing/Calibrating
7	128	Blocked/Frozen

6.3 System status

Bit Position	Value	Description
0	1	TestPassed
1	2	TestFailed
2	4	ConfigFailed
3	8	CommandFailed
4	16	GaugeFault
5	32	RedundancyPassive
6	64	RedundancyFixed

6.4 Alarm status

Bit Position	Value	Description
0	1	HardwareHiHiLevel
1	2	HardwareHiLevel
2	4	HardwareLoLevel

Bit Position	Value	Description
3	8	HardwareLoLoLevel
4	16	SoftwareRoofAlarm
5	32	SoftwareVariableAlarm
6	64	SoftwareDifferenceAlarm
7	128	UnauthorisedMovement
8	256	SoftwareHiHiLevel
9	512	SoftwareHiLevel
10	1024	SoftwareLoLevel
11	2048	SoftwareLoLoLevel
12	4096	SoftwareHiTemperature
13	8192	SoftwareLoTemperature
14	16384	SoftwareHiFlow
15	32768	SoftwareLoFlow
16	65536	LoDensity
17	131072	HiDensity
18	262144	LoTemperature
19	524288	HiTemperature
20	1048576	SoftwareHiDensity
21	2097152	SoftwareLoDensity
22	4194304	SoftwareDensityDeviation
23	8388608	SoftwareTempDeviation
24	16777216	SoftwareHiWater
25	33554432	SoftwareLoWater
26	67108864	TempDeviation
27	134217728	DensityDeviation
28	268435456	TimeToVariable
29	536870912	NoReplyFromGauge
30	1073741824	TheftAlarm
31	2147483648	Rollover
32	4294967296	SoftwareNoFlow
33	8589934592	UnauthorisedLevelMovement
34	17179869184	TankModeTimerAlarm
40	1099511627776	GaugeSpecificAlarm1
41	2199023255552	GaugeSpecificAlarm2
42	4398046511104	GaugeSpecificAlarm3
43	8796093022208	GaugeSpecificAlarm4
44	17592186044416	GaugeSpecificAlarm5
45	35184372088832	GaugeSpecificAlarm6
46	70368744177664	GaugeSpecificAlarm7
47	140737488355328	GaugeSpecificAlarm8
48	281474976710656	ProgUserDefinedAlarm1
49	562949953421312	ProgUserDefinedAlarm2
50	1125899906842620	ProgUserDefinedAlarm3

Bit Position	Value	Description
51	2251799813685240	ProgUserDefinedAlarm4
52	4503599627370490	ProgUserDefinedAlarm5
53	9007199254740990	ProgUserDefinedAlarm6
54	18014398509481900	ProgUserDefinedAlarm7
55	36028797018963900	ProgUserDefinedAlarm8
56	72057594037927900	SkinSensorLowTempAlarm
57	144115188075855000	SkinSensorHighTempAlarm
58	288230376151711000	SkinSensorHotSpotAlarm
59	576460752303423000	SkinSensorLeakAlarm

6.5 Short Alarm Status

Bit Position	Value	Description
0	1	HardwareHiHiAlarm
1	2	HardwareHiAlarm
2	4	HardwareLoAlarm
3	8	HardwareLoLoAlarm
4	16	SoftwareRoofAlarm
5	32	SoftwareVariableAlarm
6	64	SoftwareDifferenceAlarm
7	128	UnauthorisedMovement
8	256	SoftwareHiHiAlarm
9	512	SoftwareHiAlarm
10	1024	SoftwareLoAlarm
11	2048	SoftwareLoLoAlarm
12	4096	SoftwareHiTemperature
13	8192	SoftwareLoTemperature
14	16384	SoftwareHiFlow
15	32768	SoftwareLoFlow

6.6 Extended Alarm Status

Bit Position	Value	Description
0	1	LoDensity
1	2	HiDensity
2	4	LoTemperature
3	8	HiTemperature
4	16	SoftwareHiDensity
5	32	SoftwareLoDensity
6	64	SoftwareDensityDeviation
7	128	SoftwareTempDeviation
8	256	SoftwareHiWater

Bit Position	Value	Description
9	512	SoftwareLoWater
10	1024	TempDeviation
11	2048	DensityDeviation
12	4096	TimeToVariable
13	8192	NoReplyFromGauge
14	16384	TheftAlarm
15	32768	Rollover

6.7 Extended Alarm Status2

Bit Position	Value	Description
0	1	SoftwareNoFlow
1	2	UnauthorisedLevelMovement
2	4	TankModeTimerAlarm
3	248	SpareAlarmBits
4	256	GaugeSpecificAlarm1
5	512	GaugeSpecificAlarm2
6	1024	GaugeSpecificAlarm3
7	2048	GaugeSpecificAlarm4
8	4096	GaugeSpecificAlarm5
9	8192	GaugeSpecificAlarm6
10	16384	GaugeSpecificAlarm7
11	32768	GaugeSpecificAlarm8
12	65536	ProgUserDefinedAlarm1
13	131072	ProgUserDefinedAlarm2
14	262144	ProgUserDefinedAlarm3
15	524288	ProgUserDefinedAlarm4
16	1048576	ProgUserDefinedAlarm5
17	2097152	ProgUserDefinedAlarm6
18	4194304	ProgUserDefinedAlarm7
19	8388608	ProgUserDefinedAlarm8
20	16777216	SkinSensorLowTempAlarm
21	33554432	SkinSensorHighTempAlarm
22	67108864	SkinSensorHotSpotAlarm
23	134217728	SkinSensorLeakAlarm
24	4026531840	SpareAlarmBits1

7 Appendix C: Exported Map Example

7.1 Exported Map Example

```
< ?xmlversion = "1.0" encoding = "utf-8"? >
< !--Modbus map created at 10:20 on 24/05/2021-- >
< Root > < SlavePort Name = "Port" >
< Endianness >
< DataTypeEndianessBit > LittleEndian </ DataTypeEndian essBit >
< DataTypeEndianessChar > LittleEndian </ DataTypeEndianessChar >
< DataTypeEndianessDate > LittleEndian </ DataTypeEndianessDate >
< DataTypeEndianessDateTime > LittleEndian </ DataTypeEndianessDateTime >
< DataTypeEndianessDouble > LittleEndian </ DataTypeEndianessDouble >
< DataTypeEndianessFloat > LittleEndian </ DataTypeEndianessFloat >
< DataTypeEndianessFractional > LittleEndian </ DataTypeEndianessFractional >
< DataTypeEndianessGuid > LittleEndian </ DataTypeEndianessGuid >
< DataTypeEndianessInt > LittleEndian </ DataTypeEndianessInt >
< DataTypeEndianessLong > LittleEndian </ DataTypeEndianessLong >
< DataTypeEndianessShort > LittleEndian </ DataTypeEndianessShort >
< DataTypeEndianessTime > LittleEndian </ DataTypeEndianessTime >
< DataTypeEndianessText > LittleEndian </ DataTypeEndianessText >
< DataTypeEndianessUChar > LittleEndian </ DataTypeEndianessUChar >
< DataTypeEndianessUInt > LittleEndian </ DataTypeEndianessUInt >
< DataTypeEndianessULong > LittleEndian </ DataTypeEndianessULong >
< DataTypeEndianessUShort > LittleEndian </ DataTypeEndianessUShort >
</ Endianness >
< InvalidValuesUseInvalidValues = "false" >
< InvalidBit > false </ InvalidBit >
< InvalidChar > -128 </ InvalidChar >
< InvalidDate > Invalid </ InvalidDate >
< InvalidDateTime > Invalid </ InvalidDateTime >
< InvalidDouble > 1.7976931348623157E+308 </ InvalidDouble >
< InvalidFloat > 3.40282347E+38 </ InvalidFloat >
< InvalidFractional > 9999.9999 </ InvalidFractional >
< InvalidInt > -2147483648 </ InvalidInt >
< InvalidLong > -9223372036854775808 </ InvalidLong >
< InvalidShort > -32768 </ InvalidShort >
< InvalidTime > Invalid </ InvalidTime >
< InvalidText > invalid </ InvalidText >
< InvalidUChar > 255 </ InvalidUChar >
< InvalidUInt > 4294967295 </ InvalidUInt >
< InvalidULong > 18446744073709551615 </ InvalidULong >
```

< InvalidUShort > 65535 </ InvalidUShort >

</ InvalidValues >

< SlaveName = "Slave"SlaveAddress = "1" >

< Coil/ >

< HoldingRegister >

< fieldaddress = "40001"XAddress = "0x9C41"name = "ProductLevel.Value"objectType = "TFA_Common_Data_Tank" object = "TK101"dataType = "Double"multipli

< fieldaddress = "40005"XAddress = "0x9C45"name = "ProductTemperature.Value"objectType = "TFA_Common_Data_Tank"object = "TK101"dataType = "Double"m

< fieldaddress = "40009"XAddress = "0x9C49"name = "ProductLevel.Value"objectType = "TFA_Common_Data_Tank" object = "TK202"dataType = "Double"multipli

< fieldaddress = "40013"XAddress = "0x9C4D"name = "ProductTemperature.Value"objectType = "TFA_Common_Data_Tank"object = "TK202"dataType = "Double"m

< fieldaddress = "40017"XAddress = "0x9C51"name = "ProductLevel.Value"objectType = "TFA_Common_Data_Tank" object = "TK303"dataType = "Double"multipli

< fieldaddress = "40021"XAddress = "0x9C55"name = "ProductTemperature.Value"objectType = "TFA_Common_Data_Tank"object = "TK303"dataType = "Double"m

< fieldaddress = "40025"XAddress = "0x9C59"name = "ProductLevel.Value"objectType = "TFA_Common_Data_Tank" object = "TK404"dataType = "Double"multipli

< fieldaddress = "40029"XAddress = "0x9C5D"name = "ProductTemperature.Value"objectType = "TFA_Common_Data_Tank"object = "TK404"dataType = "Double"m

</ HoldingRegister >

< InputRegister/ >

< InputStatus/ >

</ Slave >

</ SlavePort >

</ Root >

8 Appendix D: Modbus Template Index

8.1 Composition Templates

8.1.1 Composition by id (double)

FileName
Composition_by_id_double.xml

Notes
Allows composition to be set using component ID’s. (Assumes text uses 16-bit characters)

1. Enter composition Id.
2. Enter component details.
3. Verify validity.
4. Write to ’SaveComposition’.

Item Table

O [1)]	N [2)]	W [3)]	A [4)]	D [5)]	R [6)]
1	CompositionName	X	0	Text	12
2	ComponentId_1	X	0	Int	2
3	ComponentMoleFraction_1	X	0	Double	4
4	ComponentId_2	X	0	Int	2
5	ComponentMoleFraction_2	X	0	Double	4
6	ComponentId_3	X	0	Int	2
7	ComponentMoleFraction_3	X	0	Double	4
8	ComponentId_4	X	0	Int	2
9	ComponentMoleFraction_4	X	0	Double	4
10	ComponentId_5	X	0	Int	2
11	ComponentMoleFraction_5	X	0	Double	4
12	ComponentId_6	X	0	Int	2
13	ComponentMoleFraction_6	X	0	Double	4
14	ComponentId_7	X	0	Int	2
15	ComponentMoleFraction_7	X	0	Double	4
16	ComponentId_8	X	0	Int	2
17	ComponentMoleFraction_8	X	0	Double	4
18	ComponentId_9	X	0	Int	2
19	ComponentMoleFraction_9	X	0	Double	4
20	ComponentId_10	X	0	Int	2
21	ComponentMoleFraction_10	X	0	Double	4
22	MoleFractionsAreValid	-	0	Bit	1
23	IsValid	-	0	Bit	1
24	SaveComposition	X	0	None	1
25	SaveSuccess	-	0	Bit	1

1) Offset
2) Name
3) Writeable
4) ArraySize
5) Datatype
6) Registers

8.1.2 Composition by id (float)

FileName
Composition_by_id_float.xml

Notes
Allows composition to be set using component ID's. (Assumes text uses 16-bit characters)

1. Enter composition Id.
2. Enter component details.
3. Verify validity.
4. Write to 'SaveComposition'.

Item Table

O [1]	N [2]	W [3]	A [4]	D [5]	R [6]
1	CompositionName	X	0	Text	12
2	ComponentId_1	X	0	Int	2
3	ComponentMoleFraction_1	X	0	Float	2
4	ComponentId_2	X	0	Int	2
5	ComponentMoleFraction_2	X	0	Float	2
6	ComponentId_3	X	0	Int	2
7	ComponentMoleFraction_3	X	0	Float	2
8	ComponentId_4	X	0	Int	2
9	ComponentMoleFraction_4	X	0	Float	2
10	ComponentId_5	X	0	Int	2
11	ComponentMoleFraction_5	X	0	Float	2
12	ComponentId_6	X	0	Int	2
13	ComponentMoleFraction_6	X	0	Float	2
14	ComponentId_7	X	0	Int	2
15	ComponentMoleFraction_7	X	0	Float	2
16	ComponentId_8	X	0	Int	2
17	ComponentMoleFraction_8	X	0	Float	2
18	ComponentId_9	X	0	Int	2
19	ComponentMoleFraction_9	X	0	Float	2
20	ComponentId_10	X	0	Int	2
21	ComponentMoleFraction_10	X	0	Float	2
22	MoleFractionsAreValid	-	0	Bit	1
23	IsValid	-	0	Bit	1
24	SaveComposition	X	0	None	1
25	SaveSuccess	-	0	Bit	1

1) Offset
2) Name
3) Writeable
4) ArraySize
5) Datatype
6) Registers

8.1.3 Composition by id (scaled short)

FileName
Composition_by_id_scaled_short.xml

Notes
Allows composition to be set using component ID's. (Assumes text uses 16-bit characters)

1. Enter composition Id.
2. Enter component details.
3. Verify validity.
4. Write to 'SaveComposition'.

Item Table

O [1)]	N [2)]	W [3)]	A [4)]	D [5)]	R [6)]
1	CompositionName	X	0	Text	12
2	ComponentId_1	X	0	UShort	1
3	ComponentMoleFraction_1	X	0	UShort	1
4	ComponentId_2	X	0	UShort	1
5	ComponentMoleFraction_2	X	0	UShort	1
6	ComponentId_3	X	0	UShort	1
7	ComponentMoleFraction_3	X	0	UShort	1
8	ComponentId_4	X	0	UShort	1
9	ComponentMoleFraction_4	X	0	UShort	1
10	ComponentId_5	X	0	UShort	1
11	ComponentMoleFraction_5	X	0	UShort	1
12	ComponentId_6	X	0	UShort	1
13	ComponentMoleFraction_6	X	0	UShort	1
14	ComponentId_7	X	0	UShort	1
15	ComponentMoleFraction_7	X	0	UShort	1
16	ComponentId_8	X	0	UShort	1
17	ComponentMoleFraction_8	X	0	UShort	1
18	ComponentId_9	X	0	UShort	1
19	ComponentMoleFraction_9	X	0	UShort	1
20	ComponentId_10	X	0	UShort	1
21	ComponentMoleFraction_10	X	0	UShort	1
22	MoleFractionsAreValid	-	0	Bit	1
23	IsValid	-	0	Bit	1
24	SaveComposition	X	0	Bit	1
25	SaveSuccess	-	0	Bit	1

1) Offset
2) Name
3) Writeable
4) ArraySize
5) Datatype
6) Registers

8.1.4 Composition by name (double)

FileName
Composition_by_name_double.xml

Notes
Allows composition to be set using component ID’s. (Assumes text uses 16-bit characters)

1. Enter composition name.
2. Enter component details.
3. Verify validity.
4. Write to ’SaveComposition’.

Item Table

O [1]	N [2]	W [3]	A [4]	D [5]	R [6]
1	CompositionName	X	0	Text	12
2	ComponentName_1	X	0	Text	12
3	ComponentMoleFraction_1	X	0	Double	4
4	ComponentName_2	X	0	Text	12
5	ComponentMoleFraction_2	X	0	Double	4
6	ComponentName_3	X	0	Text	12
7	ComponentMoleFraction_3	X	0	Double	4
8	ComponentName_4	X	0	Text	12
9	ComponentMoleFraction_4	X	0	Double	4
10	ComponentName_5	X	0	Text	12
11	ComponentMoleFraction_5	X	0	Double	4
12	ComponentName_6	X	0	Text	12
13	ComponentMoleFraction_6	X	0	Double	4
14	ComponentName_7	X	0	Text	12
15	ComponentMoleFraction_7	X	0	Double	4
16	ComponentName_8	X	0	Text	12
17	ComponentMoleFraction_8	X	0	Double	4
18	ComponentName_9	X	0	Text	12
19	ComponentMoleFraction_9	X	0	Double	4
20	ComponentName_10	X	0	Text	12
21	ComponentMoleFraction_10	X	0	Double	4
22	MoleFractionsAreValid	-	0	Bit	1
23	IsValid	-	0	Bit	1
24	SaveComposition	X	0	None	1
25	SaveSuccess	-	0	Bit	1

1) Offset
2) Name
3) Writeable
4) ArraySize
5) Datatype
6) Registers

8.1.5 Composition by name (float)

FileName
Composition_by_name_float.xml

Notes
Allows composition to be set using component ID’s. (Assumes text uses 16-bit characters)

1. Enter composition name.
2. Enter component details.
3. Verify validity.
4. Write to ’SaveComposition’.

Item Table

O 1)	N 2)	W 3)	A 4)	D 5)	R 6)
1	CompositionName	X	0	Text	12
2	ComponentName_1	X	0	Text	12
3	ComponentMoleFraction_1	X	0	Float	2
4	ComponentName_2	X	0	Text	12
5	ComponentMoleFraction_2	X	0	Float	2
6	ComponentName_3	X	0	Text	12
7	ComponentMoleFraction_3	X	0	Float	2
8	ComponentName_4	X	0	Text	12
9	ComponentMoleFraction_4	X	0	Float	2
10	ComponentName_5	X	0	Text	12
11	ComponentMoleFraction_5	X	0	Float	2
12	ComponentName_6	X	0	Text	12
13	ComponentMoleFraction_6	X	0	Float	2
14	ComponentName_7	X	0	Text	12
15	ComponentMoleFraction_7	X	0	Float	2
16	ComponentName_8	X	0	Text	12
17	ComponentMoleFraction_8	X	0	Float	2
18	ComponentName_9	X	0	Text	12
19	ComponentMoleFraction_9	X	0	Float	2
20	ComponentName_10	X	0	Text	12
21	ComponentMoleFraction_10	X	0	Float	2
22	MoleFractionsAreValid	-	0	Bit	1
23	IsValid	-	0	Bit	1
24	SaveComposition	X	0	None	1
25	SaveSuccess	-	0	Bit	1

1) Offset
2) Name
3) Writeable
4) ArraySize
5) Datatype
6) Registers