

Smartware Jar File ***for the*** ***Niagara R2 Platform***

Programmer's Guide

Release 2.301.525.v0041
February 2010



Smartware Technologies, Inc.
4 Pequet Parkway
Tonawanda, NY 14150

Sales and Support
(716) 213-2222

<http://www.smartwaretech.com>

All material is Copyright © 2007-2009 Smartware Technologies. All rights reserved.

WorkPlace Pro is a trademark of Tridium, Incorporated. Microsoft is a registered trademark and Visio is a trademark of Microsoft Corporation. TAC and I/A Series are registered trademarks of TAC. Designer Suite 2005 is a trademark of Smartware Technologies, Inc.

Table of Contents

Overview of the Smartware R2 Jar File	5
Disclaimer and Limitation of Liability	5
Obtaining the Smartware Jar File.....	5
Licensing the Smartware Jar File	6
Versions of the Smartware Jar File	7
Installing the Smartware Jar File into WorkPlace Pro	7
Release Notes.....	7
GCM Blocks	9
Data Types.....	9
EMPTY and NOT ACTIVE values	9
The GCM Blocks	9
ALARM.....	11
APT	12
COS	13
DPT	14
EDL	15
ENTHL	16
LOOP	17
OSS.....	18
OSSX	25
RNET	26
SEQ.....	27
SLECT, SLECT2	28
TREND.....	29
Conversion Blocks	31
Enhanced Smartware Blocks	33
SwAnalogSetpoint, SwBinarySetpoint.....	34
SwCmdString	37
SwFlexString, SwFlexString4, SwFlexString16	38
SwGxText.....	40
Utility Blocks.....	43
SlidingMinMax	45
SlidingSetpoint	48
ValueAnalyzer	50
ValueComparer	52
ValueCounter	54
ValueTotalizer	56

Overview of the Smartware R2 Jar File

The Smartware R2 Jar File contains a set of custom, compiled blocks in a number of major categories:

gcm	Mimics the functionality of many of the Network 8000 GCM/LCM program blocks, including a true OSS.
sw	Provides advanced functionality blocks, such as setpoints, and enhanced versions of core Niagara blocks, such as the GxText object.
conversion	Allows one-to-one conversions of specific data types to another, including basic types and Lon Snvt types.
objects	Provides a set of common utility function blocks

Refer to the later sections for more details on each category and block.

Disclaimer and Limitation of Liability

The Smartware R2 Jar file is sold and used with a Disclaimer and Limitation of Liability that does not guarantee its success for any specific purpose or provide for any recourse for any damages or loss arising from its use.

If you wish to review these legal notices before proceeding, please contact our office before installing or using the jar file..

Obtaining the Smartware Jar File

The latest version of the Smartware Jar file can be obtained only from Smartware Technologies. To avoid confusion, TAC will not be distributing the jar file.

- If we converted a Network 8000 project for you, the Jar file will be included in the files we send you.
- Eventually the Jar File will be available for download from our web site.

Licensing the Smartware Jar File

For a customer's UNC or Enterprise Server, you must include the IA-DRV-SMARTWARE driver feature in your license, either at the time it is ordered or by requesting an upgrade. TAC provides all Niagara licenses.

- If you have an existing UNC or Enterprise Server license, you will need to upgrade it to include the Smartware feature.
- The cost to include a license for the Smartware jar file in a Niagara license from TAC is about \$300 (net).
- There is no cost to upgrade TIP copies of WorkPlace Pro to run the Smartware R2 Jar File for testing and development purposes.
- The I/A License Forms have been updated (as of version 6.1) to include a place to order the Smartware Jar File (IA-DRV-SMARTWARE):

<input type="checkbox"/> IA-DRV-1001-P (BALnet 1001P - First Trunk)	<input type="checkbox"/> IA-DRV-1002-P (Enhanced 1002 Driver - 50 Controllers Second Trunk)
<input type="checkbox"/> IA-DRV-SNMP-P (SNMP Driver)	<input type="checkbox"/> IA-DRV-MS124 (Enhanced MS Driver - 124 Controllers Second Trunk)
*Note: For configurations involving more than 2 RS-485 trunks, contact Customer Service for compatability and availability.	
Additional License Features - Check required features	
<input type="checkbox"/> IA-DRV-SMARTWARE (Smartware JAR File)*	
*Note: To enable this feature on older devices (UNC-510, 600 or 610) use this form but make note of device type below.	
License Information	Please Print Clearly
Organization ID:	

For a technician machine running a TIP copy of WorkPlace Pro, no explicit licensing should be required. However, this ability relies on how the license was first issued by TAC. If the Jar file does not function on a TIP license, or if you want to verify that it will, please e-mail us a copy of your license file (from WorkPlace Pro).

To generate the license file in WorkPlace Pro:

1. Open the Admin Tool
2. Select the INSTALLATION Tab
3. Click the VIEW LICENSE button.
4. From the FILE menu, select SAVE LOCAL

Versions of the Smartware Jar File

The name of the Smartware Jar File is:

Smartware-2.301.525.v00xx.jar

Where xx represents our revision number. The "2.301.525" in the name *does not* imply that it works only with that release of Niagara. In fact, the Smartware Jar File will work with Niagara release r2.301.514 and later, and in most cases will work with the r2.301.4xx releases..

From time to time we will release updated releases of the Smartware Jar File, which will differ only by the "v00xx" number.

Installing the Smartware Jar File into WorkPlace Pro

To install the Smartware Jar File, copy the file to the standard folders in WorkPlace Pro:

C:\Niagara\r2.301.5xx\emb
C:\Niagara\r2.301.5xx\nt
C:\Niagara\r2.301.5xx\nre\modules

Unlike some other jar files, there is only one file and it can be copied into all three folders.

Be sure there is only one version of the Smartware Jar File in a folder at any time.

Release Notes

The following changes and updates have been made to the jar file:

- v0036 Introduced the OSS block
- v0037 Updated the MTR block. With MTTYP = ELAPSED TIME, when the DINP went from OFF to ON the block was still resetting the counters, emitting the MESSG, and setting DV to false. Though the GCM Reference Manual indicates that at least the MESSG should be emitted, logically this behavior should really be only for the DIGITAL STATE mode. The block has been updated to reflect this.
- v0038 Updated the OSS block. If the next scheduled event in the linked Schedule was the same state as the current state (e.g., while OFF, the next event is also an OFF state), the OSS would prestart (or prestop) incorrectly.
- v0040 Updated the HILO block. While faulted values on the input would be ignored, down and out-of-service values were treated as zeros. This has been corrected.
- v0041 Added the EDL2 block. This is the exact same block as EDL except the SHED output is a FloatStatus (EDL2) instead of an IntegerStatus (EDL).

GCM Blocks

The GCM blocks were developed to work in conjunction with the Smartware Studio Network 8000 Conversion Tool, which can translate existing GCM devices (along with Signal screens) into Niagara R2 stations with logic, device shadow objects, schedules and links.

In order to make this conversion tool possible, it was necessary for us to create a set of R2 blocks that correspond to many of the GCM blocks. While some native R2 blocks were similar (e.g., MATH and LOGIC), there were still some differences. And some (EDL and SEQ, for example) had nothing even close in the native R2 system.

In order to avoid excessive use of Program Objects, which are highly discouraged by experienced Niagara R2 programmers and Tridium alike, we created our own compiled version of these blocks.

Data Types

GCM Analog Values (AV) are translated to *FloatStatus* values.
GCM Digital Values (DV) are translated to *BooleanStatus* values.

EMPTY and NOT ACTIVE values

For Analog Values (AV), EMPTY and NOT ACTIVE values are represented by an *NaN* value. For Digital Values (DV), EMPTY and NOT ACTIVE values are represented by an *Inactive* value with the *InFault* status flag turned on.

The GCM Blocks

For the most part, the GCM blocks in the Smartware Jar File were designed to look and behave the same as their counterparts in the GCM itself. In most cases, the inputs, outputs and parameters have the same names, data types and range of values. Please refer to the *Network 8000 GCM/LCM Programmer's Manual* (TAC document F-23120) as your primary source of reference.

The following blocks are included:

ALARM	EDL	PWM
AO	ENTHL	RAMP
APT	FLOW	RESET
BTUH	HILO	RLCM
CALEN	HOLI	RNET
COS	LIMIT	SEQ
DAILY	LOGIC	SLECT
DCC	LOOP	TOTAL
DEGDA	MATH	TREND
DELAY	MTR	WEEK
DO	OSS	
DPT	POLL	

The following pages describe any specific differences between specific GCM blocks in the Smartware Jar File and the originals in the GCM.

ALARM

Mimics the operation of the GCM ALARM block.

Class Name: `tridiumx.smartware.gcm.ALARM`

Remarks

The ALARM block does not actually generate a Niagara alarm. The *STATE* output should be connected to an *BinaryInput* with the following properties set:

<code>eventEnable.to-offnormal</code>	True
<code>alarmText</code>	The alarm message
<code>alarmValue</code>	Active
<code>alarmValueEnabled</code>	True

APT

Mimics the operation of the GCM APT block and also provides a mechanism for implementing a true Enterprise Server to UNC setpoint with timed override

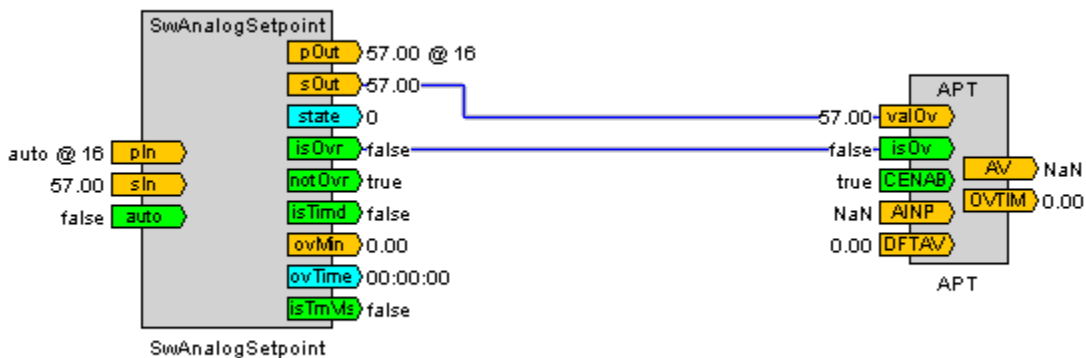
Class Name: `tridiumx.smartware.gcm.APT`

INPUTS	Data Type	Default	Description
OverrideValue [valOv]	BooleanStatus	False (fault)	The overriding value
IsOverridden [isOv]	BooleanStatus	False	If true, then the DV output is set to <i>OverrideValue</i> .

Remarks

When used as a setpoint that has an EMPTY or NA value as its default, you should put the NaN value on the *defaultAINP* property. The NaN value will be treated as an EMPTY value to other GCM blocks, such as the AO.

The *OverrideValue* and *IsOverridden* values link to the *statusOutput* and *IsOverride* outputs of an *SwAnalogSetpoint* to create a true Enterprise Server to UNC setpoint with timed override.



In this example, the *defaultAINP* property is set to NaN, so it passes through the AV. It also shows how the *SwAnalogSetpoint* object would be linked, though usually the *SwAnalogSetpoint* would be in the Enterprise Server (linked to a GxText for user commandability) and the APT would reside in the UNC, with the two linked by external subscription links.

COS

Mimics the operation of the GCM COS block.

Class Name: `tridiumx.smartware.gcm.COS`

Remarks

The COS block does not actually generate a Niagara alarm. The *STATE* output should be connected to an *BinaryInput* with the following properties set:

<i>eventEnable.to-offnormal</i>	True
<i>alarmText</i>	The alarm message
<i>alarmValue</i>	Active
<i>alarmValueEnabled</i>	True

DPT

Mimics the operation of the GCM DPT block and also provides a mechanism for implementing a true Enterprise Server to UNC setpoint with timed override

Class Name: *tridiumx.smartware.gcm.DPT*

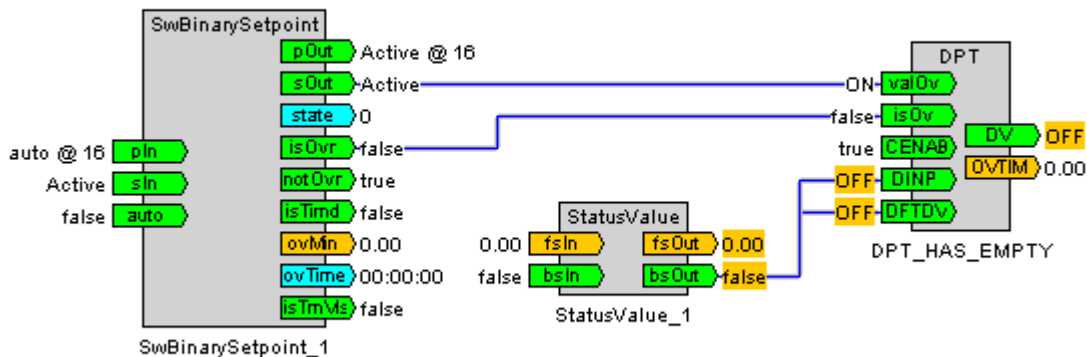
INPUTS	Data Type	Default	Description
OverrideValue [valOv]	BooleanStatus	False (fault)	The overriding value
IsOverridden [isOv]	BooleanStatus	False	If true, then the DV output is set to <i>OverrideValue</i> .

Remarks

When used as a setpoint that has an EMPTY or NA value as its default, you must put a faulted value on the DINP and DFTDV inputs. As per the original GCM block, the faulted value on DINP is not passed through to DV, but instead causes the block to use the value on the DFTDV input as DV (after any DELAY), which will pass through the fault status that other blocks (such as the DO) will treat as EMPTY.

You can generate a faulted Boolean status value with the *StatusValue* object in this jar file.

The *OverrideValue* and *IsOverridden* values link to the *statusOutput* and *IsOverride* outputs of an *SwBinarySetpoint* to create a true Enterprise Server to UNC setpoint with timed override.



In this example, the *StatusValue* object is being used to make the default (non-overridden) value of the DPT a faulted (EMPTY) value. It also shows how the *SwBinarySetpoint* object would be linked, though usually the *SwBinarySetpoint* would be in the Enterprise Server (linked to a *GxText* for user commandability) and the DPT would reside in the UNC, with the two linked by external subscription links.

EDL, EDL2

Mimics the operation of the GCM EDL block.

Class Name: `tridiumx.smartware.gcm.EDL`

Remarks

The *LD5 to LD64* inputs and the *PLS5 to PLS64* outputs are available for linking, but are not shown on the object by default. They will appear automatically if linked.

For the *PLS* outputs, the value is *Inactive* when the corresponding load should be shed, and *InFault* (with a value equal to the *NotActiveValue* parameter) if the load should not be.

EDL2

In the original EDL block the SHED output is an IntegerStatus value. To conform to the system used by the Smartware Network 8000 Conversion Service, the EDL2 block has this output as a FloatStatus. Otherwise the blocks are functionally equivalent.

Exceptions

It is unclear from the GCM documentation as to what the *Pulse Count (PLCOU)* input should do. In our implementation, this input is ignored.

The *GCM/LCM Programmer's Manual* (page 4-134) states that:

If the SLIDING WINDOW is the same size or larger than the DEMAND INTERVAL, the EDL block only starts shedding loads after the PEAK DEMAND (which is recorded by the power company) has occurred. The SLIDING WINDOW must be smaller than the DEMAND INTERVAL to prompt the EDL block to reduce the peaks.

It is unclear as to what this means, so this feature has not been implemented. The load shedding algorithm operates regardless of whether SLIDING WINDOW is smaller or larger than the DEMAND INTERVAL.

ENTHL

Mimics the operation of the GCM ENTHL block.

Class Name: *tridiumx.smartware.gcm.ENTHL*

Remarks

The ENTHL block only calculates for the RELATIVE HUMIDITY mode. Therefore, the HSSEL parameter and the DEWIN, BPRIN and BPRMM inputs are ignored.

The IUNIT parameter has been changed to a choice between DEG F (default) and DEG C. As with the original ENTHL block, when DEG C is selected the temperature inputs and outputs are in °C (instead of °F) and the AV output is in Kj/Kg (instead of BTU/pound).

LOOP

Mimics the operation of the GCM LOOP block.

Class Name: *tridiumx.smartware.gcm.LOOP*

Remarks

The LOOP block does not have a self-tune feature. Therefore, the ENCHG input is ignored.

All other functionality is implemented.

OSS

Mimics the operation of the GCM OSS block, including the table-driven Optimal Start and Stop functionality.

DISCLAIMER

As of version v0035, the OSS block is considered a beta version that is still being tested. If you use the block in a live system, you should monitor its behavior very closely to ensure that the systems are functioning properly. It is not recommended that you use the OSS block in critical situations. You should also connect a string log to the Message output as described below so that problems that do arise can be evaluated and corrected.

Class Name: `tridiumx.smartware.gcm.OSS`

Remarks

Connecting the Schedule Object

The block needs to be linked to an existing Niagara Schedule object as follows

Schedule Outputs	OSS Inputs
<i>statusOutput</i>	<i>Schedule [Sched]</i>
<i>nextEventTime</i>	<i>nextEventTime [nxtTime]</i>
<i>nextEventValue</i>	<i>nextEventValue [nxtVal]</i>

Schedule Inputs	OSS Outputs
<i>holidayOverride</i>	<i>holidayOverride [hday]</i>

Note that:

- The VACxx and OCCxx parameters do not exist. The schedule is external.
- The HOSCH input is passed through as the *holidayOverride* output. It should be linked to the *holidayOverride* input of the Niagara schedule.

The Diagnostic Message Log

The block has an extra output named *Message* which emits a string containing information about the current state of the block and noting when it changes state, achieves its setpoint or abandons a calculation.

It is highly recommend that you create a *StringLog* object and connect it to the Message output to capture this information to aid in debugging any problems that may arise in certain situations. If the system is configured for archiving, you should ensure that this log is archiving its records as well.

The table that indicates the number of minutes to prestart or prestop based on the outside air temperature is automatically dumped out through the Message output at the time specified by the *DumpTableTime* parameter (which defaults to 12:00 midnight).

Diagnostic Inputs/Outputs

INPUTS	Data Type	Default	Description
DiagResetAll [reset]	BooleanStatus	False	When set to True, the entire block, including the TABLE, is reset.
DiagDisableOss [dsblOss]	BooleanStatus	False	When set to True, the OSS algorithm is ignored and the block generally passes the current value of the <i>Schedule</i> input to the <i>Enable Outputs</i> , essentially mimicking the functionality of the OSSX block.

The OSS Algorithm

The OSS attempts to recreate the algorithm used by the GCM block. A full discussion of the algorithm follows.

- For this discussion, the *Enable Outputs* are: COENA, DAENA, FAENA, HENFL and CENFL

Abnormal Inputs

If any of the following are true, the message is ABNORMAL INPUT and all Enable Outputs are False and Faulted:

- OAT is invalid
- SPACE is invalid
- HENA and FENA are both True

Forced Inputs

If the FOENA input is True, all the *Enable Outputs* are set to FOVAL. No other outputs are updated until FOENA returns to False.

When FOENA returns to False, the state of the OSS becomes *Unknown* and the block restarts its algorithm.

Heating/Cooling Mode

The OSS block determines whether it is in a *Heating* or *Cooling* mode by checking the following conditions (in order);

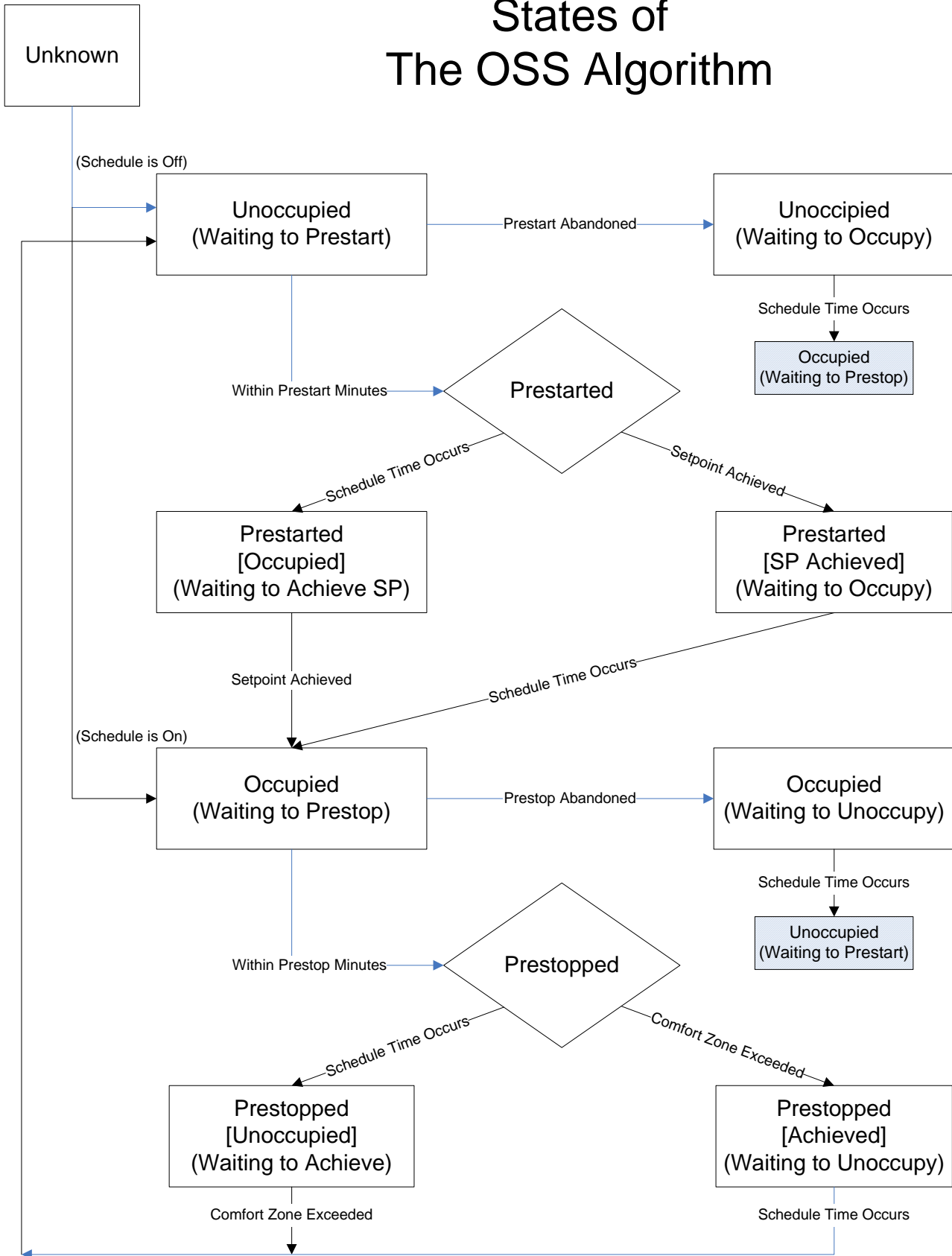
- If HENA is On, the mode is *Heating*
- If CENA is On, the mode is *Cooling*
- If $OAT < HCHGO$, the mode is *Heating*
- If $OAT > CCHGO$, the mode is *Cooling*
- Otherwise the mode is *No Heating or Cooling* and the OSS algorithm is not performed.

The States of the OSS

The OSS cycles through five distinct, general states:

<i>Unknown</i>	The block is initializing or restarting and is trying to determine the correct state.
<i>Unoccupied (Waiting to Prestart)</i>	The Schedule input is off and the OSS block is monitoring the inputs and tables to determine when to Prestart.
<i>Prestarted</i>	The OSS has prestarted, and is monitoring the inputs to determine if and when the setpoint is achieved.
<i>Occupied (Waiting to Prestop)</i>	The Schedule input is on and the OSS is monitoring the inputs and tables to determine when to Prestop.
<i>Prestopped</i>	The OSS has prestopped, and is monitoring the inputs to determine how long it takes for the Comfort Zone to be exceed.

States of The OSS Algorithm



State: *Unknown*

The OSS blocks checks the current state of the *Schedule* input:

- If *Schedule* is On, the state becomes *Occupied (Waiting to Prestop)*
- If *Schedule* is Off, the state becomes *Unoccupied (Waiting to Prestart)*

State: *Unoccupied (Waiting to Prestart)*

During this state, all the *Enable Outputs* are set to False.

The following logic rules apply (in order):

- If the *Schedule* input goes On, the state is changed to *Occupied (Waiting to Prestop)*
- If the mode is *No Heating or Cooling*, or if $MXSTA = 0$, no other calculations are made
- The OSS gets the Table Entry based on the current OAT value and determines the prestart minutes based on the Heating/Cooling mode. The prestart minutes are limited to $MXSTA$.
- If the next schedule event is within the determined prestart minutes, the state is changed to *Prestarted*.
- Otherwise, the $TMSTR$ is updated to reflect the time remaining until calculated prestart and the block continues waiting in this state.

State: *Prestarted*

During this states, the *Enable Outputs* are set as follows:

- $COENA \rightarrow$ On
- $DAENA \rightarrow$ Off
- $FAENA \rightarrow$ On
- $HENFL \rightarrow$ On if Heating/Cooling Mode is *Heating*
- $CENFL \rightarrow$ On if Heating/Cooling Mode is *Cooling*

The following logic rules apply (in order):

- If the calculated Heating/Cooling mode has changed, the prestart is abandoned.
- If the *Schedule* input goes On and then goes Off (indicating that the prestart failed to achieve its goals before the scheduled occupied period expired), the prestart is abandoned.

- The block monitors the inputs to determine if the Setpoint has been achieved:
 - If SP is valid:
 - If SPACE reaches SP ($\text{SPACE} \geq \text{SP}$ (for Heating) or $\text{SPACE} \leq \text{SP}$ (for Cooling)), then the Setpoint has been achieved
 - If SP is invalid:
 - SPACE must rise (Heating) or fall (Cooling) at least 0.5 degrees within MXDLY in order to continue to consider the prestart. If MXDLY passes without a 0.5 degree change, the prestart is abandoned.
 - SPACE is monitored every 10 minutes. If SPACE changes less than 1 degree over the 10 minute period, then it is assumed that Setpoint has been achieved.
- If the Setpoint has been achieved, the number of minutes it took to reach the setpoint is calculated and that value inserted into the TABLE. The value is marked as 'calculated' (as opposed to 'interpolated'), and the interpolated values above and below it in the table column are recalculated. If the Schedule input is still off, the state is changed to *Prestarted (Achieved, Waiting to Occupy)*. If not, the state is changed to *Occupied (Waiting to Prestop)*.
- Otherwise, the block continues to monitor the inputs.

If the prestart is abandoned:

- If the schedule has come on, the state changes to *Occupied (Waiting to Prestop)*.
- If the schedule is still off, the state changes to *Unoccupied (Waiting to Occupy)*

State: *Occupied (Waiting to Prestop)*

During this state, all the *Enable Outputs* are set to True.

The following logic rules apply (in order):

- If the *Schedule* input goes Off, the state is changed to *Unoccupied (Waiting to Prestart)*
- If the mode is *No Heating or Cooling*, or if $\text{MXSTP} = 0$, no other calculations are made
- The OSS gets the Table Entry based on the current OAT value and determines the prestop minutes based on the Heating/Cooling mode. The prestop minutes are limited to MXSTP.
- If the next schedule event is within the determined prestart minutes, the state is changed to *Prestopped*.
- Otherwise, the TMSTP is updated to reflect the time remaining until calculated prestop and the block continues waiting in this state.

State: *Prestopped*

During this state, the *Enable Outputs* are set as follows:

- COENA → Off
- DAENA → On
- FAENA → On
- HENFL → Off
- CENFL → Off

The following logic rules apply (in order):

- If the calculated Heating/Cooling mode has changed, the prestop is abandoned.
- If the *Schedule* input goes Off and then goes On (indicating that the prestop did not result in a loss of comfort before the next scheduled occupancy period), the prestop is abandoned.
- The block monitors the inputs to determine if the Comfort Zone has been exceeded:
 - If SP is valid:
 - If $SPACE < SP - COMFT$ (for Heating) or $SPACE > SP + COMFT$ (for Cooling), then the Comfort Zone has been exceeded.
 - If SP is invalid:
 - The value of SPACE at the moment the prestop occurred ($SPACE_{prestop}$) is treated as the setpoint. Then, if $SPACE < SPACE_{prestop} - COMFT$ (for Heating) or $SPACE > SPACE_{prestop} + COMFT$ (for Cooling), then the Comfort Zone has been exceeded.
- If the Comfort Zone has been exceeded, the number of minutes it took to reach the discomfort is calculated and that value inserted into the TABLE. The value is marked as 'calculated' (as opposed to 'interpolated'), and the interpolated values above and below it in the table column are recalculated. If the Schedule input is still on, the state is changed to *Prestopped (Achieved, Waiting to Unoccupy)*. If not, the state is changed to *Unoccupied (Waiting to Prestart)*.
-
- Otherwise, the block continues to monitor the inputs.

If the prestop is abandoned:

- If the schedule has gone off, the state changes to *Unoccupied (Waiting to Prestart)*.
- If the schedule is still on, the state changes to *Occupied (Waiting to Unoccupy)*

OSSX

A placeholder for the GCM OSS block. It does not implement the Optimal Start and Stop functionality.

Class Name: `tridiumx.smartware.gcm.OSSX`

Remarks

When hooked up to a schedule, the block can work as a placeholder for an OSS block, with important differences.

- There is no optimal start and stop calculation.
- All five outputs (COENA, DAENA, FAENA, HENFL, CENFL) follow the state of the schedule, and are always the same.
- The FOVAL and FOENA inputs are functional and force all five outputs accordingly.
- The STRTM, TMSTR, STPTM and TMSTP outputs are calculated properly.
- The VACxx and OCCxx parameters do not exist. The schedule is external.
- The HOSCH input is passed through as the *holidayOverride* output. It should be linked to the *holidayOverride* input of the Niagara schedule.
- The MXSTA, MXSTP, HCHGO, CCHGO, COMFT and MXDLY parameters are ignored.
- The SPACE, SP, OAT, HENA, CENA and FRTAB inputs are ignored.
- The HTSTR, HTSTP, CLSTR, CLSTP and TBTMP outputs are not calculated.

The Schedule object should be linked to the OSSX block as follows:

Schedule Outputs	OSSX Inputs
<i>statusOutput</i>	<i>Schedule</i>
<i>nextEventTime</i>	<i>nextEventTime</i>
<i>nextEventValue</i>	<i>nextEventValue</i>

Schedule Inputs	OSSX Outputs
<i>holidayOverride</i>	<i>holidayOverride</i>

RNET

Simulates a GCM RNET block.

Class Name: *tridiumx.smartware.gcm.RNET*

INPUTS	Data Type	Default	Description
AnalogInput [avIn]	FloatStatus	NaN	The input value if <i>DATYP</i> = ANALOG VALUE
DigitalInput [dvIn]	BooleanStatus	False	The input value if <i>DATYP</i> = DIGITAL VALUE

Remarks

The *GCNUM*, *AVNUM* and *DVNUM* properties which express where the RNET block should get its value, are actually ignored and are included for documentation purposes.

Instead, this **RNET** block takes its inputs from the new *AnalogInput* and *DigitalInput* inputs. These inputs should be linked to the Niagara equivalent of the appropriate slot in the target SNET block in the other GCM.

If the SNET slot contains a constant value, it should be set as the *defaultAnalogInput* or *defaultDigitalInput* property in this RNET block.

SEQ

Mimics the operation of the GCM SEQ block.

Class Name: `tridiumx.smartware.gcm.SEQ`

Remarks

The *ONn* and *OFFn* outputs are available for linking, but are not shown on the object by default. They will appear automatically if linked.

The VERNIER function is not supported, so the *VERNR* parameter is ignored.

The *DELON* and *DELOF* parameters are specified in seconds.

The *ROTIM* parameter is specified in minutes.

All delays are effectively the *minimum* amount of the delay, and are likely to be a few seconds longer due to the update cycle time of the Niagara framework.

SLECT, SLECT2

Mimics the operation of the GCM SEQ block.

Class Name: *tridiumx.smartware.gcm.SLECT*
tridiumx.smartware.gcm.SLECT2

Remarks

The block can select both an Analog and Digital value at the same time (it has both sets of (*AINP1/AINP2/AV* and *DINP1/DINP2/DV*) of inputs and outputs). Therefore, it does not need (or have) a *DATYP* parameter.

SLECT2

In the original SLECT block, you can specify a default AINP value of NaN to represent an EMPTY or NOT ACTIVE value that will pass through to AV. However, there was no way to have a default DINP value in a faulted state to represent an EMPTY digital value (though you can put a faulted value on *DINP1* or *DINP2* explicitly using the **StatusValue** object).

To resolve this case, the SLECT2 version adds the *IN1Empty* and *IN2Empty* parameters. If either is set to True, the corresponding input value will be treated as EMPTY, and if selected, the outputs will be NaN with fault (for AV) and False with fault (for DV).

TREND

Creates a means for gathering the inputs for a GCM TREND block.

Class Name: `tridiumx.smartware.gcm.TREND`

Remarks

The TREND block itself does not store the trended values. It is intended to link to one to four Analog or Binary logs, configured according to the parameters of the TREND log.

To simulate the functionality of the TREND log, the following TREND block outputs can be linked to the inputs of the logs:

<i>logAnalogValue1, logAnalogValue2, ... logDigialValue1, logDigialValue2, ..</i>	Linked to the <i>statusInput</i> of the corresponding log
<i>logExecuteTrigger</i>	Linked to <i>executeTrigger</i> of the log (with its <i>execution frequency</i> set to <i>onTrigger</i>)
<i>logDoClearTrigger</i>	Linked to <i>doClearTrigger</i> of the log
<i>logEnable</i>	Linked to <i>logEnable</i> of the log

The INP1-4 inputs have been replaced by INP1A, INP1D, etc. to handle the different data types.

Conversion Blocks

Conversion blocks are used to convert a single value from one compatible data type to another. There are a number of conversion blocks in the standard Niagara jar files, including *coreRuntime*, *lonWorks* and *lonExtras*.

The Smartware Jar File includes a set of additional conversion objects (contained in the conversion folder in the jar file) that were not found elsewhere.

The name of the block describes the type of the input and the converted output. Common abbreviations include:

Bs	Boolean Status
Bp	Boolean Priority
Fs	Float Status
Fp	Float Priority
Int	Integer
Ip	Integer Priority

The conversion blocks include:

BpToBs	IntPriorityToHvacModeEnum
BsToFp	IntStatusToHvacEmergencyEnum
FloatPriorityToHvacModeEnum	IntStatusToHvacModeEnum
FpToFp	IntToFp
FpToHvacModeEnum	IntToHvacEmergencyEnum
Fs2Fp	IntToHvacModeEnum
FsToBs	IntToLonLevDiscEnum
FsToFp	IpToHvacModeEnum
FsToHvacEmergEnum	IsToHvacEmergencyEnum
FsToInt	IsToHvacModeEnum
FsToOccEnum	LonLevDiscEnumToInt
HvacModeEnumToInt	SnvtStateDemux
HvacModeEnumToIntStatus	SnvtStateMux
HvacModeEnumToIs	

Enhanced Smartware Blocks

These blocks (contained in the sw folder in the jar file) are enhanced versions of standard Niagara objects and functionality.

SwAnalogSetpoint
SwBinarySetpoint Designer to replace a standard *AnalogOutput* or *BinaryOutput* object used to create a commandable, user-specifiable value. It provides for setting a value, setting an override value, and setting a timed override. When used in conjunction with the **SwGxText** object, it supplies information to show color coded backgrounds (for overrides) and a countdown timer (for timed overrides).

SwCmdString Creates a user-commandable object that persistently stores a string value.

SwFlexString
SwFlexString4
SwFlexString16 Enhances the *programLib FlexString* object to merge and format up to four input strings

SwGxText Enhances the standard **GxText** object to provide for multiple background colors and automatic formatting of values.

SwAnalogSetpoint, SwBinarySetpoint

A commandable, highly-formatted Setpoint object for storing an analog/binary value that can be changed and overridden (including timed overrides) by the user.

Class Name: *tridiumx.smartware.sw.SwAnalogSetpoint*
tridiumx.smartware.sw.SwBinarySetpoint

INPUTS	Data Type	Description
prioritizedInput [pln]	FloatPriority	The normal input value.
statusInput [sln]	FloatStatus	The normal input value (if <i>prioritizedInput</i> is not linked)
ForceAuto [auto]	BooleanStatus	Forces the output values to be equal to the input values (ignoring any overrides in effect). The <i>ForceAutoAction</i> parameter determines exactly when the transition occurs.

OUTPUTS	Data Type	Description
prioritizedOutput [pOut]	FloatPriority	The current value of the setpoint
statusOutput [sOut]	FloatStatus	The current value of the setpoint
state	IntegerStatus	0 if not overridden; 1 if timed override; 2 if overridden without the timer; 3 if overridden with the timer and the time should be displayed.
IsOverride [isOvr]	BooleanStatus	True when the override is in effect
IsNotOverride [notOvr]	BooleanStatus	True when the override is not in effect
IsTimedOverride [isTimd]	BooleanStatus	True when the timed override is in effect
OverrideMinutesRemaining [ovMin]	FloatStatus	The number of minutes remaining in a timed override.
OverrideTimeRemaining [ovTime]	Duration	The time remaining in a timed override
IsTimeVisible [isTmVis]	BooleanStatus	True when the time should be displayed in a timed override.

PARAMETERS	Data Type	Description
units	EngUnits	The engineering units for display
decimalFormat	DecimalFormat	The decimal format for display
priorityForWriting	ControlPriority	The priority level for the <i>prioritizedOutput</i> output
MinOverrideValue	Float	The minimum value that the user can specify as the override value.
MaxOverrideValue	Float	The maximum value that the user can specify as the override value.
MaxTimerMinutes	Float	The minimum value that the user can specify as the override time (in minutes)

ShowTimerCyclesTotal	Integer	The total cycles used to determine when the override time remaining should be displayed (e.g. in which <i>IsTimeVisible</i> is set to True).
ShowTimerCyclesOn	Integer	The number of cycles (out of <i>ShowTimerCyclesTotal</i>) in which the override time remaining is displayed (e.g. in which <i>IsTimeVisible</i> is set to True).
ForcelsOverrideToTrue	Boolean	If True, the <i>IsOverride</i> output is forced to True at all times.
commandTags	CommandTags	The text displayed on the command menu
OverrideState	Enum	The current state of the override
OverrideValue	Float	The current override value
OverrideExpiresAt	DateTime	The expiration time of a timed override
ForceAutoAction	Enum	Determines when the input value is forced as the output value (ignoring any override) based on the ForceAuto input.
AutoTimedOverrideMinutes	Float	The number of minutes to automatically set the timer for when the user sets the Override value.

Remarks

If *MinOverrideValue* and *MaxOverrideValue* are both set to 0, no range checking is done for the override value.

The user can set the Override Value and Override Timer values (in minutes) separately from the command menu.

- To automatically start the timer when the user sets only the value, set the *AutoTimedOverrideMinutes* parameter.

When the user has set a timed override:

- The *IsTimedOverride* output is set to True
- The remaining time is available on the *OverrideMinutesRemaining* and *OverrideTimeRemaining* outputs.
- The *IsTimeVisible* output will toggle between True and False. For every period of *ShowTimerCyclesTotal* cycles, the output will be true for the first *ShowTimeCyclesOn* cycles (for example, with values of 6 and 2, the output will be on for 2 cycles and then off for 4).

The *ForceAuto* input can be used to programmatically ignore the user-specified override and use the input value as the output value. The *ForceAutoAction* determines exactly how the block reacts to this input.

- If *ForceAutoAction* = WHEN_ON, the override value is ignored as long as the *ForceAuto* input remains on. If the user attempts to reset the override, it will still be ignored.
- If *ForceAutoAction* = RISING_EDGE, the override value is cleared when the *ForceAuto* input goes from False to True. The user can reset the override again, even if the *ForceAuto* remains on.
- If *ForceAutoAction* = FALLING_EDGE, the override value is cleared when the *ForceAuto* input goes from True to False. The user can reset the override again, even if the *ForceAuto* remains off.

To link the SwAnalogSetpoint object to a SwGxText for maximum effect:

SwAnalogSetpoint	SwGxText Inputs
<i>statusOutput</i>	<i>binding</i>
<i>State</i>	<i>boundBackground</i>
<i>IsTimeVisible</i>	<i>boundShowTimer</i>
<i>OverrideTimeRemaining</i>	<i>boundTimeRemaining</i>

SwCmdString

Creates a user-commandable object that persistently stores a string value.

Essentially a **CpString** object with the following differences:

- The *cmdLink* Flex Input property has been removed.
- The value entered by the user is stored in the new *StoredValue* property, and will therefore be stored in the station database and backups.

Class Name: *tridiumx.smartware.sw.SwCmdString*

OUTPUTS

value	String	The current user-specified string value
-------	--------	---

PARAMETERS

commandText	String	Defines how (and if) the command appears on the object's right-click command menu. You must enter a text value to make the object provide a user command
StoredValue	String	The current user-specified string value

Remarks

You must specify a value for *commandText* or the object will not be commandable.

SwFlexString, SwFlexString4, SwFlexString16

Enhances the *programLib* **FlexString** object to merge and format up to four input strings.

Class Name: *tridiumx.smartware.sw.SwFlexString*

INPUTS	Data Type	Description
link	String	The formatted output value as a flex input (e.g. it can be linked to and push values into the property of an object).

OUTPUTS	Data Type	Description
value	String	The formatted output value
strInput1 [str1]	String	The first input string (used to replace %1 in <i>FormatString</i>)
strInput2 [str2]	String	The second input string (used to replace %2 in <i>FormatString</i>)
strInput3 [str3]	String	The third input string (used to replace %3 in <i>FormatString</i>)
strInput4 [str4]	String	The fourth input string (used to replace %4 in <i>FormatString</i>)

PARAMETERS	Data Type	Description
FormatString	String	The format string used to generate the output

Remarks

The **SwFlexString** provides two important functions: Merging and Formatting strings and providing a Flex Input attribute to link to and push values in the another object's properties.

The output is created by starting with a copy of the *FormatString* parameter, and then merging in some of the input values. The special codes %1, %2, %3 and %4 are used as placeholders for the input strings.

For example, if the input strings are:

```
strInput1 = "Cafeteria"  
strInput2 = "VAV"  
strInput3 = "Setpoint"
```

Then the following *FormatString* values will yield the following output values

<u>FormatString</u>	<u>output</u>
Just Plain Text	Just Plain Text
%1	Cafeteria
The room is '%1'	The room is 'Cafeteria'
Room: %1 System: %2	Room: Cafeteria System: VAV
%1 %2 %3	Cafeteria VAV Setpoint

The calculated output value of the object is available as the *value* output and the *link* flex input.

Refer to the Niagara Standard Programmer's Reference (e.g., sections on **CpAnalog**, etc. controls) for some notes regarding the use and overuse of Flex Inputs.

The **SwFlexString4** and **SwFlexString16** objects provide additional link outputs for linking multiple objects with the same output value.

SwGxText

Enhances the standard **GxText** object to provide for multiple background colors and automatic formatting of values.

Class Name: `tridiumx.smartware.sw.SwGxText`

INPUTS	Data Type	Description
binding	Flex	The value to display in the SwGxText box
boundBackground	IntegerStatus	The index number of the background color to display
boundShowTimer	BooleanStatus	When True, the SwGxText box displays the value linked to the <i>boundTimeRemaining</i> input
boundTimeRemaining	Duration	The value to display when the <i>boundShowTimer</i> input is set to True.

PARAMETERS	Data Type	Description
mapping	Table	A table of colors to use as the background color. The value in the table corresponding to the <i>boundBackground</i> input is used.
defaultBoundBackgroundColor	Integer	The index in the color table to use when <i>boundBackground</i> is not linked.
applyAnalogUnits	Boolean	If true, the engineering units specified in <i>units</i> are used when displaying analog values.
units		If <i>applyAnalogUnits</i> is true, the units to use to display analog values.
applyAnalogDecimalFormat	Boolean	If true, the decimal format specified in <i>decimalFormat</i> are used when displaying analog values.
decimalFormat		If <i>applyAnalogDecimalFormat</i> is true, the decimal format to use to display analog values.
applyBinaryText	Boolean	If true, the active/inactive text values specified in <i>activeInactiveText</i> are used when displaying binary values.
activeInactiveText		If <i>applyBinaryText</i> is true, the active/inactive text values to use to display binary values.
applyMultistateText	Boolean	If true, the values specified in <i>stateText</i> are used when displaying multi-state values.
stateText		If <i>applyMultistateText</i> is true, the values to use to display multi-state values.

Remarks

The **SwGxText** provides two key additional functions beyond the regular GxText object:

- A table that maps integer values to colors can be specified, and the background color selected with the *boundBackgroundColor* input. The *defaultBoundBackgroundColor* is used if the input is not linked.
- A full set of formatting parameters (*units*, *decimalFormat*, *activeInactiveText* and *stateText*) are provided, and can be used to force the value to display with these units. There is a corresponding Boolean parameter (*applyAnalogUnits*, *applyAnalogDecimalFormat*, *applyBinaryText* and *applyMultistateText*) to enable each of these units.

Utility Blocks

These blocks (contained in the *objects* folder in the jar file) are utility blocks that provide other functionality not found in other jar files.

Most of the blocks are self-explanatory. Others are documented in more detail in the following pages.

NOTE: *There may be other blocks in the jar file that are not listed in this document. It is recommended that you not use those blocks, as their accuracy and completeness may be questionable.*

Annotation	Provides a way to display four lines of text on a WorkPlace Pro page. Also displays the current license status of the Smartware Jar File.
CurrentDateTime	Outputs the current date and time as a date value,
DegreesConversion	Converts a degrees value from Celsius to Fahrenheit, or from Fahrenheit to Celcius.
EdgePulseBs	Creates a pulse (of configurable duration) whenever the binary input value changes.
EdgeTrigger	Creates a trigger whenever the binary input value changes.
FlipFlop	Maintains a binary output value that switches every execution cycle. When configured to execute only on_trigger, acts as a triggerable flip-flop. Can also be used in conjunction with the <i>EdgeTrigger</i> to create an edge-based flip-flop.
FormatSeconds	Takes an integer number of seconds and formats it as time values.
MinMaxAvgSum16	Calculates the minimum, maximum, sum and average of up to 16 analog values.
MultiLogic32	Provides a logic function (AND, OR, XOR or NOT) for up to 32 inputs, with selectable inversion of each input.
SelectFs	Selects between two analog values based on a binary input value.
SlidingMinMax	A sliding min/max function (see detailed description page)
SlidingSetpoint	A sliding setpoint (see detailed description page)
StatusTester	Generates an analog and binary value with selectable status states (e.g., inFault, inAlarm, etc) for testing.
StatusValue	Generates an analog and binary value with selectable status states (e.g., inFault, inAlarm, etc) for testing.
SwapFs	Selectively swaps the values of two analog inputs based on a binary input value
UserInfo	Displays information about the logged in user

ValueAnalyzer	Compares an input to a setpoint (within a tolerance) and looks at a status value and returns a different value for each possible condition. Essentially a combination of a Compare block and several Select blocks (see detailed description page).
ValueComparer	Tries up to three different comparisons in order and returns a different value depending on which (if any) is true (see detailed description page).
ValueCounter	Looks at up to 16 inputs and counts the number of inputs that match a specified value. Up to 4 different value comparisons can be made at the same time (see detailed description page).
ValueTotalizer	Adds together five separate values from four different input sources. Essentially five 4-input adders combined together (see detailed description page).

SlidingMinMax

Holds two sets of two setpoint values (Min and Max) each that can be programmatically and periodically adjusted towards each other.

Class Name: `tridiumx.smartware.objects.SlidingMinMax`

INPUTS	Data Type	Default	Description
statusInput [sIn]	FloatStatus	70	The input value to compare.
statusSetpoint [sSp]	FloatStatus	70	The setpoint value to compare the input to.
statusReset [sRst]	BooleanStatus	FALSE	When TRUE, the outputs are reset to their initial values.
statusCalculate [sCalc]	BooleanStatus	TRUE	When FALSE, the inputs are ignored and the current output values are maintained. When the value transitions from FALSE to TRUE, the outputs are recalculated immediately.

OUTPUTS	Data Type	Default	Description
statusMinA [sMinA]	FloatStatus		The <i>MinA</i> output setpoint.
statusMaxA [sMaxA]	FloatStatus		The <i>MaxA</i> output setpoint.
statusMinB [sMinB]	FloatStatus		The <i>MinB</i> output setpoint.
statusMaxB [sMaxB]	FloatStatus		The <i>MaxB</i> output setpoint.
statusRecalcCounter [sReclc]	FloatStatus		The number of minutes until the next scheduled recalculation.

PARAMETERS	Data Type	Default	Description
InitialMinA	Float	40	The initial value of the <i>MinA</i> output.
InitialMaxA	Float	100	The initial value of the <i>MaxA</i> output.
InitialMinB	Float	40	The initial value of the <i>MinB</i> output.
InitialMaxB	Float	100	The initial value of the <i>MaxB</i> output.
IncrementA	Float	2	The amount to increase or decrease the <i>MinA</i> and <i>MaxA</i> outputs per calculation.
IncrementB	Float	2	The amount to increase or decrease the <i>MinB</i> and <i>MaxB</i> outputs per calculation.
MinimumSpreadA	Float	0	Determines how close the <i>MinA</i> and <i>MaxA</i> values can get to each other as they approach one another.
MinimumSpreadB	Float	0	Determines how close the <i>MinB</i> and <i>MaxB</i> values can get to each other as they approach one another.
CalculateIntervalMinutes	Float	10	The number of minutes between recalculations.
Tolerance	Float	0.5	The amount above or below the <i>statusSetpoint</i> value that the <i>statusInput</i> can be and still be considered equal.

Remarks

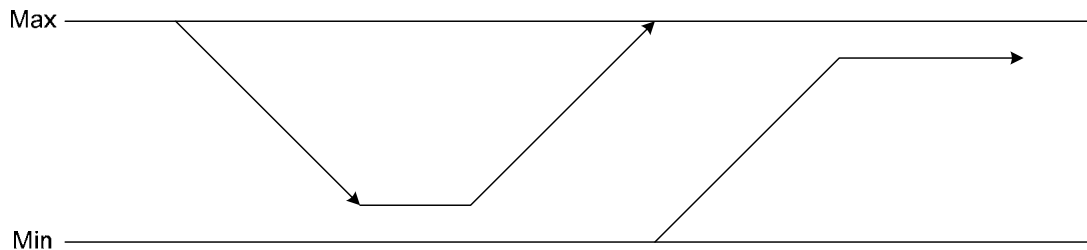
The **SlidingMinMax** maintains two sets of separate output setpoints (set A and B).

For each set, there is a Minimum output (*statusMinA* / *statusMinB*) and a Maximum output (*statusMaxA* / *statusMaxB*) that are maintained within the block.

When the block is initialized, the outputs are set to the specified initial value parameters (*InitialMinA*, *InitialMaxA*, *InitialMinB* and *InitialMaxB*) and the recalculation counter (*statusRecalcCounter*) begins to countdown from *CalculateIntervalMinutes*. When the counter reaches 0, the *statusInput* value is compared to the *statusSetpoint* input to determine how the outputs should change.

Condition	Change in Min Outputs	Change in Max Outputs
Input < Setpoint	Go down (if not at minimum)	Go down (if Min is not at minimum)
Input = Setpoint	Remain the same	Remain the same
Input > Setpoint	Go up (if Max not at maximum)	Go up (if not at maximum)

The intent is to move the Min and Max outputs towards each other, with only one value being off its initial value at any time. For example, If *statusInput* < *statusSetpoint*, the values of the Min and Max outputs will move as follows:



- For the first part, the Max output will be reduced by the corresponding Increment (*IncrementA* / *IncrementB*) value until it comes within the Spread amount (*MinimumSpreadA* / *MinimumSpreadB*) of the Min output. When it does reach this spread, the output values will remain there.
- If the *statusInput* value now exceeds the *statusSetpoint* value, the Max output value will begin to rise until it reaches its initial value. Once it does, the Min output value will start to increase until it comes within the spread of the Max value.

If *statusInput* is equal to *statusSetpoint*, both the Min and Max outputs will hold their values. If the inputs are within the *Tolerance* value of one another, they are considered equal.

If *statusReset* is TRUE, the outputs are reset to their initial values (and remain there until *statusReset* is FALSE).

If *statusCalculate* is FALSE, the outputs will retain their current values regardless of the value of *statusInput*. gd

When *statusCalculate* changes from FALSE to TRUE, the outputs will be adjusted immediately based on the value of *statusInput* and *statusSetpoint*. This allows the outputs to be moved manually faster than the normal interval by toggling the *statusCalculate* value from FALSE to TRUE and back again.

If *statusInput* or *statusSetpoint* is not a valid input, the output values will remain unchanged.

SlidingSetpoint

Holds a setpoint value that can be programmatically and periodically incremented or decremented based on an input value.

Class Name: `tridiumx.smartware.objects.SlidingSetpoint`

INPUTS	Data Type	Default	Description
statusInput [sIn]	FloatStatus	0	The input value that determines if the setpoint should increase, decrease or remain the same.
statusReset [sRst]	BooleanStatus	FALSE	When TRUE, the setpoint is reset to the <i>InitialSetpoint</i> value.
statusCalculate [sCalc]	BooleanStatus	TRUE	When FALSE, the input is ignored and the current setpoint value is maintained. When the value transitions from FALSE to TRUE, the setpoint is recalculated immediately.

OUTPUTS	Data Type	Default	Description
statusSetpoint [sSp]	FloatStatus		The current value of the setpoint
statusAtMax [sAtMax]	BooleanStatus		Indicates if the setpoint has reached <i>MaxSetpoint</i> .
statusAtMin [sAtMin]	BooleanStatus		Indicates if the setpoint has reached <i>MinSetpoint</i> .
statusRecalcCounter [sReclc]	FloatStatus		The number of minutes until the next scheduled recalculation.

PARAMETERS	Data Type	Default	Description
SetpointUpValue	Float	1	If <i>statusInput</i> equal this value, the setpoint will increase on the next recalculation.
SetpointDownValue	Float	-1	If <i>statusInput</i> equal this value, the setpoint will decrease on the next recalculation.
InitialSetpoint	Float	70	The initial value of the setpoint when the block is initialized or reset.
MinSetpoint	Float	40	The minimum value of the setpoint.
MaxSetpoint	Float	100	The maximum value of the setpoint.
SetpointIncrement	Float	2	The amount to increase or decrease the setpoint per calculation.
CalculateIntervalMinutes	Float	10	The number of minutes between recalculations.

Remarks

The **SlidingSetpoint** maintains an output setpoint value.

When the block is initialized, *statusSetpoint* is set to *InitialSetpoint* and the recalculation counter (*statusRecalcCounter*) begins to countdown from *CalculateIntervalMinutes*. When the counter reaches 0, the *statusInput* value is compared to *SetpointUpValue* and *SetpointDownValue*. If the *statusInput* matches one of these values, the setpoint is increased or decreased by the *SetpointIncrement* amount.

If *statusReset* is TRUE, the setpoint is reset to the value of *InitialSetpoint* (and remains there until *statusReset* is FALSE).

If *statusCalculate* is FALSE, the setpoint will retain its current value regardless of the value of *statusInput*.

When *statusCalculate* changes from FALSE to TRUE, the setpoint will be adjusted immediately based on the value of *statusInput*. This allows the setpoint to be moved manually faster than the normal interval by toggling the *statusCalculate* value from FALSE to TRUE and back again.

The setpoint will not exceed the range of *MinSetpoint* and *MaxSetpoint*. The outputs *statusAtMax* and *statusAtMin* indicate whether the setpoint has reached one of these limits.

If *statusInput* is not a valid input, the setpoint value will remain unchanged.

ValueAnalyzer

Compares an input to a setpoint (within a tolerance) and looks at a status value and returns a different value for each possible condition.

Essentially a combination of a Compare block and several Select blocks.

Class Name: `tridiumx.smartware.objects.ValueAnalyzer`

INPUTS	Data Type	Default	Description
statusInput [sIn]	FloatStatus	0	The input value to compare
statusSetpoint [sSp]	FloatStatus	0	The setpoint to compare the input to
statusStatus [sStat]	BooleanStatus	FALSE	The status flag value
statusEnable [sEnab]	BooleanStatus	TRUE	If FALSE, all other inputs are ignored and the <i>OutDisabled</i> value is used.

OUTPUTS

statusOutput [sOut]	FloatStatus		The output value (as described in the remarks)
---------------------	-------------	--	--

PARAMETERS

MinInput	Float	0	The minimum allowable value of <i>statusInput</i>
MaxInput	Float	100	The maximum allowable value of <i>statusInput</i>
MinSetpoint	Float	0	The minimum allowable value of <i>statusSetpoint</i>
MaxSetpoint	Float	100	The maximum allowable value of <i>statusSetpoint</i>
Tolerance	Float	0	The amount above or below the setpoint value that the input can be and still be considered a match
ToleranceSpPct	Float	1	The amount (expressed as a percent of the setpoint) above or below the setpoint value that the input can be and still be considered a match
OutDisabled	Float	0	The value for <i>statusOutput</i> if <i>statusEnable</i> is false or any of the inputs are invalid
OutFalseInLtSp	Float	1	The value for <i>statusOutput</i> if <i>statusStatus</i> is false and <i>statusInput</i> < <i>statusSetpoint</i>
OutFalseInEqSp	Float	2	The value for <i>statusOutput</i> if <i>statusStatus</i> is false and <i>statusInput</i> = <i>statusSetpoint</i> (within the tolerance)
OutFalseInGtSp	Float	3	The value for <i>statusOutput</i> if <i>statusStatus</i> is false and <i>statusInput</i> > <i>statusSetpoint</i>
OutTrueInLtSp	Float	1	The value for <i>statusOutput</i> if <i>statusStatus</i> is true and <i>statusInput</i> < <i>statusSetpoint</i>
OutTrueInEqSp	Float	2	The value for <i>statusOutput</i> if <i>statusStatus</i> is true and <i>statusInput</i> = <i>statusSetpoint</i> (within the tolerance)

OutTrueInGtSp	Float	3	The value for <i>statusOutput</i> if <i>statusStatus</i> is true and <i>statusInput</i> > <i>statusSetpoint</i>
---------------	-------	---	---

Remarks

The **ValueAnalyzer** returns one of seven values depending on the comparison of *statusInput* to *statusSetpoint* along with the value of *statusStatus*, as follows:

Comparison	<i>statusStatus</i>	<i>statusEnable</i>	<i>statusOutput</i>
Input < SP	FALSE	TRUE	<i>OutputFalseInLtSp</i>
Input = SP	FALSE	TRUE	<i>OutputFalseInEqSp</i>
Input > SP	FALSE	TRUE	<i>OutputFalseInGtSp</i>
Input < SP	TRUE	TRUE	<i>OutputTrueInLtSp</i>
Input = SP	TRUE	TRUE	<i>OutputTrueInEqSp</i>
Input > SP	TRUE	TRUE	<i>OutputTrueInGtSp</i>
		FALSE	<i>OutputDisabled</i>

In comparing the values, if the difference between the values is less than the Tolerance, they are considered equal. The Tolerance can be specified as an absolute value (*Tolerance*) or a percent of *statusSetpoint* (*ToleranceSpPct*). If both are specified, the higher resulting tolerance is used.

If *statusInput* is below *MinInput* or greater than *MaxInput*, the value of *MinInput* or *MaxInput* is used in the calculation.

If *statusSetpoint* is below *MinSetpoint* or greater than *MaxSetpoint*, the value of *MinSetpoint* or *MaxSetpoint* is used in the calculation.

If any of the inputs are invalid (fault, down, out of service or NaN), the output is *OutputDisabled*.

ValueComparer

Tries up to three different comparisons in order and returns a different value depending on which (if any) is true.

Class Name: *tridiumx.smartware.objects.ValueComparer*

INPUTS	Data Type	Default	Description
statusInputA [sInA]	FloatStatus	0	Input Value A
statusInputB [sInB]	FloatStatus	0	Input Value B
statusInputC [sInC]	FloatStatus	0	Input Value C
CompareA	FloatStatus	1	The value to compare <i>InputA</i> to
CompareB	FloatStatus	1	The value to compare <i>InputB</i> to
CompareC	FloatStatus	1	The value to compare <i>InputC</i> to

OUTPUTS

statusOutput [sOut]	FloatStatus		The output value
---------------------	-------------	--	------------------

PARAMETERS

OutValueA	Float	0	The value for <i>statusOutput</i> if <i>statusInputA</i> \geq <i>CompareA</i>
OutValueB	Float	0	The value for <i>statusOutput</i> if <i>statusInputB</i> \geq <i>CompareB</i> (and <i>statusInputA</i> $<$ <i>CompareA</i>)
OutValueC	Float	0	The value for <i>statusOutput</i> if <i>statusInputC</i> \geq <i>CompareC</i> (and <i>statusInputA</i> $<$ <i>CompareA</i> and <i>statusInputB</i> $<$ <i>CompareB</i>)
OutDefault	Float	2	The value for <i>statusOutput</i> if none of the comparisons are true.

Remarks

The **ValueComparer** makes three comparisons in order and returns a value based on which, if any, is true:

```
If statusInputA  $\geq$  CompareA, then statusOutput = OutValueA
Otherwise
  If statusInputB  $\geq$  CompareB, then statusOutput = OutValueB
  Otherwise
    If statusInputC  $\geq$  CompareC, then statusOutput = OutValueC
    Otherwise
      statusOutput = OutDefault
```

If an input or compare value is down, in fault, out of service or NaN, the corresponding comparison will be ignored.

ValueCounter

Looks at up to 16 inputs and counts the number of inputs that match a specified value. Up to 4 different values comparisons can be made at the same time.

Class Name: *tridiumx.smartware.objects.ValueCounter*

INPUTS	Data Type	Default	Description
statusInput1..16 [sIn1..16]	FloatStatus	NaN	The 16 input values to count

OUTPUTS	Data Type	Default	Description
statusCountA [sA]	FloatStatus		The count of input values equal to <i>ValueA</i>
statusCountB [sB]	FloatStatus		The count of input values equal to <i>ValueB</i>
statusCountC [sC]	FloatStatus		The count of input values equal to <i>ValueC</i>
statusCountD [sD]	FloatStatus		The count of input values equal to <i>ValueD</i>
statusCountOther [sX]	FloatStatus		The count of input values not equal to any of the 4 choices (A, B, C or D)

PARAMETERS	Data Type	Default	Description
ValueA	Float	0	The value to compare the inputs to in order to be counted in <i>statusCountA</i>
ValueB	Float	1	The value to compare the inputs to in order to be counted in <i>statusCountB</i>
ValueC	Float	2	The value to compare the inputs to in order to be counted in <i>statusCountC</i>
ValueD	Float	3	The value to compare the inputs to in order to be counted in <i>statusCountD</i>
Threshold	Float	0	The value above or below the compare value that the input can be and still be considered a match (see remarks)

Remarks

The **ValueCounter** looks at each of the 16 inputs. If the input is equal to *ValueA*, the *statusCountA* output is increased by 1.

The *statusCountA* output will indicate the number of inputs that are “equal” to the *ValueA* parameters. Likewise, *statusCountB*, *statusCountC* and *statusCountD* will indicate the number of inputs equal to *ValueB*, *ValueC* and *ValueD* respectively.

The *statusCountOther* output will indicate the number of valid inputs that don’t match any of the *Value* parameters.

Values are “equal” if the absolute value of the difference between them is less than the Threshold parameter (e.g., if Threshold = 0.1 and *ValueA* = 50, then input values between 49.9 and 50.1 would be considered matches to *ValueA*). Be aware that due to rounding, you should not rely on exact matches of decimal values (i.e., it is uncertain if the values of exactly 49.9 or 50.1 would match, though; 49.91 definitely would and 49.89 would definitely not)

If an input is down, in fault, out of service or NaN, it will be ignored completely. It will also not be included in the *statusCountOther* output. Therefore, the total of the five outputs will not always equal 16, but will always equal the number of valid, non-NaN inputs.

To count more than 16 values at one time, use multiple **ValueCounter** blocks and add the results together using one or more **ValueTotalizer** blocks.

ValueTotalizer

Adds together five separate values from four different input sources. Essentially five 4-input adders combined together.

Class Name: *tridiumx.smartware.objects.ValueTotalizer*

INPUTS	Data Type	Default	Description
statusInputA1..4 [sA1..sA4]	FloatStatus	NaN	The four A values to add together
statusInputB1..4 [sB1..sB4]	FloatStatus	NaN	The four B values to add together
statusInputC1..4 [sC1..sC4]	FloatStatus	NaN	The four C values to add together
statusInputD1..4 [sD1..sD4]	FloatStatus	NaN	The four D values to add together
statusInputX1..4 [sX1..sX4]	FloatStatus	NaN	The four X values to add together

OUTPUTS	Data Type	Default	Description
statusTotalA [sA]	FloatStatus		The sum of the four A inputs
statusTotalB [sB]	FloatStatus		The sum of the four B inputs
statusTotalC [sC]	FloatStatus		The sum of the four C inputs
statusTotalD [sD]	FloatStatus		The sum of the four D inputs
statusTotalX [sX]	FloatStatus		The sum of the four X inputs

Remarks

The **ValueTotalizer** is essentially five separate adders (A, B, C, D and X), each with up to four inputs (1 – 4).

If an input is down, in fault, out of service or NaN, it will be ignored completely and not included in the total.

To count more than 4 values at one time, use multiple **ValueTotalizer** blocks cascaded together.

The **ValueTotalizer** block can be used to aggregate the results from multiple **ValueCounter** blocks.