Working with GEMs in DeltaV Live

Introduction

DeltaV Live introduces the concept of GEMs (Graphical EleMents) and GEM classes. GEMs are the building blocks for DeltaV Live displays, similar to dynamos in DeltaV Operate in most ways. They are class-based objects used to create reusable items that contain dynamic data and animation behavior based on referenced system parameters. GEMs can be used on their own or to build more complex (or layered) GEMs, allowing for common structures to be created and reused throughout the library.

Each GEM has a set of customizable properties that give context to each instance, such as a module name, a function block, or a size property to define positioning options for the instance. In combination with defined standards and functions from the library, GEM libraries can be developed to satisfy even the most demanding HMI specifications. Common items like icons or data groupings, alarm information, abnormal status, etc. can generally be broken down into common component GEMs and then assembled together to create purpose-built GEMs.

This document contains HMI philosophy considerations and best practices for making use of GEM classes in DeltaV Live. Keep in mind that there are multiple ways to create the same visual result in an operator graphic, and there is a balance between ease of use and efficiency. This white paper describes strategies for structuring GEMs for simplicity and ease of use, and considerations when creating built-for-purpose GEMs.

This document is targeted at graphics engineers who already have familiarity with DeltaV Live, have a need to design or create new GEM classes, and need to know how best to leverage the class-based GEM capabilities within their graphics.

This document will focus on recommendations and best practices and will refrain from duplicating the information from Books Online, except where it is expeditious to do so.

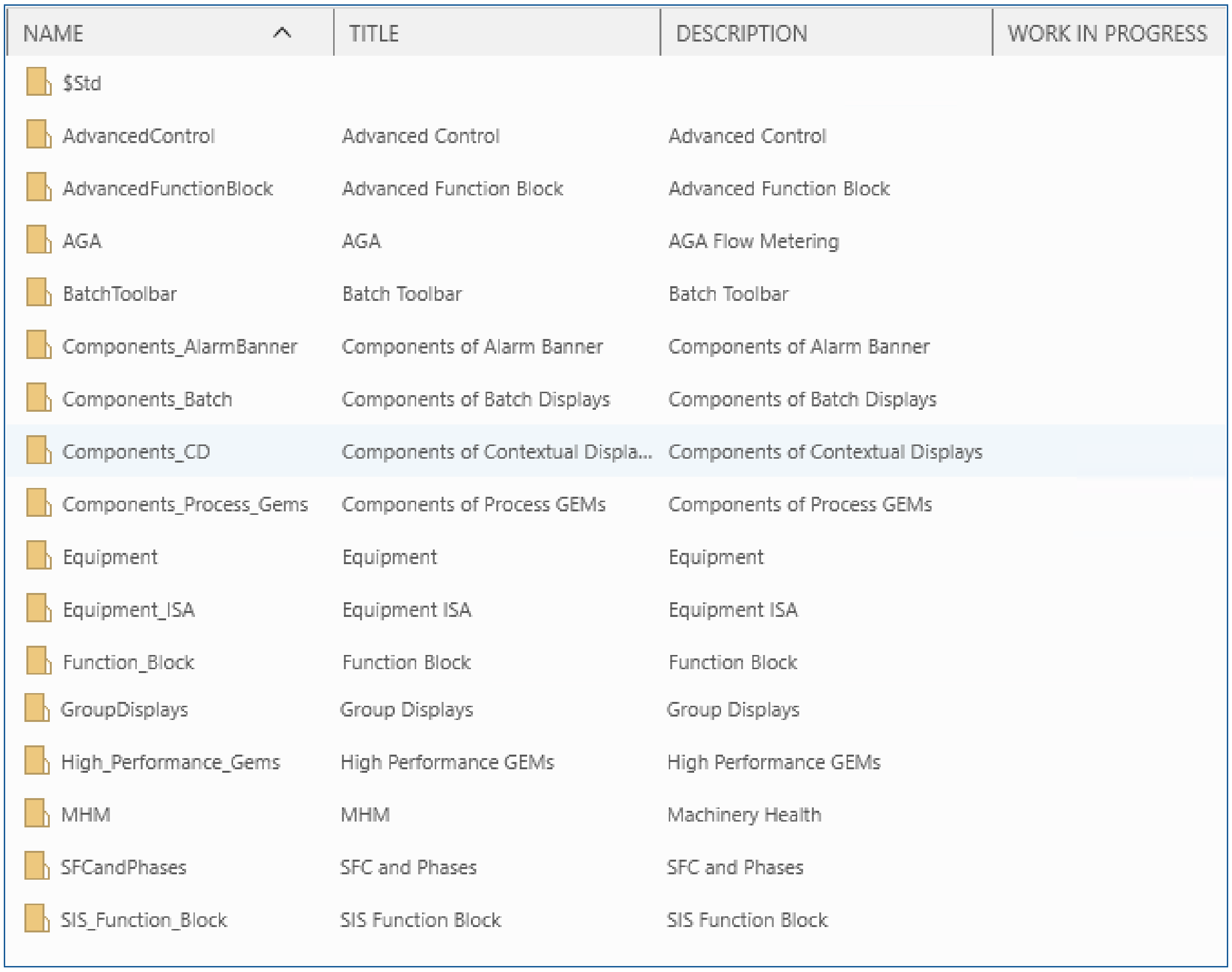
For more information about DeltaV Live, as well as a high-level overview of GEM classes, we recommend that you refer to:

- Welcome to DeltaV Live (Whitepaper)
- DeltaV Books Online

Introduction to GEM Classes and Folders

The default Emerson Library created during the DeltaV Live installation contains a set of GEM classes, templates, standards, functions, themes, and languages. DeltaV Books Online provides more detail on the structure of the database.

The out-of-the-box (OOB) GEM classes are contained in a folder called Emerson, which contains seventeen sub-folders. Users can also create their own project-specific folders. By grouping GEM classes in a project folder, you can easily export these items in a single action. You can add additional folders as needed, up to 4 levels deep.



NAME	TITLE	DESCRIPTION	WORK IN PROGRESS
$Std			
AdvancedControl	Advanced Control	Advanced Control	
AdvancedFunctionBlock	Advanced Function Block	Advanced Function Block	
AGA	AGA	AGA Flow Metering	
BatchToolbar	Batch Toolbar	Batch Toolbar	
Components_AlarmBanner	Components of Alarm Banner	Components of Alarm Banner	
Components_Batch	Components of Batch Displays	Components of Batch Displays	
Components_CD	Components of Contextual Displa...	Components of Contextual Displays	
Components_Process_Gems	Components of Process GEMs	Components of Process GEMs	
Equipment	Equipment	Equipment	
Equipment_ISA	Equipment ISA	Equipment ISA	
Function_Block	Function Block	Function Block	
GroupDisplays	Group Displays	Group Displays	
High_Performance_Gems	High Performance GEMs	High Performance GEMs	
MHM	MHM	Machinery Health	
SFCandPhases	SFC and Phases	SFC and Phases	
SIS_Function_Block	SIS Function Block	SIS Function Block	

OOB GEM Class Folder Structure.

GEM classes in folders called “Components” are building blocks intended to be used by GEM and contextual display designers or engineers. Component GEMs can be used together by display engineers to build more functional GEM classes such as process GEMs (including “High_Performance_GEMs”), contextual displays, alarm banners, and batch display items Process GEM classes will ultimately be used by display engineers to build operator displays.

Here are some tips to make the display engineer’s job easier:

- Process GEMs should be designed with only the number of exposed properties that they need, depending on the intended use cases for the GEMs. For instance, alarm names may not need to be exposed on the configuration form of the Process GEMs. These could be defined by the GEM designer to match the control modules’ alarm name standards.
- Use Selection property to allow one property to drive multiple sub properties based on one decision. This reduces the number of choices the display engineer has to make.
- Make use of Tool Tips to guide the display engineer on how to set up a GEM instance.
- Consider adding a description of applicable function blocks and main feature set to GEMs that you create. This can help display designers understand which GEMs to use when they view GEM folders in the library.
- The display engineer should be able to select from a small number of Process GEMs that match Plant P&ID items or equipment that align with project needs and can accommodate various options.

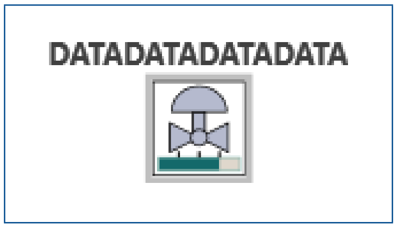
It is strongly recommended to leave the OOB GEM classes as found and consider creating a new library for adding project-specific classes. Unused or unmodified GEM classes should be left in their original location. The new library and project folder contains GEM classes that will be used for a particular project and these may be unmodified, modified, or new classes.

If a user wants to avoid the possibility of Emerson overriding any custom changes in a new library when the OOB GEMs (including component GEMs) are updated or repaired, GEMs in the new library should be copied from the OOB library to a new library first. When updates or fixes are provided for the OOB library by Emerson, the user will be given a choice to either accept or reject any updates to their GEM classes.

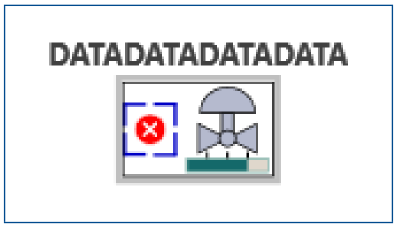
When to Make a New GEM Class

One of the core principles underlying GEM classes in DeltaV Live is that they represent a common set of functionality and are intended to be reused multiple times on displays or other GEM classes, e.g. the following representation of a control valve:

Wherever a GEM class is actually used on a display or other GEM class, that usage is called a GEM instance or just a GEM.



GEM classes can be built with a variety of configuration options (see GEM Properties) that allow them to appear or behave differently for each GEM instance. For example, the GEM class for our control valve may have an option to hide or show alarming and status icons. Above, the icons are hidden, but when the option is turned on, the valve may look like this:



The reusability of GEM classes as well as the ability to create configuration options are beneficial because they allow the configuration of the appearance and behavior of similar GEM instances to be managed in one place in the library, while still allowing for a wide variety of different use cases for each of the instances.

Subsequently, the question naturally arises during GEM class design: When should you create a new GEM class, and when should you use a configuration option to add functionality to an existing GEM class instead?

Here, it may also be worth pointing out that the two options are not necessarily mutually exclusive – you may decide to create a new GEM class where its instances are used as components inside the existing GEM class and then turned on or off with a configuration option.

That said, there are at least 3 factors to consider when deciding whether to add functionality to an existing GEM class or to create a new class:

- The user experience of the operator who will load displays containing the GEM instances.
- The user experience of the engineer who will build displays or other GEM classes by making use of the GEM instances.
- The effort associated with configuring and maintaining the GEM class.

Ultimately, it is important to consider the user experience of the operator who is going to be interacting with the displays. After all, each piece of functionality in the GEM class is there to help them monitor and control the processes within their span of responsibility.

Present Online Option

If we continue to use the example of our control valve above, when the status icon is not ever intended to be shown to the operator for a particular GEM instance, we want to make sure that the display is not still subscribing to the status parameters and thus unnecessarily slowing down the display's performance. You can accomplish this by creating two GEM classes, one with status icons and one without, or by separating the status icons into their own GEM class that engineers can add as needed while building displays, but neither of those options is ideal because they would add an additional burden to the configuration engineer whose job is to build the displays. They would need to know about a greater number of GEMs, and they would be made responsible for remembering to use the right GEMs in the right situations.

The ideal case may be for the GEM class to be able to include the functionality as part of the class definition without negatively impacting runtime performance of the display when the functionality is not needed on an instance. To specifically address this, DeltaV Live provides a property called 'Present Online' as an option on groups and GEM instances. When this property is set to false, the group or GEM is never even loaded into DeltaV Live runtime. The most typical use of 'Present Online' is to tie it to a configuration option on the GEM class (this is also a handy capability to remember if you ever want to show something to configuration engineers that doesn't need to be shown to operators). Using our control valve example, this means that the configuration engineer can have a single GEM class that they need to know about, and then they can turn the status icons on or off as needed for each instance. In each instance where the icons are turned off, there will be no negative performance impact associated with unnecessarily subscribing to the status parameters on the display.

Sometimes however, adding more functionality to a GEM class is not necessarily the right call. Taking the thought to the extreme, we may imagine a single universal GEM class that contains extensive options to switch between different control strategy types and presentation choices. Generally speaking, your goal should not be to build a multi-tool as much as it should be to build the right tool for the right purpose. To that end, you need to give some consideration to how you want configuration engineers to plan and build displays.

GEM design should start with an understanding of the control modules they will be used with.

You may decide it makes sense to build a few different GEMs for analog indicators, or you may decide that it makes sense to build one single analog indicator GEM with a configuration option to switch between different presentations. You may decide to build a single valve GEM class that can be used to represent analog control valves, on/off control valves, 3-way valves, hand valves, and others, or you may decide to build these as entirely separate GEM classes. Digging into the components of GEM classes, you may decide to build a single common GEM frame class with all the status icons, alarm indications, loop information, and user interactions, or you may decide that it makes sense to build these as separate GEM classes that can be put together each time you are building a new top-level GEM class that will be used directly on displays.

There's no single right answer, and the important thing is to give some thought to the factors outlined above. First, build for the operator, and then consider the display engineer user experience. How many GEMs does the display designer have to choose from? By building fewer GEMs with Present Online options, the user can be given clearer choices on which GEM to select, and the GEM Property Titles and tool tips can help guide the configuration options on the GEM. The user will need to decide on the appropriate balance between the effort required to build the GEM classes and the total effort required to build displays or other GEM classes.

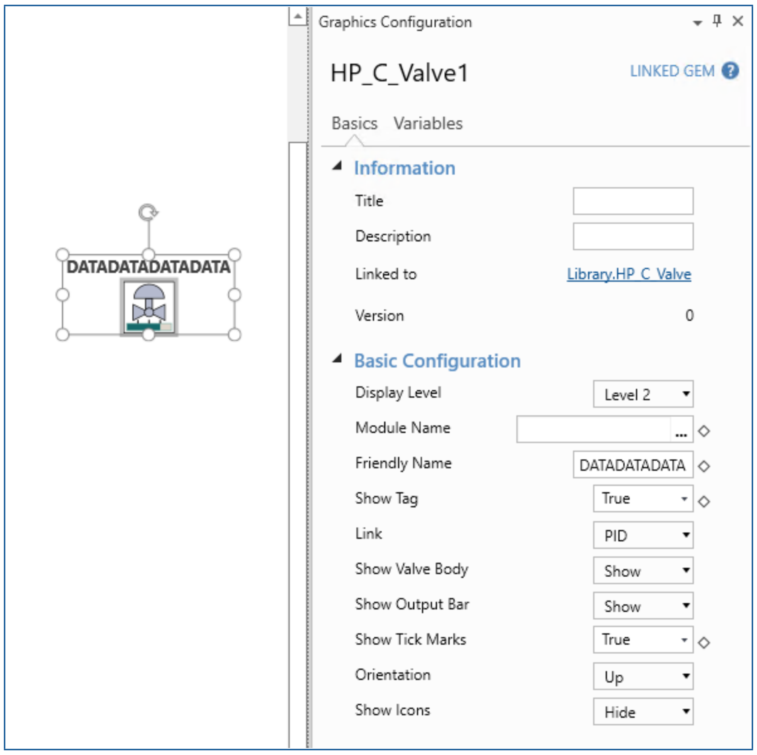
Finally, try to plan ahead. You will not be be able to anticipate every need, so try to leave the door open for future enhancements. There are GEM classes that you may need to create down the line, and remember that future engineers will inherit the work you put into your library.

For each piece of functionality that you are designing into a GEM class, think about the simplest way that you can build it while providing the correct experience that you want to provide for the operator and for the engineers who will be making use of the library. This may end up meaning that you decide not to create a complex GEM class wherever you find that simpler components will suffice to provide an acceptable or equivalent experience. Or it may mean that you decide to rely upon the "Present Online" functionality for nearly all changes to GEM appearance, and not to use configuration options to drive adjustments to the size or positioning of GEM class components if you can help it, since those can be the source of quite a bit of complexity.

Designing Configuration Options

GEM Properties

When a GEM instance is created in a display in Graphics Studio, the properties of the GEM are visible in the Graphics Configuration pane, which is the DeltaV Live version of the Dynamo Configuration dialog in DeltaV Operate. In addition to the default properties of display objects (e.g. name, geometry, visibility), there are typically GEM-specific properties that need to be defined, such as a module name or orientation.



HP_C_Valve1

Information

Title	
Description	
Linked to	Library.HP_C_Valve
Version	0

Basic Configuration

Display Level	Level 2
Module Name	
Friendly Name	DATADATADATA
Show Tag	True
Link	PID
Show Valve Body	Show
Show Output Bar	Show
Show Tick Marks	True
Orientation	Up
Show Icons	Hide

GEM Instance Properties Example.

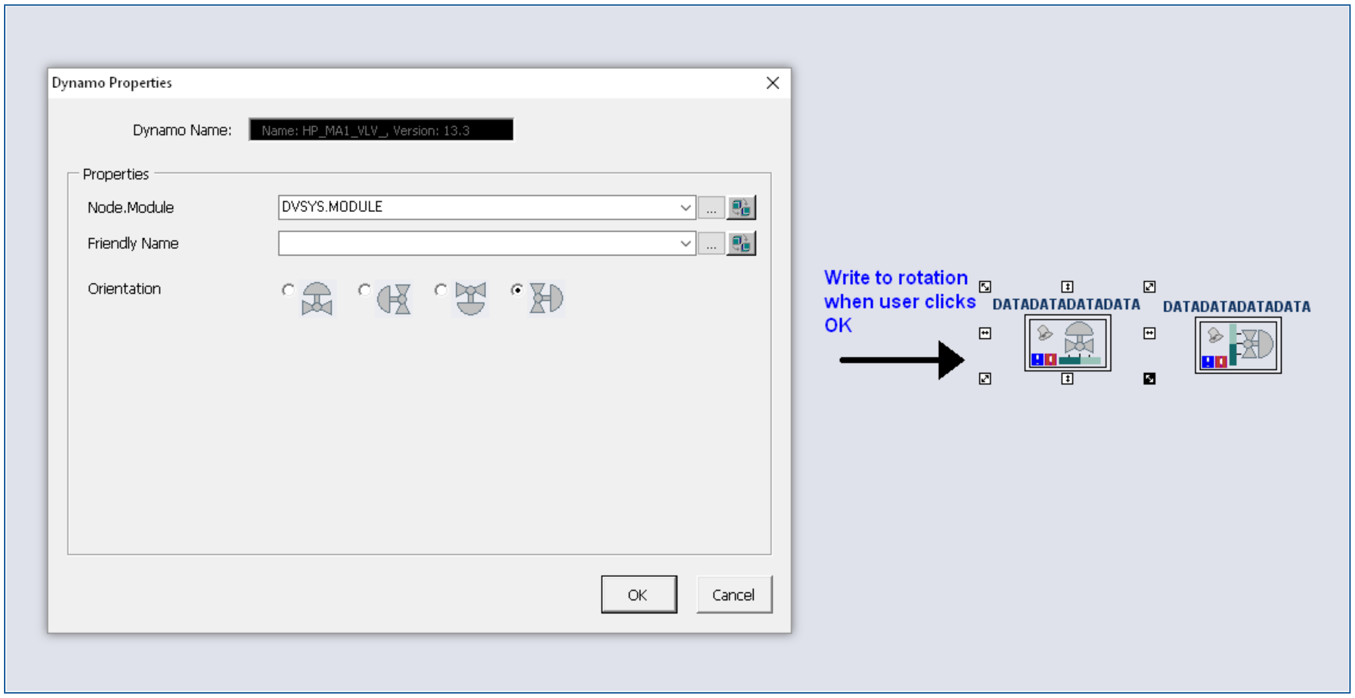
The control valve GEM and some of its properties are shown above. The “Information” section is common to all GEMs and includes the “Linked to” property, which identifies the source GEM class, in this instance ACME.HP_C_VALVE. Selecting this link takes you directly to the GEM class’s properties, where modifications can be made to the class. It is also possible to unlink the GEM from the class and modify the instance to create a unique implementation that is independent from the class definition.

More information about specific GEM properties may be found in Books Online; this document is not intended to be a duplicate of that resource.

DeltaV Operate Dynamo Forms versus DeltaV Live GEM Properties

It is worth understanding that there is a significant difference between the way that DeltaV Operate handles dynamo configuration and the way that DeltaV Live handles GEM configuration. The way that these properties work in DeltaV Live also has an impact on GEM design.

In DeltaV Operate, dynamo forms are essentially simple computer programs. When the user presses a button on the form, a script is executed and makes changes to the properties of shapes contained within the dynamo. This means that dynamo forms are highly flexible, but they can also be very difficult to modify, and graphics engineers who seek to make one-off changes to shape properties directly can have unintended effects when the dynamo form is used again later.


Dynamo Properties

Dynamo Name: Name: HP_MA1_VLV_, Version: 13.3

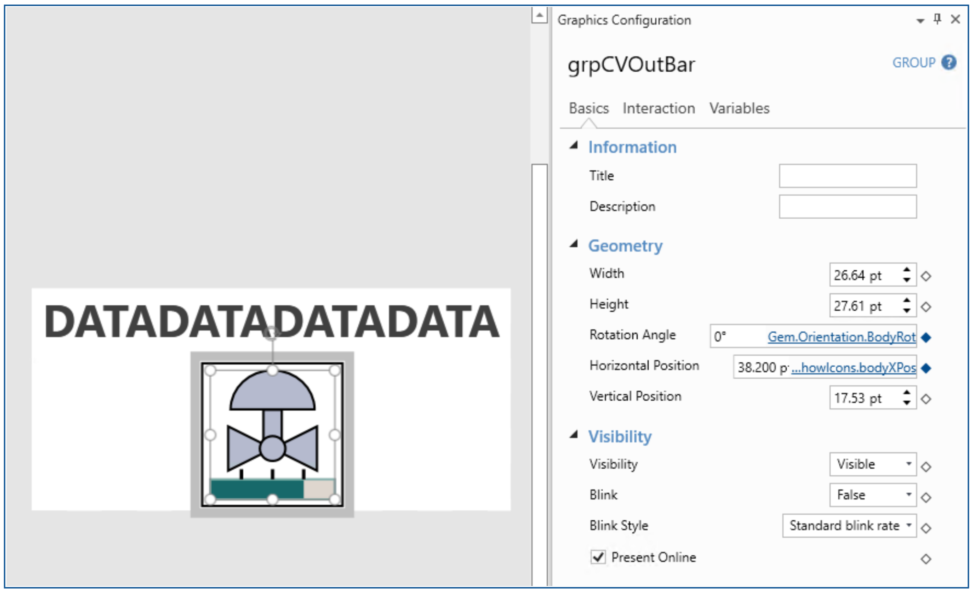
Properties

Node.Module: DVSYS.MODULE

Friendly Name:

Orientation

In DeltaV Live, this works differently. Instead of the form executing a script that writes to shape properties, the shape properties within the GEM can be configured to reference the GEM properties. Above, we showed the control valve GEM has a property called 'Orientation'. Below you can see that within the GEM class, the group containing the control valve GEM has a rotation property that is set up to reference the GEM's Orientation property. When the Orientation selection changes on a GEM instance, the rotation of the group inside also changes as a result of the reference.


Graphics Configuration

grpCVOutBar — GROUP

Basics | Interaction | Variables

Information
- Title:
- Description:

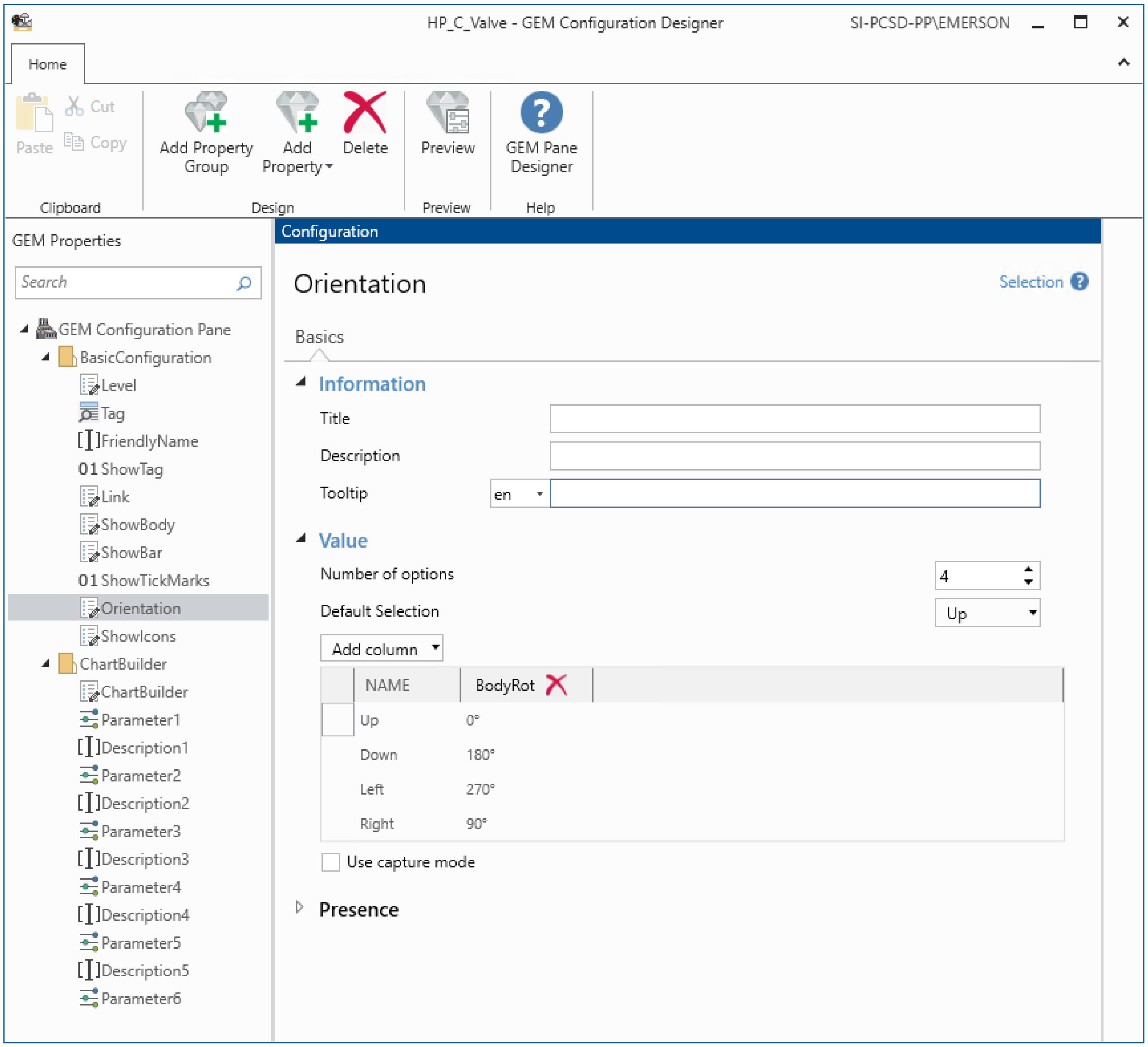
Geometry
- Width: 26.64 pt
- Height: 27.61 pt
- Rotation Angle: 0° Gem.Orientation.BodyRot
- Horizontal Position: 38.200 p ...howIcons.bodyXPos
- Vertical Position: 17.53 pt

Visibility
- Visibility: Visible
- Blink: False
- Blink Style: Standard blink rate
- ✓ Present Online

Note that you should not just expose every component GEM property; only expose the properties that you need to. Consider that some properties can be hardcoded with standards, so that these options are managed globally from the standard rather than configuring them in each GEM class. It is worth remembering that modifying a GEM class requires all affected displays to be published, while updating a standard does not.

Selection Properties in DeltaV Live

GEM properties are configured using the GEM Configuration Designer, which is accessible via the ribbon in Graphics Studio when editing a GEM class.



Number of options: 4
Default Selection: Up
Add column

NAME	BodyRot
Up	0°
Down	180°
Left	270°
Right	90°

Use capture mode

The Orientation property discussed above is an example of a Selection property. These are often used to simplify the user experience for the engineer who will be building the graphics. In the case of the control valve, we wanted to provide “Up”, “Down”, “Left”, and “Right” options to indicate the position of the actuator, and we didn’t want to make the graphics engineers have to know which specific rotation angles to use. Inside the Orientation property is a subproperty called Body Rotation (of type Degree), which contained the values 0°, 180°, 270°, and 90° corresponding to each of the possible selections.
In the pictures above, you can see that the group’s rotation property is actually set up to reference the ‘Body Rotation’ subproperty of ‘Orientation’.

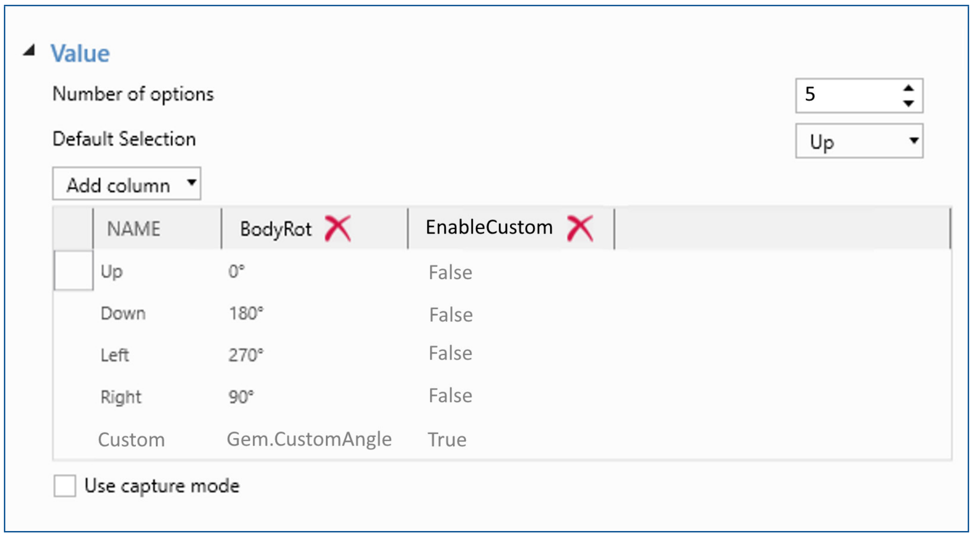
Each Selection property can have many different subproperties, meaning that when the graphics engineer makes a change in the selection, several shape or group properties could be impacted. Subproperties can also be True/False (aka Boolean) and be used in conjunction with the "Present Online" functionality discussed above. Using these, it is even possible to create GEM classes with multiple choices for visual representation that are completely different from one another, if desired.

Likewise, selection subproperties may reference other GEM properties or subproperties, including within the same Selection property. Below is an example of this.

Consider adding a custom orientation angle to this Selection property. This would allow a display engineer to rotate the valve to any angle. How could this be done?

1. Add a new property named "CustomAngle" of type Degree Angle to the GEM and position it below the Orientation property.
2. Add a 5th option to the Selection and call it "Custom". Define the BodyRot subproperty to point to the new "CustomAngle" property.
3. Add a second SubProperty of type Boolean called EnableCustom and set its value to False for the first 4 options and True for "Custom".
4. Set the Presence option for the new "CustomAngle" property to point to the new Boolean subproperty EnableCustom.

Now, when the user goes to set the Orientation angle, he can select from UP, DOWN, LEFT, RIGHT and Custom. When Custom is selected, the CustomAngle property will appear, allowing him to enter any angle. If Custom is not selected, the CustomAngle property does not appear.


Value

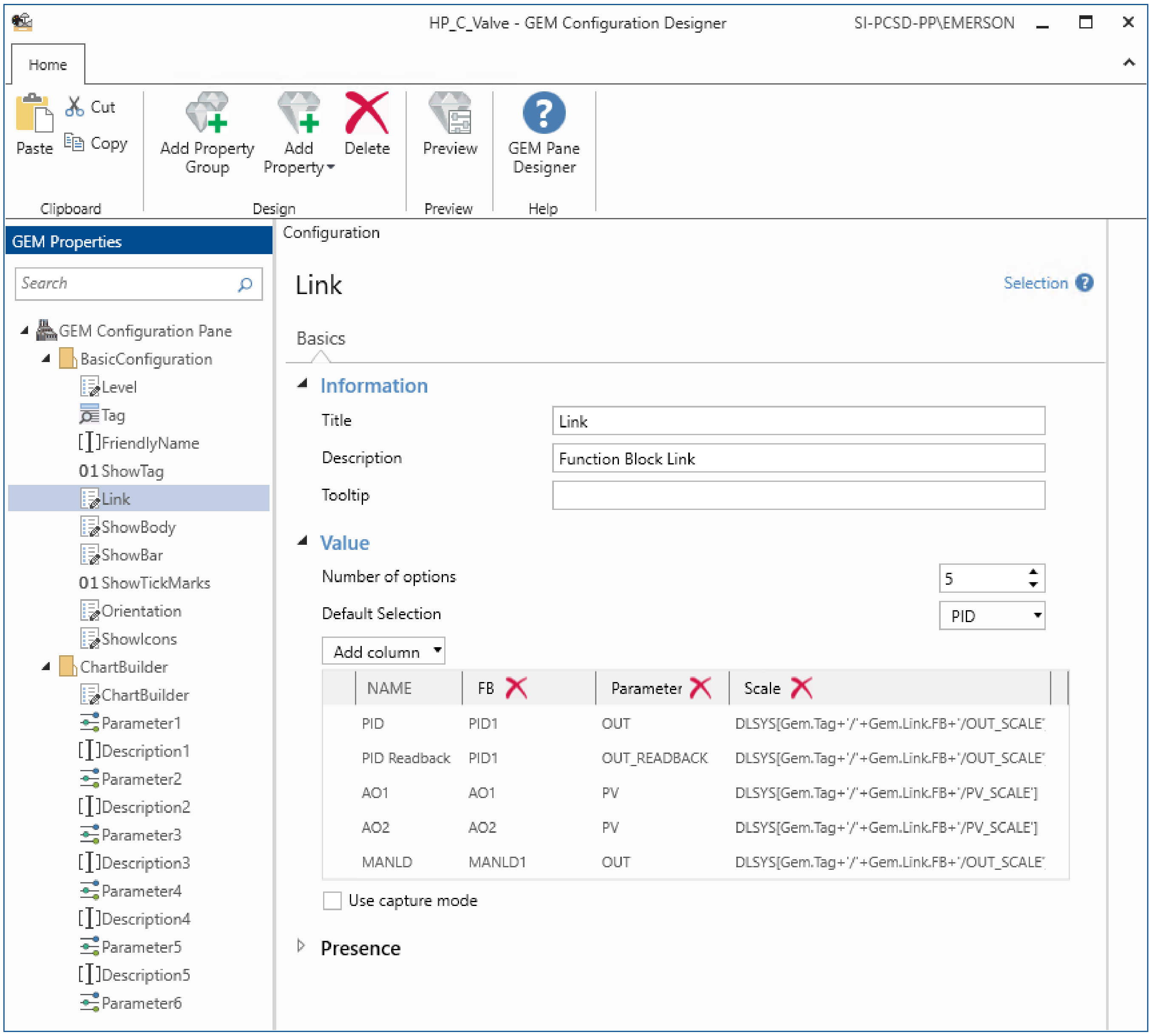
Number of options: 5

Default Selection: Up

NAME	BodyRot	EnableCustom
Up	0°	False
Down	180°	False
Left	270°	False
Right	90°	False
Custom	Gem.CustomAngle	True

☐ Use capture mode

The Link selection shown below allows the same GEM class to be useful for multiple different function blocks. Based on the selection chosen by the graphics engineer, the GEM changes a number of values that impact the animations used within the GEM instance. One of these is the Scale subproperty, which contains the DeltaV parameter reference to use for the output scale of the control valve. As you can see above, the parameter paths for the Scale subproperty contain references to the FB subproperty of the same Link property.



Title	Link
Description	Function Block Link
Tooltip	
Number of options	5
Default Selection	PID

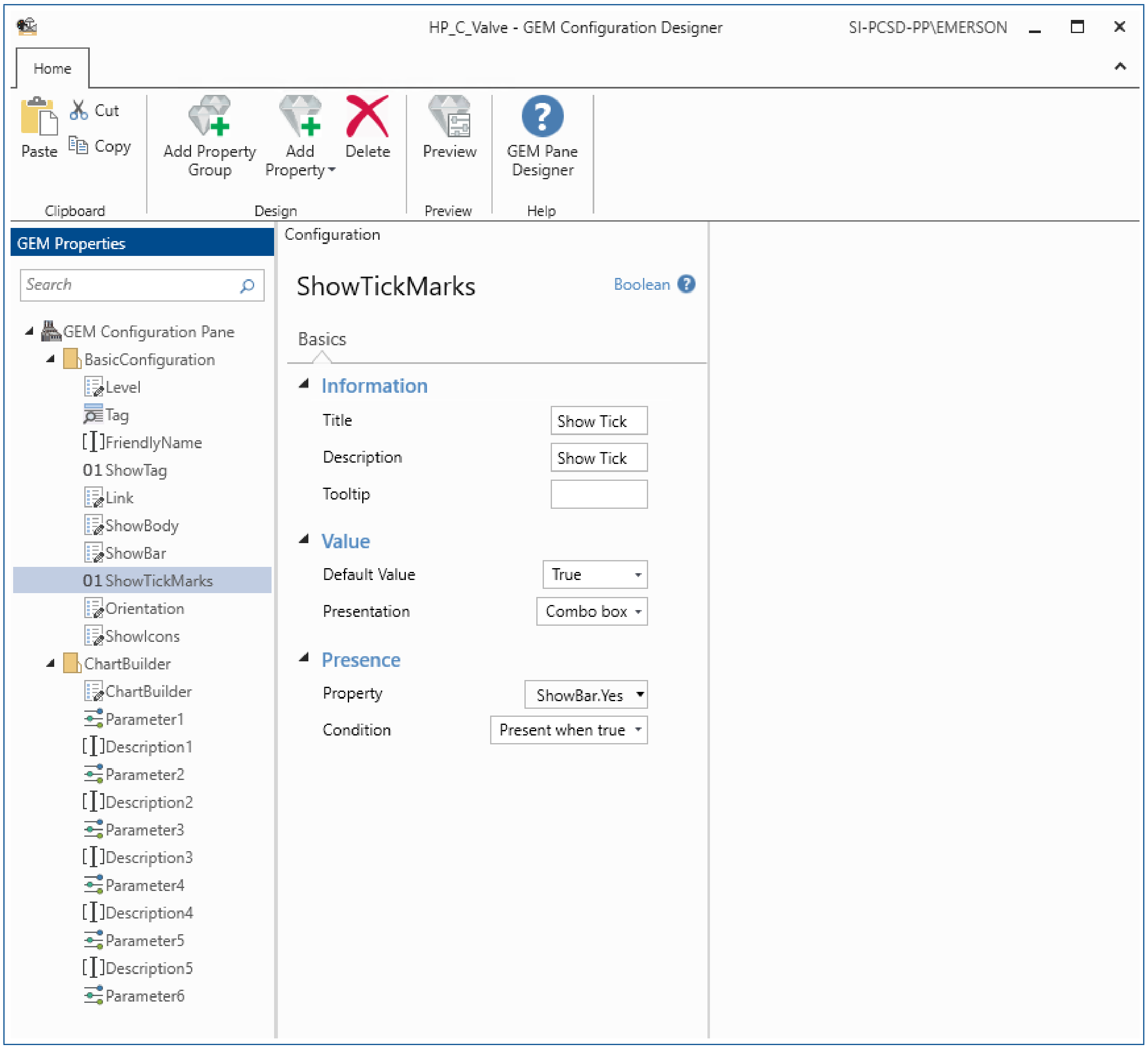
NAME	FB	Parameter	Scale
PID	PID1	OUT	DLSYS[Gem.Tag+'/'+Gem.Link.FB+'/OUT_SCALE'
PID Readback	PID1	OUT_READBACK	DLSYS[Gem.Tag+'/'+Gem.Link.FB+'/OUT_SCALE'
AO1	AO1	PV	DLSYS[Gem.Tag+'/'+Gem.Link.FB+'/PV_SCALE']
AO2	AO2	PV	DLSYS[Gem.Tag+'/'+Gem.Link.FB+'/PV_SCALE']
MANLD	MANLD1	OUT	DLSYS[Gem.Tag+'/'+Gem.Link.FB+'/OUT_SCALE'

☐ Use capture mode

These examples illustrate that Selection properties can be used to create fairly basic GEM configuration options, but also that Selection properties can help facilitate the creation of very flexible and complex configuration options as well.

Property Presence

It may be necessary at times to show or hide certain properties based on other property values. This can be accomplished using the Presence settings for each GEM property. In the example below, the ShowTickMarks property allows graphics engineers to choose whether or not to show tick marks along the output bar graph. This option is really only relevant when the output bar is being shown, so the ShowTickMarks property is configured to only be present on the configuration form when the 'Yes' subproperty for the 'ShowBar' selection property is True, which occurs when ShowBar is set to 'Show'.



ShowTickMarks — Boolean

Information

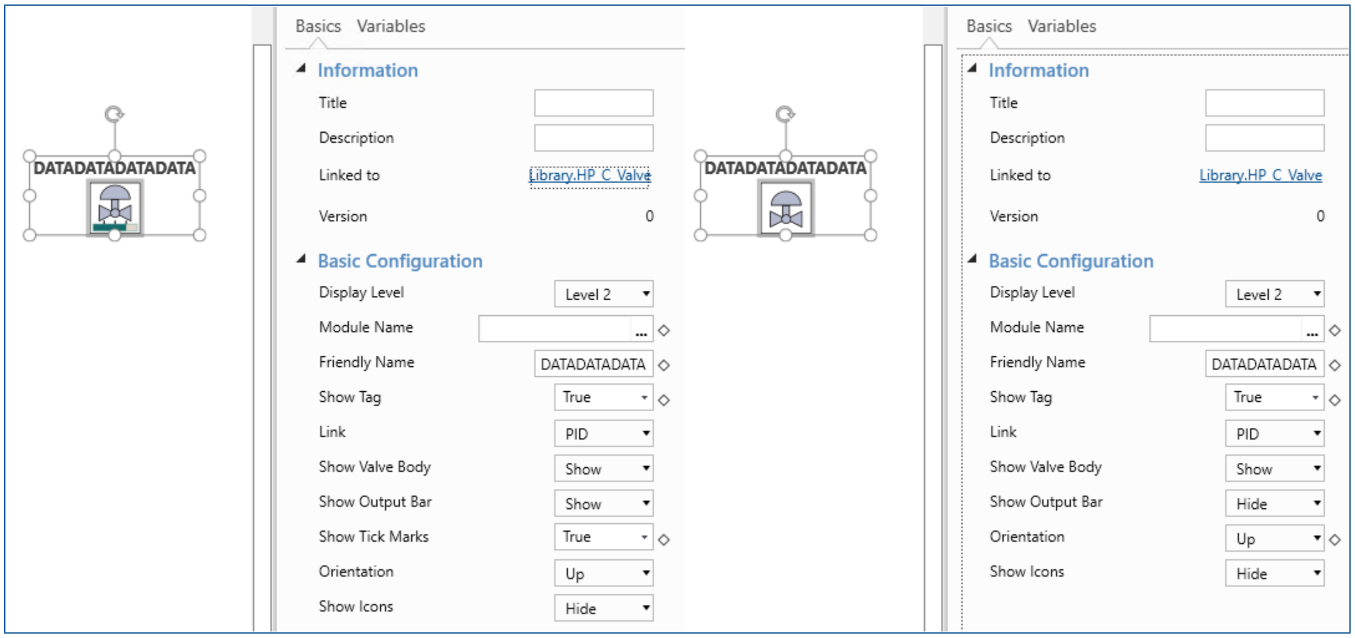
Title	Show Tick
Description	Show Tick
Tooltip	

Value

Default Value	True
Presentation	Combo box

Presence

Property	ShowBar.Yes
Condition	Present when true



	Left instance	Right instance
Information		
Title		
Description		
Linked to	Library.HP_C_Valve	Library.HP_C_Valve
Version	0	0
Basic Configuration		
Display Level	Level 2	Level 2
Module Name		
Friendly Name	DATADATADATA	DATADATADATA
Show Tag	True	True
Link	PID	PID
Show Valve Body	Show	Show
Show Output Bar	Show	Hide
Show Tick Marks	True	
Orientation	Up	Up
Show Icons	Hide	Hide

As in the example of 'Show Tick Marks', the Presence setting is a very useful way to simplify the GEM instance configuration experience for graphics engineers when a piece of GEM functionality which normally has its own configuration options is turned off via the Present Online capability.

It is also possible to delete GEM properties, but note that it is recommended to avoid deleting a GEM property if there are linked instances of the GEM being used in displays. When a property is deleted, all instances will flag the now-unbound property configuration, highlighted in a red box. You must explicitly delete the unbound property configuration to completely remove it.

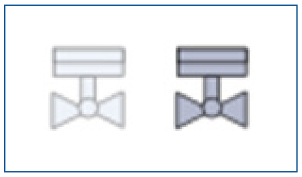
Flexible vs. Built for Purpose

Before building any GEM class, thought should be given to the question of whether the GEM class should be more flexible or more built for purpose, and how the flexibility should be implemented for the configuration experience that is intended for the configuration experience that is intended.

This topic is important because if you need to change a GEM class to be more or less flexible at a later time, it can often require making changes to the GEM configuration form. It can often require changes to the GEM configuration form. If you have already started using the GEM class elsewhere in configuration, making changes to the configuration form could require significant effort to revisit all the places that the GEM class was used in order to reconfigure the instances.

There are several ways that a GEM class may be made more flexible or more built for purpose. Below are a couple of examples.

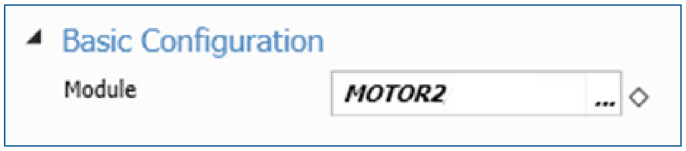
1. You want to animate the body color of a discrete control valve, as follows:



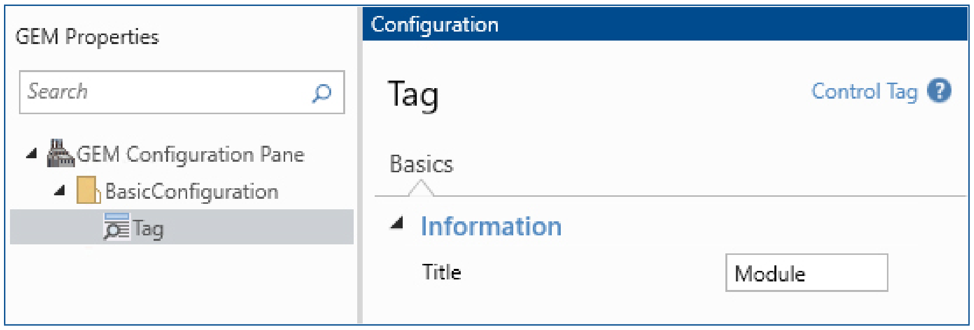
What are your options?

a. The most built-for-purpose approach might be to put an option on the configuration form for the module tag.

GEM instance configuration:

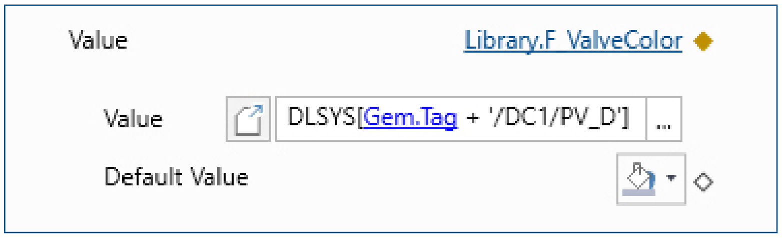

Basic Configuration
Module: MOTOR2

GEM class form configuration:


GEM Properties
GEM Configuration Pane
BasicConfiguration
Tag

Configuration
Tag
Control Tag
Basics
Information
Title: Module

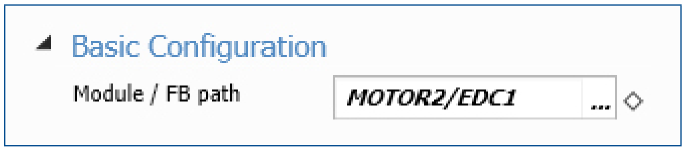
Shape color configuration:


Value: Library.F_ValveColor
Value: DLSYS[Gem.Tag + '/DC1/PV_D']
Default Value

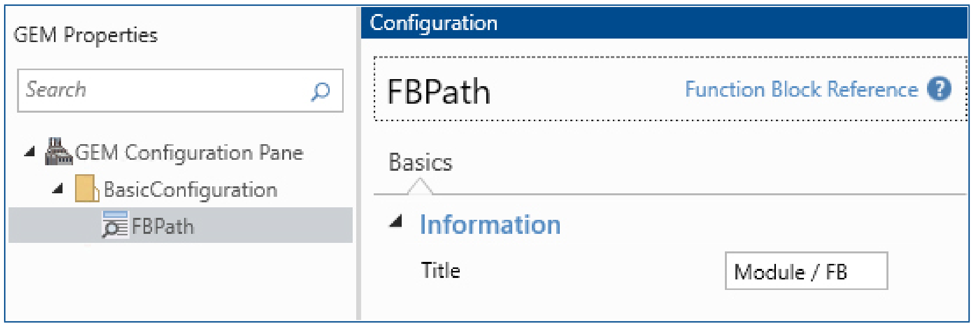
A benefit to this approach is that the instance is very simple to configure, but it is limited in that it is hardcoded to use a function block called DC1. If you can imagine scenarios where you may want to use this GEM with a differently-named function block, then you might want to add more flexibility in the configuration.

b. For more flexibility, consider putting an option on the configuration form for the function block path, instead of the module tag.

GEM instance configuration:


Basic Configuration
Module / FB path: MOTOR2/EDC1

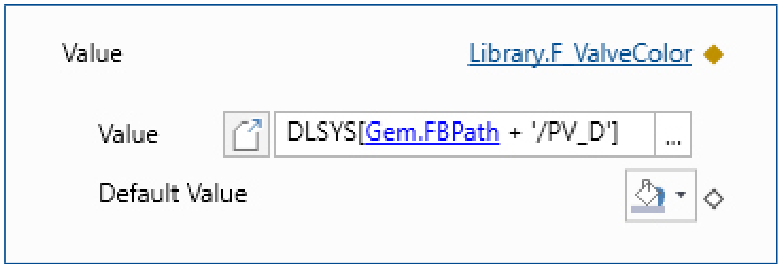
Gem class form configuration:



FBPath — Function Block Reference

Title: Module / FB

Shape color configuration:

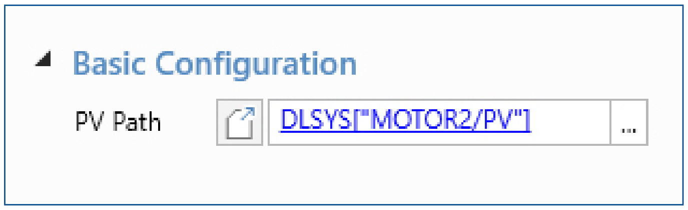


Value: DLSYS[Gem.FBPath + '/PV_D']

A benefit to this approach is that now the GEM can be used with any function block that has the parameter PV_D in it, so it is useful for both the DC block as well as the EDC block. However, it is limited in that it must be used with a function block with the parameter name PV_D specifically. If you can imagine scenarios where you may want to use this GEM with a differently-named parameter or with no function block, then you may want to add more flexibility in the configuration.

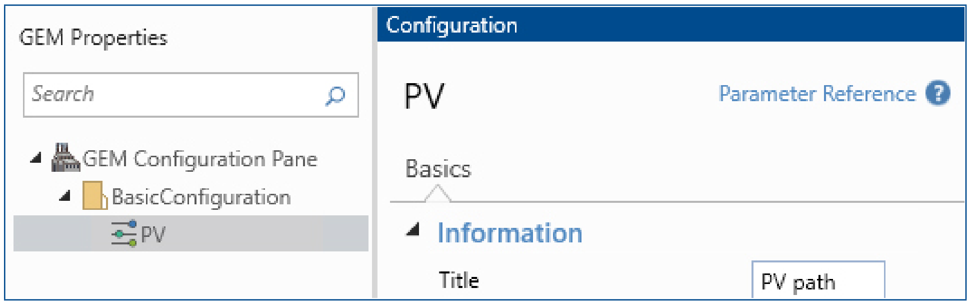
c. For even more flexibility, you might put an option on the configuration form for the parameter path, instead of the function block path:

GEM instance configuration:



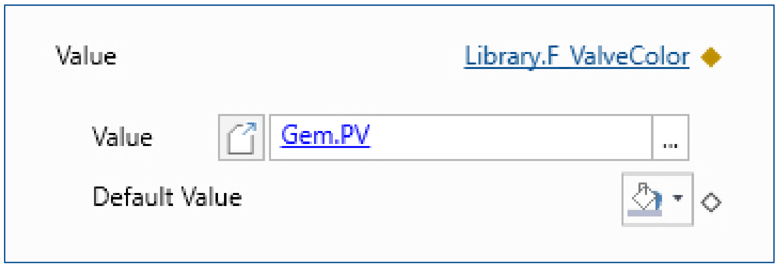
PV Path: DLSYS["MOTOR2/PV"]

Gem class form configuration:



Title: PV path

Shape color configuration:


Value: Library.F_ValveColor

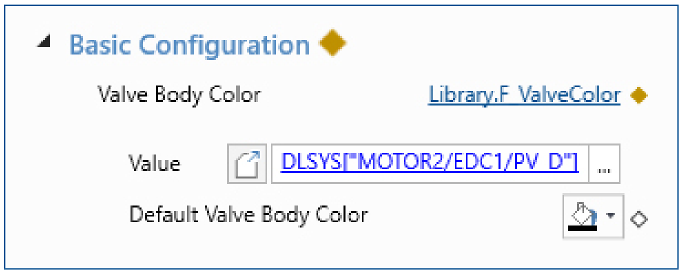
Value: Gem.PV

Default Value

A benefit to this approach is that now the GEM can be used with any discrete parameter path, including module-level parameters that aren't named PV_D. This appears to maximize flexibility while keeping the animation inside the GEM class configuration. However, this approach is limited in that the conversion function used to animate the body color is hardcoded within the GEM class. If you can imagine scenarios where you may want to use this GEM with a different color animation, then you may want to add more flexibility in the configuration.

d. For even more flexibility, you might put an option on the configuration form for the valve color itself, instead of the parameter path:

GEM instance configuration:

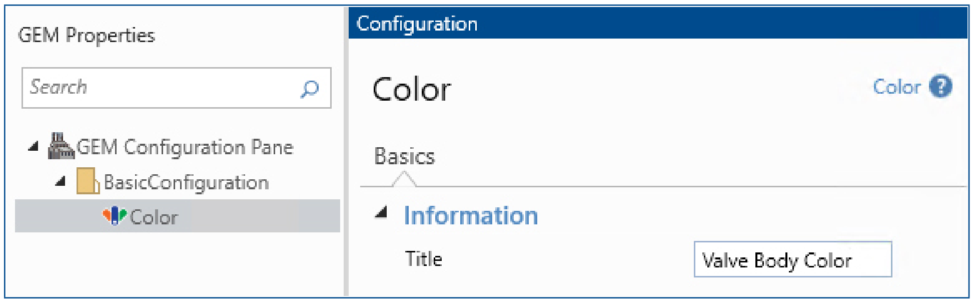


Valve Body Color: Library.F_ValveColor

Value: DLSYS["MOTOR2/EDC1/PV_D"]

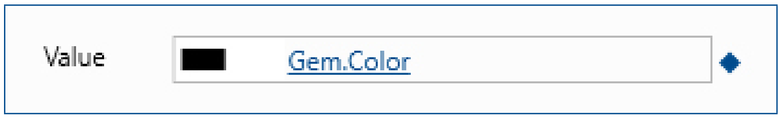
Default Valve Body Color

GEM class form configuration:



Title: Valve Body Color

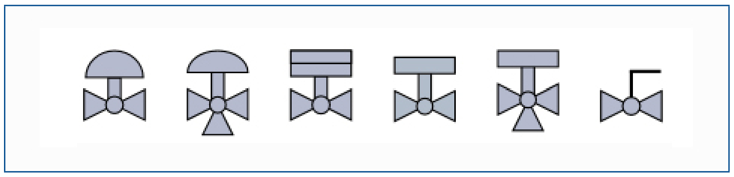
Shape color configuration:



Value: Gem.Color

A benefit to this approach is that now the GEM can be used with any color conversion function or even hardcoded colors. It has much fewer limitations than any of the other less flexible GEMs, but the flexibility has come at a cost. For each instance of this GEM class, the configuration engineer must now know and configure which conversion function or hardcoded color to use.

2. You want to create GEM classes that can be used to represent multiple valve types, as follows:



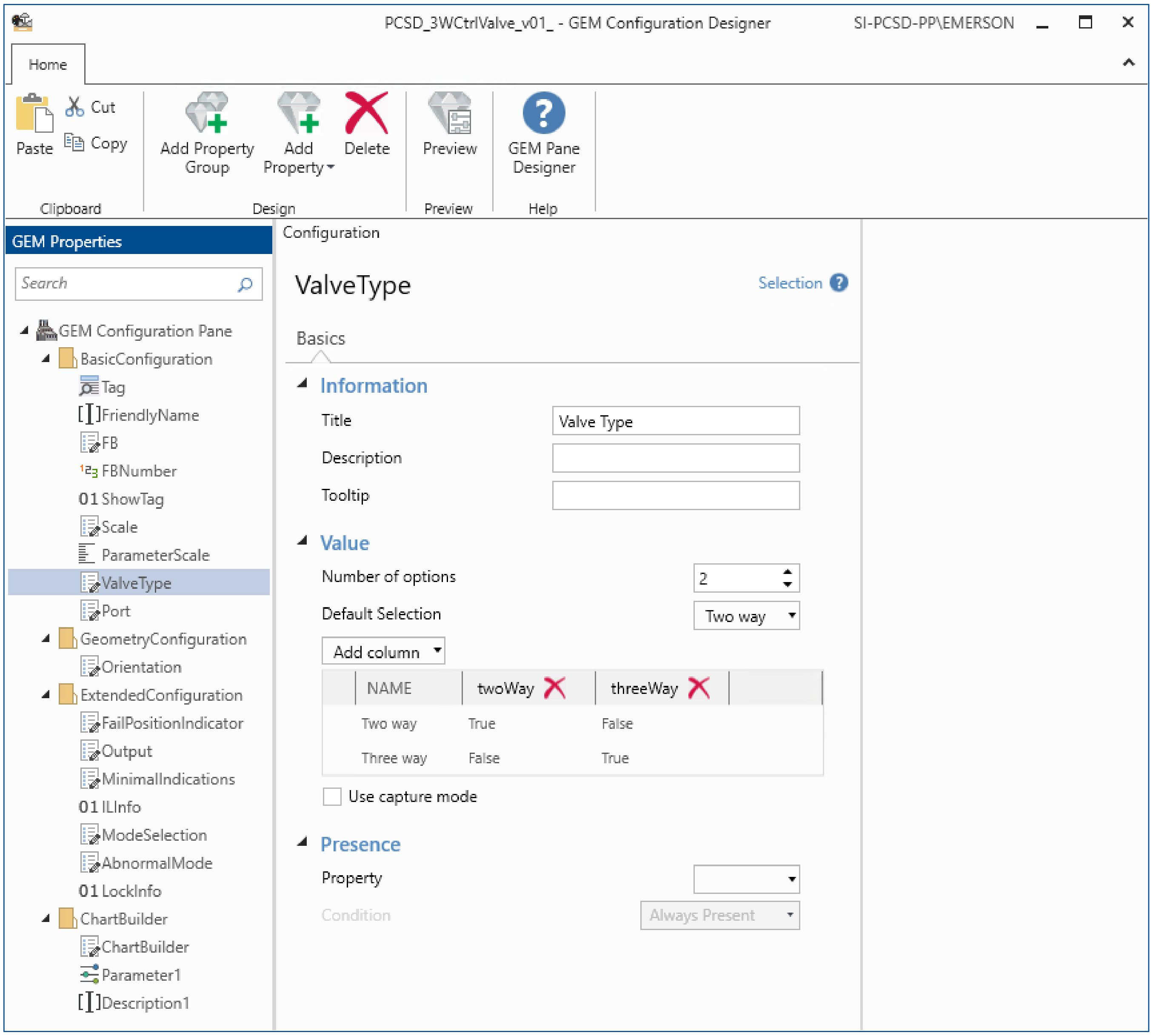
What are your options?

a. The most built-for-purpose approach would be to build each of these as a separate GEM class. This is the simplest way to provide all the required functionality, because each GEM class would have specific animations and configuration form for its own purpose, with minimal complexity built into the GEM class. However, this approach may put more of a burden on the engineer who will make use of the GEM classes to know ahead of time which ones they would need to use in each circumstance.

b. An approach that gives graphics engineers more flexibility would be to combine one or more of these together into a GEM class with configuration options to switch between the different valve types. Within this approach, there are multiple ways you might engineer the GEM classes, and each of them carries an additional degree of complexity for handling the different presentation options and corresponding animation changes.

You may decide to create a separate GEM class for each control strategy type (e.g. Analog Control Valve / Discrete Control Valve / Manual Valve). This would be useful in cases where graphics engineers know ahead of time whether a valve will be analog or discrete. Following this approach, within the Analog Control Valve GEM class and the Discrete Control Valve GEM class you would likely want to use “a GEM property” for switching between a 2 way valve and a 3 way valve. This property would be referenced by groups or GEM instances inside of the class that contain the presentation and animations that are specific to 2 way vs 3 way valves. Two notes to remember:

i. The Present Online capability is a boolean property on groups and GEM instances which must be True for the item to be present. This means if you need to switch between 2 entirely different groups depending on a selection (e.g. 2-way valve group with its own unique 2-way valve animations and 3-way valve group with its own unique 3-way valve animations), you can accomplish this with a selection property and 2 boolean subproperties. See the example for the Valve type option below.



ValveType

Information

Title: Valve Type
Description:
Tooltip:

Value

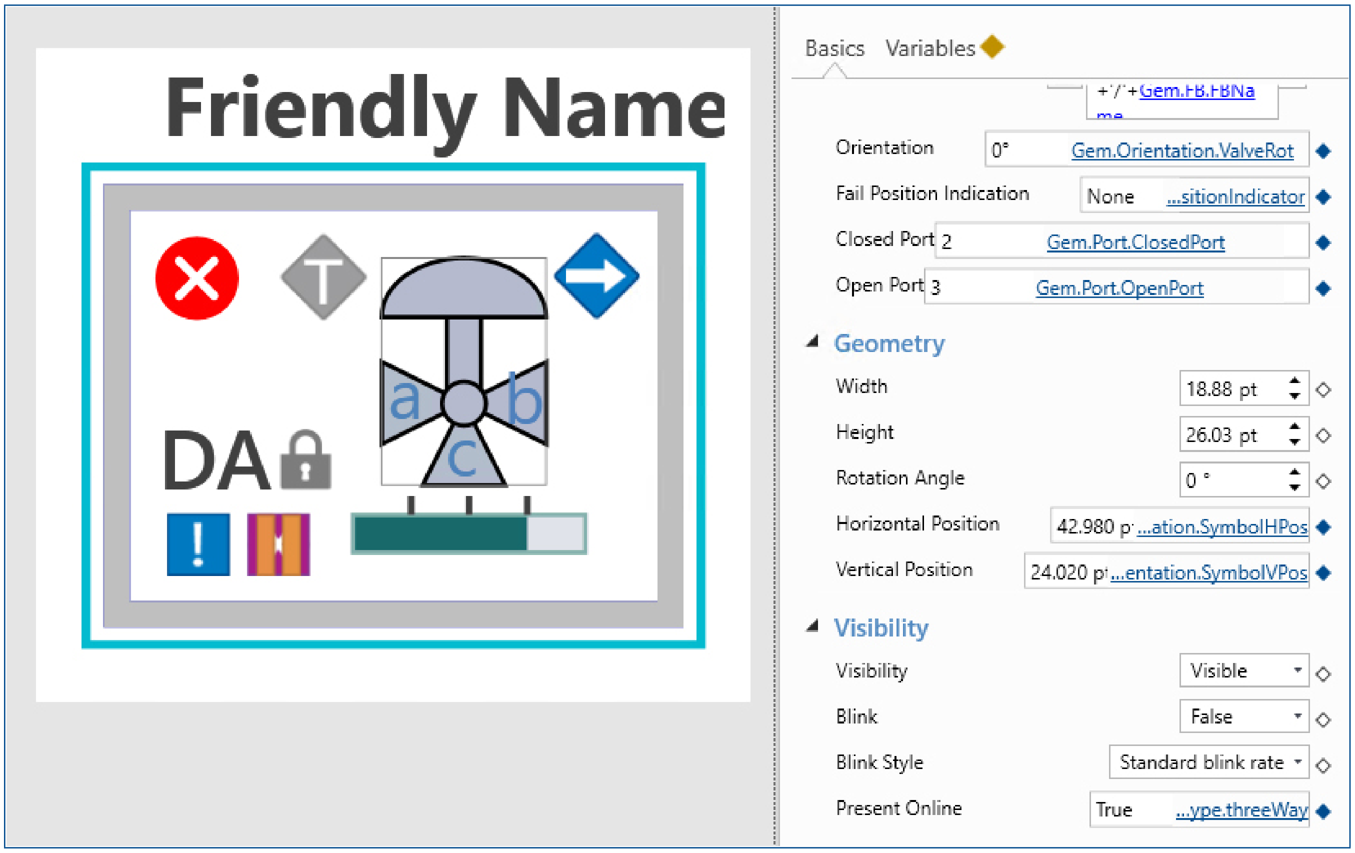
Number of options: 2
Default Selection: Two way

NAME	twoWay	threeWay
Two way	True	False
Three way	False	True

Use capture mode: ☐

Presence

Property:
Condition: Always Present



+'/'+Gem.FB.FBName

Orientation	0°	Gem.Orientation.ValveRot
Fail Position Indication	None	...sitionIndicator
Closed Port	2	Gem.Port.ClosedPort
Open Port	3	Gem.Port.OpenPort

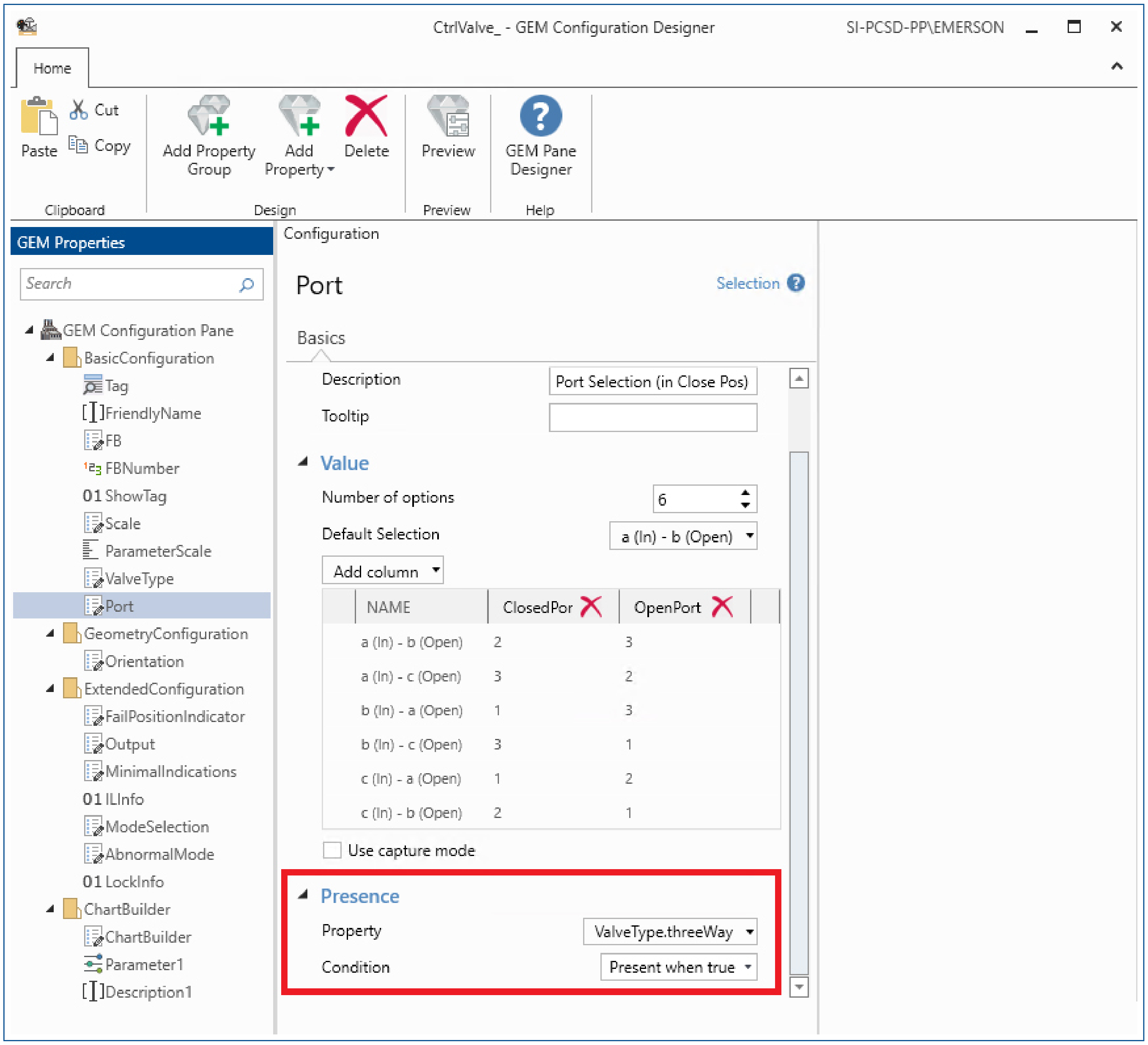
Geometry

Width	18.88 pt	
Height	26.03 pt	
Rotation Angle	0 °	
Horizontal Position	42.980 p	...ation.SymbolHPos
Vertical Position	24.020 p	...entation.SymbolVPos

Visibility

Visibility	Visible	
Blink	False	
Blink Style	Standard blink rate	
Present Online	True	...ype.threeWay

ii. The Presence setting may be used to hide or show additional GEM properties on the configuration form based on the boolean property or subproperty value of a different property. See the example of the Port selection property for our multipurpose control valve below.



Port

Basics

Description: Port Selection (in Close Pos)

Tooltip:

Value

Number of options: 6

Default Selection: a (In) - b (Open)

Add column

NAME	ClosedPor	OpenPort
a (In) - b (Open)	2	3
a (In) - c (Open)	3	2
b (In) - a (Open)	1	3
b (In) - c (Open)	3	1
c (In) - a (Open)	1	2
c (In) - b (Open)	2	1

Use capture mode

Presence

Property: ValveType.threeWay

Condition: Present when true

c. For even more flexibility, you may decide to make a single GEM class with all valve types in it. For this, you would follow much the same approach as above by making use of even more GEM properties and Presence settings configuration controlling whether different groups, GEM instances, and GEM properties' presence are turned off or on.

In this case, you may find it useful or necessary to put configuration within multiple levels of grouping, where each group is used with a Present Online reference to a different GEM property so that more complex boolean logic can be used to turn groups on and off according to the GEM property selections.

In the case of the valve types, you may imagine one GEM selection property for Control Strategy Type and then another GEM selection property for Valve Type (two way or three way). Depending on the specific configuration, you may also find that you need to duplicate the Valve Type option in case the subproperties need to be different depending on the Control Strategy Type chosen.

It is hopefully now possible to imagine how combining a large amount of configuration into a single GEM class can end up in a highly complex set of configuration of GEM class component properties, GEM properties, and GEM class structure which must be carefully orchestrated in order to achieve the desired result.

Desigining a greater amount of flexibility into the GEM class can provide benefits for the engineers who need to configure GEM instances, but results in an increasingly complex configuration for you to manage in the class.

There are advantages and disadvantages to designing fully built-for-purpose GEMs as well as designing GEMs with great flexibility. It is also worth adding here that there are other reasonable ways to add more flexibility than those shown above. In our examples, we limited the GEM instance configuration to a single property, but you could also decide to have two properties, e.g. one for the module tag and a separate one for the function block.

There is no single correct answer that dictates how much flexibility you should allow or which approach you should use. The answer for you will depend on how you want the GEMs to be used now and how you might want them to be able to be used in the future. That said, for the sake of design consistency, you should consider using the same approach for all GEMs whose purpose is similar. For instance, a best practice we have found is to use parameter path references on the configuration form for all GEMs whose purpose is to be used as a component inside of other GEM classes.

It is worth giving some thought to this early in a project and deciding on the approach that works best for your needs so that you don't have to change course later on after you've already started using the GEM in a lot of places. If you happen to be in this position and are looking for advice, see "Can I work faster by making bulk changes?" below.

Where Should I Keep My Animation Logic?

For ease of graphics maintenance, it is recommended to keep your animation logic in a consistent place in the library, as well as to minimize the number of places where a particular animation must be configured.

For instance, if a particular icon will be used to represent a common status condition in several different GEM classes, you may decide to make a separate GEM class to represent that icon and to keep all the animation logic together in the separate GEM class. That way, if you ever need to adjust the animation logic, you only have to go to one place to do so.

Alternately, you may decide to create a conversion function in order to keep the animation logic. In the same way, if you ever need to adjust the animation logic, you only have to go to one place to do so.

Depending on your preference, you may consider the behavior in either of these cases to be a benefit or a drawback.

- After adjusting the animation logic contained within a GEM class, any display that makes use of the GEM class either directly or indirectly will need to be published after the GEM class is changed. This may be considered a benefit if you prefer to systematically deploy the animation logic changes one display at a time.
- After adjusting the animation logic contained within a conversion function, the conversion function itself is the item that gets published. This may be considered a benefit if you prefer to deploy the animation logic changes all at once, and not force the operator to refresh configuration multiple times. Remember that changes can be published to a single workstation first to test them prior to rolling out the changes to all operators.

In general, either of these approaches can be viable and depends upon your preference. However, it may be worth mentioning that color animations require the use of conversion functions in handling the color table lookup.

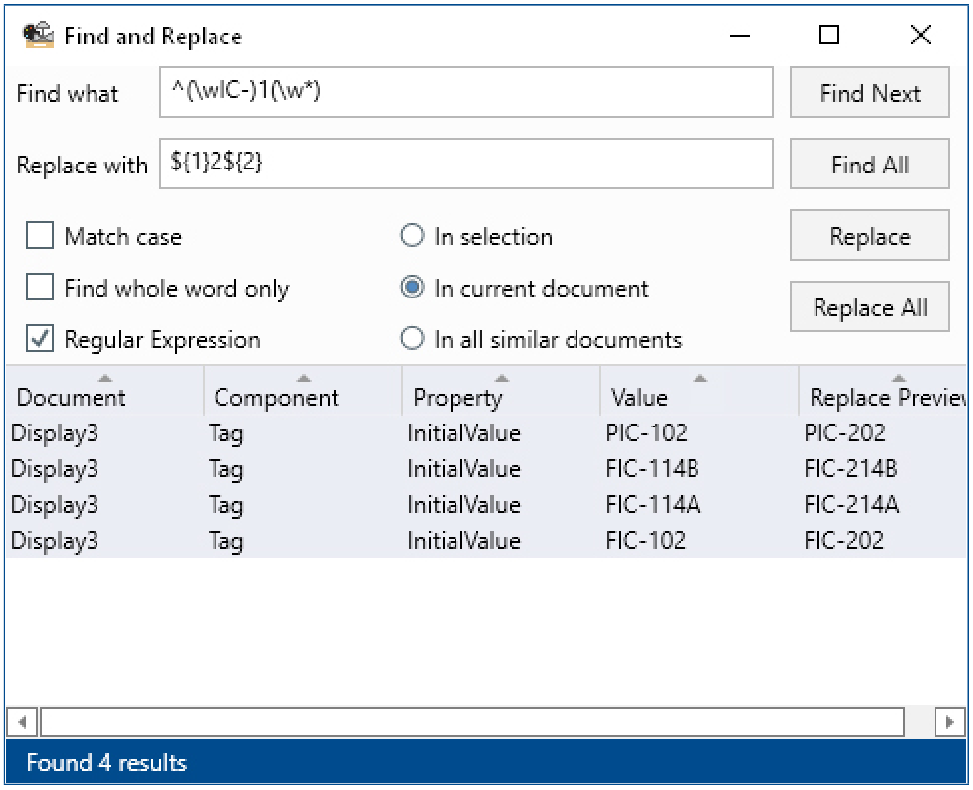
Note that when it is said above that you only have to go to one place to adjust the animation logic, it is assumed that the conversion function inputs and the GEM class properties are unchanged. In either of the above cases, changing the information that the animation logic needs to run will also require reconfiguration wherever the GEM class or conversion function is used, unless the inputs are configured to use standard values, which can be changed globally for the entire library.

Bulk Configuration Changes in Graphics Studio

Find and Replace

The built-in Find and Replace functionality may be accessed from the Ribbon or by pressing Ctrl+F. This is a very powerful tool which may be used to find and replace strings (including within string properties) control tags, parameter references, graphics expressions, or other scripting. You may change the scope of the search from a selection within a document (e.g. GEM class or display), to the whole document, or to all documents of the same type. Note that, at this time, it cannot be used with library name references.

Using the built-in Find and Replace tool, you can also make use of Regular Expressions, and with those, grouping constructs (also known as capture groups). These can help you find a wider range of string patterns and replace only the desired portion. Below is an example where you may have copied an entire display and you want to find the unit number in all of the control tags and replace it with a different unit number.



Find what: `^(\wIC-)1(\w*)`

Replace with: `${1}2${2}`

- [ ] Match case
- [ ] Find whole word only
- [x] Regular Expression

( ) In selection [✓ In current document] ( ) In all similar documents

Document	Component	Property	Value	Replace Previe
Display3	Tag	InitialValue	PIC-102	PIC-202
Display3	Tag	InitialValue	FIC-114B	FIC-214B
Display3	Tag	InitialValue	FIC-114A	FIC-214A
Display3	Tag	InitialValue	FIC-102	FIC-202

Found 4 results

Note that this is only an example of how you might use the Find and Replace dialog to accomplish a complex task quickly that may otherwise take a long time. This document is not intended to provide a full reference on how to find and replace using regular expressions, and for more information we recommend consulting the **regular expression language quick reference provided by Microsoft**.

Relink GEMs

You can also make bulk changes to displays by relinking GEM instances to a different class. This may be desired if you are introducing functionality to a certain GEM class, the change is only relevant to a subset of the instances of the GEM class, and you do not want your GEM classes to contain the added complexity of new GEM properties or new Present Online configuration OR you do not want to introduce unpublished display changes to displays where the new functionality will be turned off.

Relinking GEM instances can be done in two ways. First, you may select one or more GEM instances within a display and choose ‘Relink GEM’ from the ribbon. Then you will choose the new GEM class that you want all the instances to now reference. Finally, the display will need to be saved and published. If you choose to close the display without saving, you will cancel the relinking activity.

A second method of relinking GEMs can be done from the File > Info tab; using this tab for a GEM class, you can see references to the GEM class where it is being used, both directly and indirectly. From there, you may select one or of the direct references and click the Relink button. You will then choose the new GEM class that you want all the instances to now reference. Finally, all the displays will be updated in the background. Relinking from the GEM class File > Info tab is powerful because you can cover many displays at the same time, although note that it does not give you the opportunity to cancel. Instead, if you want to reverse the relinking activity in this way, you will need to perform the same operation in reverse, starting from the new GEM class’s File > Info tab.

While relinking using either method, all GEM property values that can be transferred will be transferred by property name. Any GEM property values that cannot be transferred will remain in configuration as unbound properties in case you want to relink back to the original GEM so that the original configuration isn’t lost.

Open Format XML export

Another way to perform bulk changes is to use a text editor or other tool for finding and replacing text across the exported XML files. This can significantly reduce the effort and time involved in certain configuration tasks.

Note that caution should be taken when performing such an operation, since the exported files can easily become broken if even one character is added or deleted incorrectly as part of a replace operation, and it is not always simple to find the error. Always maintain a backup copy of your original configuration, and do not publish displays for production until the configuration has been tested thoroughly.

Only attempt to edit the XML files directly if you are an expert and know what you are doing. Emerson does not provide customer support for tasks involving direct editing of the exported XML files.

GEM Class Performance Considerations

In general, the more that a GEM class contains or does at runtime, the greater its performance impact will be to the displays that contain instances of the GEM class.

Every GEM class impacts performance differently, based on its functionality. Each unique data reference, animation, conversion function usage, script, display element or contained GEM class, and even each grouping of elements that are already there will impact the callup times of a display.

Even for the same GEM class, it is possible that different GEM instances can impact display performance differently based upon how they are configured. This is because elements which are ‘turned off’ by virtue of the Present Online capability will not be loaded into the runtime of the display and so will not contribute to the display’s callup times . In order to minimize the performance impact of your GEMs, you should plan to use the Present Online option when building GEM classes for any piece of functionality that will be needed for some GEM instances, but not all of them.

Another consideration is the order in which animations are going to execute, especially if you have complex animations with logic that references multiple things such as standards and DeltaV parameter references. These should be ordered in such a way that the references with the biggest impacts to performance are least likely to be executed.

For example, if you have an animation such as:

Visibility = {standard value reference} && DVSYS[{DeltaV parameter reference}]

If the standard value reference evaluates as False, the DVSYS reference, which is computationally more taxing on performance, will not execute. In contrast, if the order is reversed and the DVSYS reference is first, it would always execute and may result in diminished runtime performance.

In conclusion, for complex animations with logical expressions with multiple conditions that evaluate both standard values and DVSYS references, you should evaluate the standard value before the DVSYS reference when possible.

Other factors that impact performance are discussed with the Complexity Index in Graphics Studio. These include online event handlers, control tags, and unique control parameters. We refer you to Books Online for a thorough description of these contributing factors.

FAQ

1. How do I manage changes in DeltaV Live, does it have a built-in version control?
    a. Built-in version control is on our roadmap. Until it is available, we recommend regularly exporting the graphics configuration and maintaining a separate version control for the exported configuration files.
2. Is it possible to lock down my GEM class configuration?
    a. By default, users with ‘Can Configure’ privileges have the ability to edit both displays and GEM classes in the library. However, the ability to edit displays and the ability to edit GEM classes in the library have their own Function Security properties in DeltaV Explorer, so these can be assigned to a different lock if desired. This means that you may restrict certain users from being able to edit any and all GEM classes, but individual GEM classes cannot be locked.
3. What is the best way to save changes I make to the OOB GEM library?
    a. If a user plans to make numerous changes to the OOB library, it is recommended that they create their own library, which will still leverage the OOB library.

       To achieve this, it is recommended to copy the existing OOB library into a new library as a starting point, then make modifications, although note that some changes from the OOB functionality may not be supported by Emerson.

       Also, there are items (e.g. standards, references, component GEMs) in a library that is copied from the OOB library, referenced by the GEMs that remain in the original library. While it isn’t likely that Emerson will change standards or references, the modified library would receive those updates as well.

       End users are responsible for managing their own libraries if they are modified from the OOB library.
4. How does upgrading work with GEM classes that I’ve changed? What happens when I try to import the OOB library after I’ve made modifications to it?
    a. Emerson takes precautions to ensure that any changes that may have been made to the GEMs do not get overridden with upgrades. An OOB utility is provided that will check for modifications to the OOB library during an upgrade. In the case that a user has modified GEM classes but does not move them to a new library first, the user can choose either to override existing, modified GEMs with the newly imported ones or save the modified ones in a backup folder as a copy.

5. Are GEMs resizable?

   a. The answer to this depends entirely on the configuration of the GEM class. If the GEM class contains any elements whose size or position is controlled by a GEM property, then attempting to resize the GEM instances may result in aberrant behavior. Likewise, if the GEM class contains any size, position, or rotation animations, then attempting to resize the GEM instances may result in aberrant behavior.

      Many of the out of the box GEMs are not able to be resized by the end user because they contain elements whose size or position references a GEM property and because they have size, position, or rotation animations.

      Also note that resizing the GEM class may not result in all instances being resized, so be careful when using the "resize to fit contents" if there are linked instances of the GEM being used in displays.

Contact Us
www.emerson.com/contactus