System 800xA Engineering

Process Graphics

System Version 5.1

Power and productivity
for a better world ™

ABB

System 800xA Engineering

Process Graphics

System Version 5.1

Table of Contents

Section 3 - Graphic Aspects

Section 4 - Expressions

Section 5 - Resource Management

Section 6 - Diagnostics Window

Section 7 - Faceplate Framework

Section 8 - Tutorial

Appendix A - Standard Building Blocks

Appendix B - Standard Building Blocks for AC 800M

Appendix C - User-Defined Aspect Categories

Appendix D - Generic Elements and Object Types

Appendix E - Sizes of Faceplates

Appendix F - Upgrading Faceplates

Appendix G - Graphics Builder Shortcut Keys

Index

Revision History

About This User Manual

Any security measures described in this User Manual, for example, for user access, password security, network security, firewalls, virus protection, etc., represent possible steps that a user of an 800xA System may want to consider based on a risk assessment for a particular application and installation. This risk assessment, as well as the proper implementation, configuration, installation, operation, administration, and maintenance of all relevant security related equipment, software, and procedures, are the responsibility of the user of the 800xA System.

This manual contains information about System 800xA, Engineering, Process Graphics.

User Manual Conventions

Microsoft Windows conventions are normally used for the standard presentation of material when entering text, key sequences, prompts, messages, menu items, screen elements, etc.

Feature Pack

The Feature Pack content (including text, tables, and figures) included in this User Manual is distinguished from the existing content using the following two separators:

Feature Pack Functionality _____

<Feature Pack Content>

_____

Feature Pack functionality included in an existing table is indicated using a table footnote (*) :

*Feature Pack Functionality

Unless noted, all other information in this User Manual applies to 800xA Systems with or without a Feature Pack installed.

Warning, Caution, Information, and Tip Icons

This User Manual includes Warning, Caution, and Information where appropriate to point out safety related or other important information. It also includes Tip to point out useful hints to the reader. The corresponding symbols should be interped as follows:

Electrical warning icon indicates the presence of a hazard which could result in *electrical shock.*

Warning icon indicates the presence of a hazard which could result in *personal injury.*

Caution icon indicates important information or warning related to the concept discussed in the text. It might indicate the presence of a hazard which could result in *corruption of software or damage to equipment/property.*

Information icon alerts the reader to pertinent facts and conditions.

Tip icon indicates advice on, for example, how to design your project or how to use a certain function

Although Warning hazards are related to personal injury, and Caution hazards are associated with equipment or property damage, it should be understood that operation of damaged equipment could, under certain operational conditions, result in degraded process performance leading to personal injury or death. Therefore, fully comply with all Warning and Caution notices.

Terminology

A complete and comprehensive list of terms is included in *System 800xA System Guide Functional Description (3BSE038018*).* The listing includes terms and

definitions that apply to the 800xA System where the usage is different from commonly accepted industry standard definitions and definitions given in standard dictionaries such as Webster's Dictionary of Computer Terms. Terms that uniquely apply to this User Manual are listed in the following table.

Term/Acronym	Description
Faceplate	A faceplate is an aspect that provides a graphical representation of a certain aspect object, with presentation of certain properties related to the object, and mechanism for operator interaction.
Graphic Display	A graphic display is an aspect that provides a visual presentation. It consists of static graphics representing for example tanks, pipes etc., and graphic elements that present dynamic information. Graphic displays are used to present the state of a process.
NLS	Native Language Support or National Language Support.
VBPG	Visual Basic Process Graphics.
WPF	Windows Presentation Foundation.
Process Object	A process concept/equipment, for example, valve, motor, or tank.
Graphic Item	Graphic objects used in PG2. They define a graphical representation of graphic aspects.
Input Item	Item that defines the input behavior of a graphic aspect.
Input property	A user-defined property of a graphic aspect.
Data entity	Data entities include aspect object properties, aspect views, and aspect verbs.
Edit panel	Area in the Graphics Builder where the graphic aspect is being edited.
Graphics Builder	Tool used to configure graphic aspects.

Term/Acronym	Description
Graphic aspect	A generic name for all aspects implemented by Process Graphics 2 aspect system. They are configured using Graphics Builder.
Graphic Element	Building blocks used in other graphic aspects.
Invocation Object	The object for which a graphic aspect (a graphic display or graphic element) is invoked.
Symbol	A text string of an expression that is to be resolved to a system entity like property of an aspect object, a logical color, NLS text, or to a graphic entity such as an expression variable or input property.
Aspect Object Property	A property defined by an aspect of an aspect object, which is can be accessed from Process Graphics using the data subscription facility. It is often referred to as OPC property inappropriately.
Runtime	Refers to runtime operation.
<...>	Refers to keyboard navigation. For example, <ENTER> refers to the ENTER key.

Released User Manuals and Release Notes

A complete list of all User Manuals and Release Notes applicable to System 800xA is provided in *System 800xA Released User Manuals and Release Notes (3BUA000263*).*

System 800xA Released User Manuals and Release Notes (3BUA000263*) is updated each time a document is updated or a new document is released. It is in pdf format and is provided in the following ways:

- Included on the documentation media provided with the system and published to ABB SolutionsBank when released as part of a major or minor release, Service Pack, Feature Pack, or System Revision.

- Published to ABB SolutionsBank when a User Manual or Release Note is updated in between any of the release cycles listed in the first bullet.

A product bulletin is published each time *System 800xA Released User Manuals and Release Notes (3BUA000263*)* is updated and published to ABB SolutionsBank.

Section 1 Introduction

Process Graphics provides the tools for creating a graphical user interface for supervision and operation of a site. The following can be performed in Process Graphics:

- Creating building blocks (for example, graphical representation of Tank or Valve), graphic elements that are used for building graphic displays.

- Building graphic displays that provide an overview of the site.

- Configuring faceplates that are used for controlling and monitoring the site.
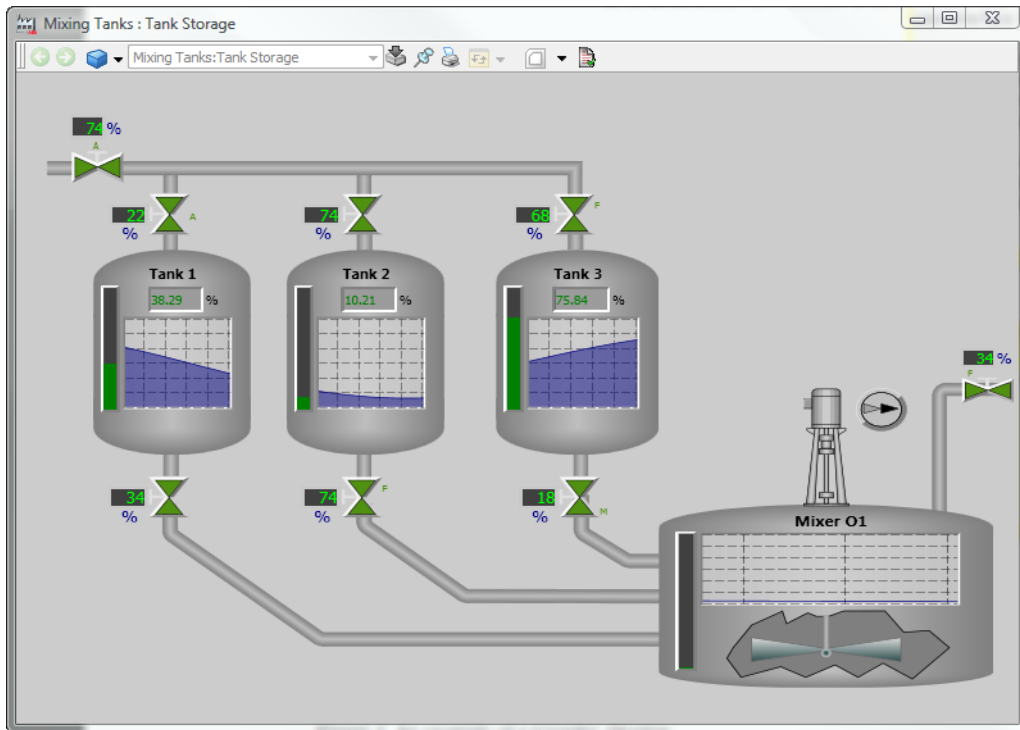
Figure 1. An example of a Graphic Display

The following are the important sections included in this manual.

*   Section 2, Graphics Builder, which describes how to build graphics and gives
    an overview of the features of Graphics Builder.

*   Section 3, Graphic Aspects, which describes components and structures of
    graphic aspects. This section also explains about object aware elements,
    generic elements, and standard input handling.

*   Section 4, Expressions, which describes the syntax of writing expressions for
    graphic aspects. Expressions are used to receive data from the system and
    present the data in graphic aspects. This section also includes information on
    the data types available for graphic aspects.

- Section 7, Faceplate Framework, which describes the procedure for configuring faceplates and faceplate elements.

- Section 8, Tutorial, which describes the workflow to create graphic aspects and to use various functionalities of Graphics Builder.

- Appendix A, Standard Building Blocks, which describes the building blocks used in Process Graphics for creating graphic aspects.

Prerequisites and Requirements

To configure the graphic aspects, the user must belong to the IndustrialITApplicationEngineer group. The user is expected to be familiar with the graphical user interface of Microsoft Windows platforms.

Process Graphics is designed to have the best performance when used on client nodes. It is possible to display graphics on server nodes. But observe that server hardware might have low performance graphic cards, which do not provide the appropriate display exchange characteristics. In addition, the overall performance of the server node may be negatively affected while calling up graphic displays.

Section 2 Graphics Builder

Graphics Builder is a tool that helps to configure graphic aspects such as graphic displays, graphic elements, and faceplate elements. The main parts of Graphics Builder are Edit Panel, Menu Bar, Tool bar, Toolbox window, Expression Editor, Properties window, Status Bar, and Element Explorer.
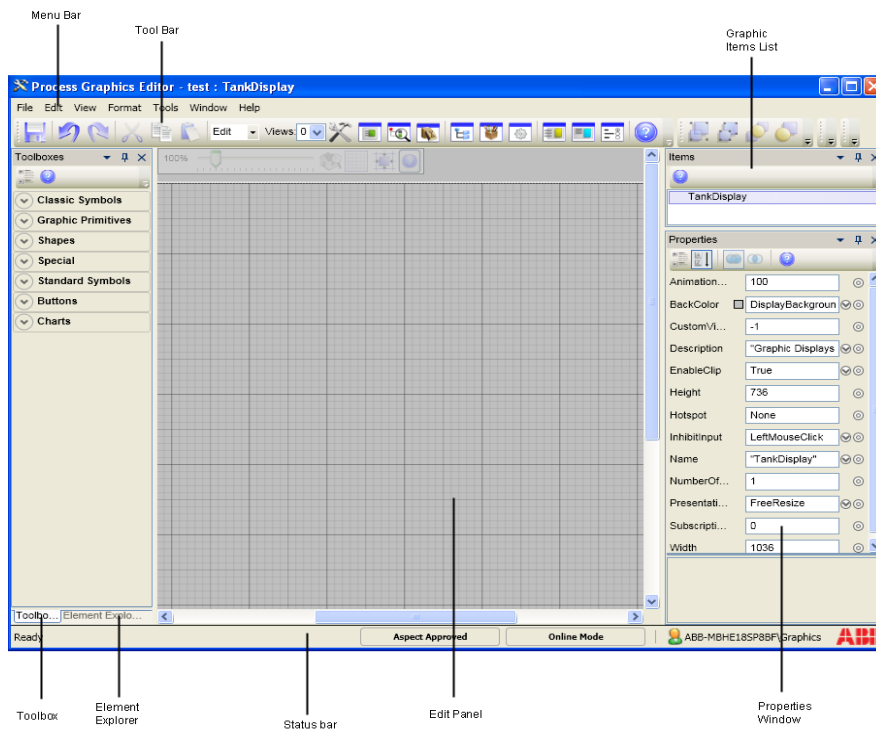
Figure 2. Graphics Builder

The edit panel allows the user to edit the graphic aspects. Graphics item list allows the user to select items in the edit panel.

Graphics Builder provides the Properties window and Expression editor to configure the properties of items. It also allows the user to format the graphic items using menu bar and tool bar.

The frequently used windows in Graphics Builder are docked windows. The state of these windows persist when the user reopens the Graphics Builder for the next time.

In Graphics Builder, the user can also create input properties, expression variables, and solution library that are accessed using the menu bar and tool bar.

Only the aspects belonging to *Process Graphics 2* category can be opened in the Graphics Builder.

Clicking allows the user to view the online help. This is available for all the features of Graphics Builder.

The following are specific features of the Graphics Builder:

- Properties Window is used to configure the properties of graphic items and input items.

- Expression Variables are used to define and edit the expression variables.

- Expression Editor is used as an advanced alternative to configure property values. It contains a richer set of data selectors than available in the Properties Window. It also helps the user to add expressions containing references to aspect object properties, resources, expression variables, or input properties.

- Toolbox Window to display the graphic building blocks defined in the system.

- Element Explorer to select and insert graphic elements in the currently edited graphic aspect.

- Input Properties to define and edit user defined properties**.**

- Solution Library to define graphic entities as reusable solutions.

- User Enumerations to create user-defined data types.

- Data References and Resource References to view or change the data-entity references or resource references in the graphic aspect.

These windows are accessed through the menu bar, toolbar, context menus, and shortcut keys.
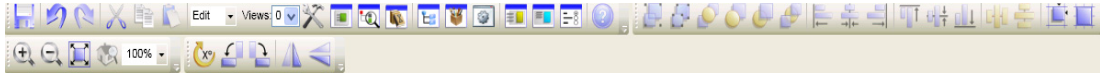
Figure 3. Toolbar of Graphics Builder

Creating a New Graphic Aspect

This section describes the procedure for creating graphic aspects. Execute the following steps:

1.   In workplace, select the object on which the graphic aspect should be created.

2.   Right-click the object and select **New Aspect** from the context menu. The **New Aspect** dialog box appears.
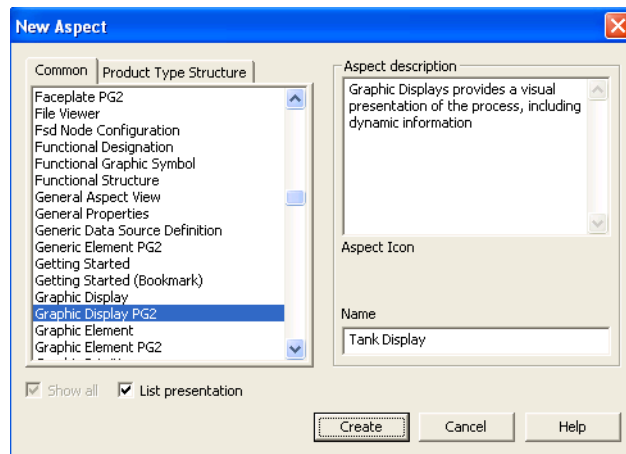
Figure 4. New Aspect

3.   Select **Graphic Display PG2** (for graphic display), **Graphic Element PG2** (for graphic element), or **Generic Element PG2** (for generic element) from the list.

4.   Type a name for the new aspect and click **Create**.

Launching the Graphics Builder

This section helps the user to launch the Graphics Builder for editing the graphic aspect. There are three different ways to start the Graphics Builder.

1.   In the workplace, right-click the graphic aspect from the aspect list and select **Edit** from the context menu.

2.   In the workplace, right-click the background of runtime view of a graphic aspect and select **Edit** from the context menu.

3.   Select **Edit** from the tool bar in the preview pane of the workplace.

To exit from the Graphics Builder, select **File > Exit** in the Graphics Builder.

For more information on creating a graphic aspect, refer to Creating a New Graphic Aspect on page 31.

Modes

Graphics Builder allows the user to edit a graphic aspect and to view the results of the graphic aspect when it is provided with dynamic values.

There are two modes available in the Graphics Builder.

- Edit

- Live

Edit mode is the default mode of the Graphics Builder. This allows the user to build the graphic aspect without any interactive connection to dynamic data. For example, if *BarValue* property of Bar item is configured to a dynamic process value, this mode provides the visual representation of the Bar without any dynamic values.

Live mode allows the user to view the behavior of graphic aspect when it is provided with dynamic values. For example, if *BarValue* property of Bar item is configured to a dynamic process value, this mode allows the user to view change in values with respect to the system value to which it is connected.

The modes are selected from the **View** menu, or from the toolbar.

XYPlot, **Trend**, and **AspectViewControl** primitives do not work in **Live** mode.

Status Bar

Status bar of the Graphics Builder displays the following:

- State of Graphics Builder.

- Aspect Approve State.

- Online/Offline Mode.

- Current User.

The state of the Graphics Builder is displayed as *Ready* when the graphic aspect is ready for modification or *Saved* after the graphic aspect is saved.

> If there are any unsaved changes to the graphic aspect, the title of the Graphics Builder will include a "*****" appended to the end of the title (For example, *Process Graphics Editor - TestObj : TestDisplay******) and the state of the Graphics Builder will be *Ready*.
>
> If the graphic aspect is saved, the state of the Graphics Builder changes to *Saved* and the "*****" is removed from the title of the Graphics Builder.

_____

The state of a graphic aspect is displayed in the status bar as *Aspect Approved* or *Aspect Unapproved*. For more information on approving graphic aspects, refer to *System 800xA, Tools (2PAA101888*)*.

The status bar specifies whether the graphic aspect is in *Online Mode* or *Offline Mode*. For more information, refer to Off-Line Engineering on page 146.

The status bar also displays the login name of current user of the system.

The status bar appears on the bottom of Graphics Builder as shown in Figure 5. Select **View > Status Bar** to control visibility of the status bar.

Figure 5. Status Bar

Docking Support

Docking support is integrated with the Graphics Builder. This helps the user to organize the **Toolbox**, **Graphic Items**, **Properties**, **Solution Library**, and **Element Explorer** windows in a flexible way. Docking supports the following modes:

• Use as a standard window (floating mode).

• Dock to any side of the existing window.

• Dock as a tabbed window inside other dock supported windows.

- Auto hide.

Figure 6 shows the Toolbox window with docking support.
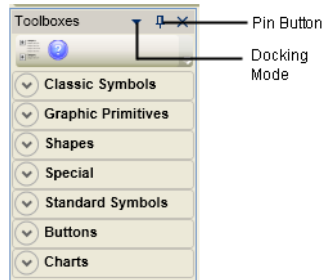
Figure 6. Dock Window

The pin button is used to toggle the docked window in Auto Hide mode, that is, the window appears as a tab along the edge to which the window is docked. Auto hidden mode is automatically restored to the original position when the mouse is hovered over or clicked on the header area of the tab.

Table 1 explains the different modes of docking.

Table 1. Docking Modes

Mode	Description
Floating	In this mode, the user can move and place the window as required.
Dockable	In this mode, the window can be docked. Dock position indicators are visible to the user at the positions where the window can be docked.

Table 1. Docking Modes

Mode	Description
Auto Hide	In this mode, the windows are auto hidden. They appear as a tab. Click the tab to view or hide the window.
Hide	In this mode, the windows appear hidden. This is equivalent to the Close button. To view the windows, select the required window from the menu bar or toolbar.

The position and state of the docked windows persist while starting the Graphics Builder for the next time.

Double-click the title bar of the window to enable floating mode for the window or to ain the default docking mode.

Select **Window > Reset Docking Layout** to reset to the default docking mode. This resets the state of all windows.

Editing Operations on Graphic Items

Graphics Builder allows the following edit operations to be done on graphic items:

• Undo and Redo.

• Cut, copy and paste.

• Delete.

• Select.
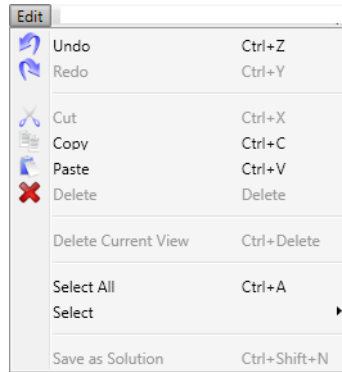
Figure 7. Edit menu

Right-click on the edit panel or graphic item to open the context menu.

The Edit operations (Cut, Copy, Paste, and Delete) can also be selected from the context menu.

Graphics Builder also allows the user to format the graphic items. The following formatting operations can be done on graphic items:

•   Grouping and ungrouping.

•   Aligning.

•   Ordering.

•   Rotating.

•   Flipping

•   Horizontal and Vertical Spacing.
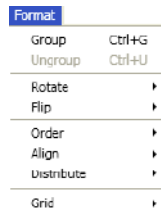
Figure 8. Format menu

Feature Pack Functionality _____
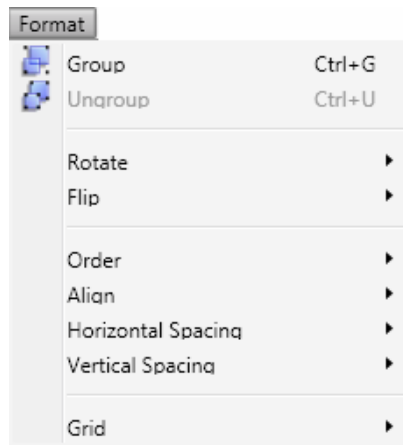
Figure 9. Format menu

_____

Right-click on the graphic item to open the context menu.

The formatting operations (Order, Rotate, Grid, and Flip) can also be selected from the context menu.

Manipulation Operation

This section describes various operations performed on the graphic items in the edit panel. It includes drag operation, using mouse or keyboard for moving a selection or moving graphic items, resizing a graphic item, and rotation of a graphic item.

Selecting a Graphic Item

There are four ways for selecting a graphic item.

1. Click the graphic item in edit panel.

2. Select the graphic item from the graphics item list.

3. Select **Edit > Select All** to select all the graphic items appearing in the edit panel.

4. Click a graphic item and select **Left**, **Right**, **Up**, or **Down** from **Edit > Select**. This selects the graphic items appearing to the left, right, top or bottom of the selected item.

Table 2 describes the different select operations and the effect on selection list in the edit panel.

Table 2. Selection using Mouse Click

Operation	Effect on the selection list
Plain selection using a mouse click	Click the required graphic item in the edit panel. The previously selected graphic item will be removed from the selection.
Selection with SHIFT	A new graphic item is added to the selection.
Selection with CTRL	The selected graphic item is removed from the selection list. If the item is not selected, it will be added to the selection list.

For more information on selecting input items, refer to Graphics Item List on page 57.

A drag operation also affects selection of graphic items in the edit panel. Drag on the edit panel and a bounding rectangle appears that specifies the drag selection area.
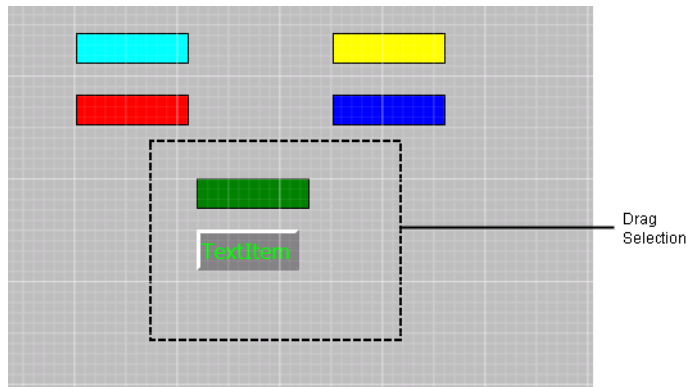
Figure 10. Selecting Graphic Items

Table 3 describes the different drag operations and the effect on selection list in the edit panel.

Table 3. Drag Selections

Operation	Effect on the selection list
Plain drag selection	Previously selected graphic items are removed from the selection.
	New graphic items are added to the selection inside the bounds of the drag selection area.

Table 3. Drag Selections

Operation	Effect on the selection list
Drag selection with SHIFT	New graphic items are added to the selection depending on the drag selection area.
Drag selection with CTRL	Selected graphic items appearing in the drag selection area, are removed from the selection list.

Moving graphic Items

Graphic items can be moved by performing a drag operation. All selected graphic items can be dragged and moved to the required position in the edit panel.

Graphic items can also be moved using the up, down, left, and right arrow keys. The move operation continues until the key is released. The items are moved in increments of grid size. The items move one pixel at a time while pressing CTRL.

For more information on the grid and snapping items to the grid, refer to Grid and Snap on page 50.

The user can change the selection of a graphic item to another selection using keyboard. Only one graphic item should be selected. Press ALT and the up, down, left, or right arrow key to move the selection to next graphic item in the edit panel.

Resizing graphic items

Points bounding the graphic item appears on selecting a graphic item.
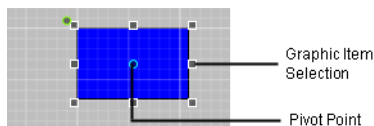
Figure 11. Resizing Graphic Items

Eight points surround the graphic item on selection. Drag any of these points to resize the graphic item.

The graphic items keep the aspect ratio on pressing SHIFT. Otherwise, dragging the corner handle allows the height and width to be changed independently.

Rotating graphic items

Most graphic items contain a rotate handle that appears on selecting the graphic item. Dragging the rotate handle rotates the graphic item around the pivot point. By default, the pivot point of a graphic item is in the middle of bounding box of the item.
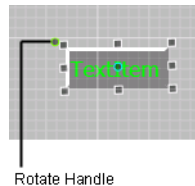
Figure 12. Rotating Graphic Items

If two or more graphic items are selected, the rotation of each item is based on the same rotation angle.

For more information on rotating graphic items, refer to Rotating a Graphic Item on page 49.

Deleting a Graphic Item

There are five ways to delete a graphic item.

1.   Select a graphic item and press DELETE.

2.   Select a graphic item and select **Edit > Delete**.

3.   Select a graphic item. Right-click on the graphic item and select **Delete** from the context menu.

4.   Select a graphic item from the graphics item list. Right-click and select **Delete** from the context menu.

Editing a Polygon, Polyline, Flexible Pipe, or High Performance Pipe

Select **Polygon**. **Polyline**, **Flexible Pipe** primitive from **Toolboxes > Shapes** or a **High Performance Pipe** from **Toolboxes > High Performance**. Draw the primitive in the edit panel as required.

For more information on drawing a polygon, polyline, or flexible pipe, refer to Polygon on page 423, Polyline on page 424, FlexiblePipe on page 422, or High Performance Pipe on page 474 respectively.

In the **Properties** window, the **PointList** property of the polygon, polyline, flexible pipe, or high performance pipe includes the points used to form the shape of the item. The points displayed in this property can be edited to modify the shape of the item.

The user can also add or delete points for polygon, polyline, flexible pipe, or high performance pipe.

Add points. Execute the following steps to add a point:

1.   Select the polygon, polyline, flexible pipe, or high performance pipe.

2.   In the polygon, polyline, flexible pipe, or high performance pipe, right-click anywhere on line to add a new point to this existing line (see Figure 13).

3.   Select **Add Point** from the context menu. A yellow colored circle appears on the line where a new point is added (see Figure 14).

Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Delete
Save as Solution	Ctrl+Shift+N
Order	▶
Rotate	▶
Grid	▶
Flip	▶
Add Point	
Delete Point	
Input Items	▶
Data References...	Ctrl+Shift+D
Resource References...	Ctrl+Shift+R

Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Delete
Save as Solution	Ctrl+Shift+N
Order	▶
Rotate	▶
Grid	▶
Flip	▶
Add Point	
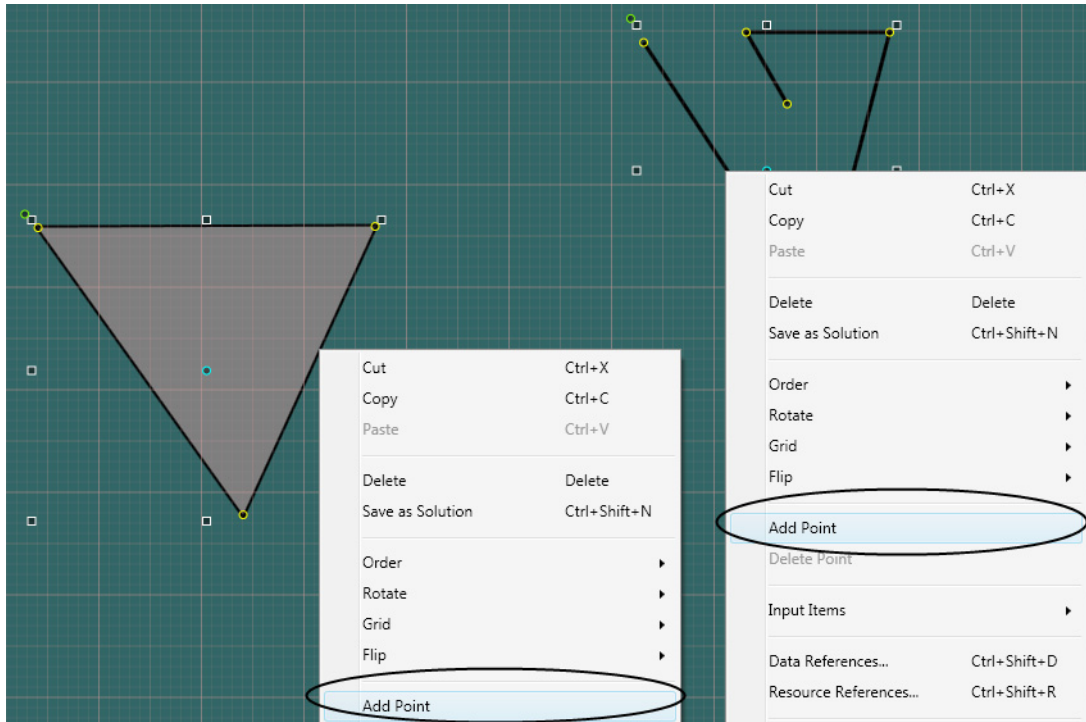
Figure 13. Context menu of a polygon/polyline
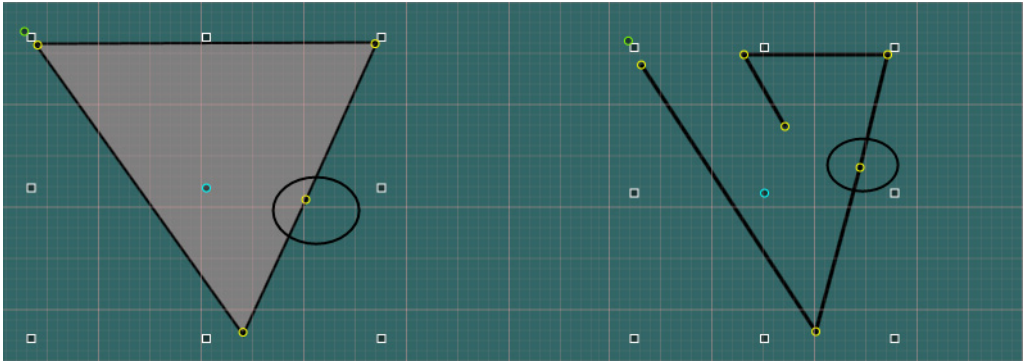
Figure 14. Adding a new point to a polygon/polyline

4. Drag the yellow colored point as required to draw the new point for the polygon, polyline, flexible pipe, or high performance pipe.

Figure 15. Polygon/polyline with added points

Delete points. Execute the following steps to delete a point:

1. Select the polygon, polyline, flexible pipe, or high performance pipe.

2.   To delete a point in polygon, polyline, flexible pipe, or high performance pipe, right-click the point (see Figure 13).

3.   Select **Delete Point** from the context menu (see Figure 14).
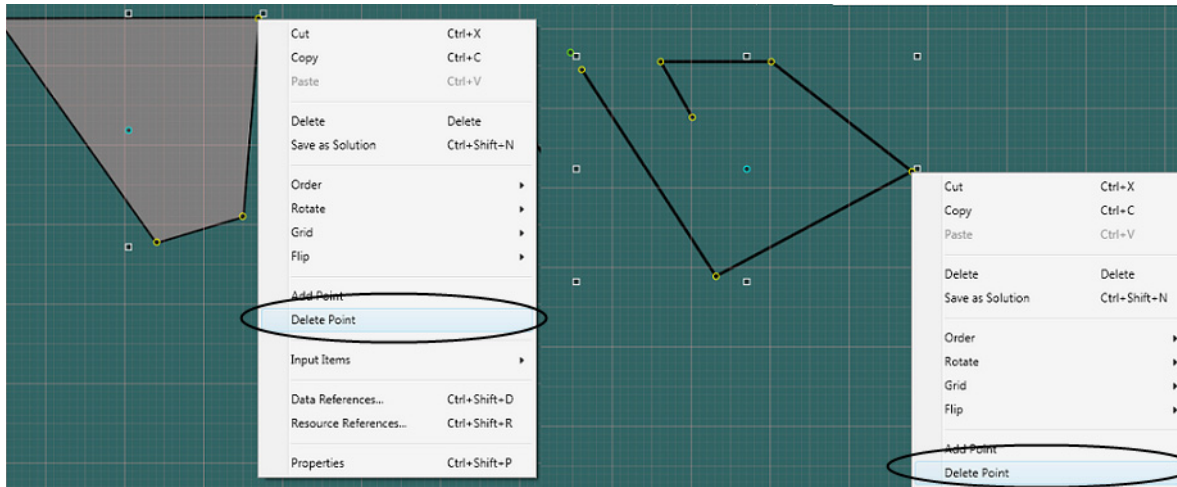
Figure 16. Context menu of a polygon/polyline
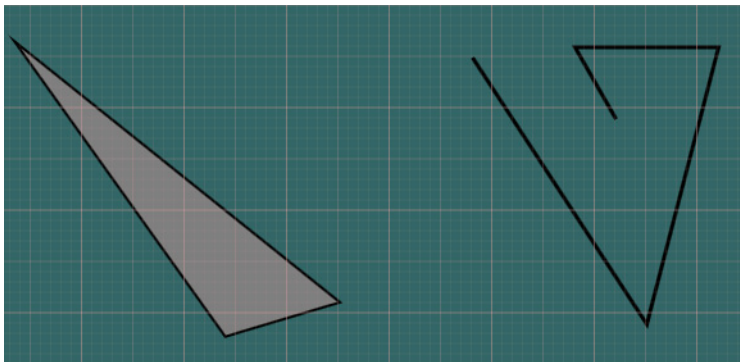
Figure 17. Polygon/polyline after deleting the point

Grouping and Ungrouping

A group arranges a set of graphic items together so that they can be configured as a single entity.

Execute the following steps for grouping or ungrouping the items:

1. Select the graphic items which are to be grouped.

2. Select **Format > Group** to group the items. To ungroup the items, select **Format > Ungroup**.

Aligning Graphic Items

The graphic items in the edit panel can be aligned left, right, or center. The items can also be aligned along the top or bottom edges or along their middle points. Alignment of graphic items is relevant only if two or more graphic items are selected. The alignment is done based on the last selected graphic item.

The last selected graphic item has the bounding rectangle in light grey.

The following are the alignments possible in Graphics Builder:

• **Lefts** aligns the graphic items to the left of the last selected graphic item.

• **Rights** aligns the graphic items to the right of the last selected graphic item.

• **Centers** aligns the graphic items horizontally along the center of the last selected graphic item.

• **Middles** aligns the graphic items vertically along the center of the last selected graphic item.

• **Bottoms** aligns the graphic items along the bottom most edge of the last selected item.

• **Tops** aligns the graphic items along the top most edge of the last selected item.

To align the graphic items:

1. Select the graphic items that are to be aligned.

2. Select **Format > Align** and select the required alignment for the item.

Ordering Graphic Items

The graphic items in the edit panel can be ordered by moving them forward or backward or to the front, or behind another graphic item. To order the graphic items:

1. Select the graphic items that are to be ordered.

2. Select **Format > Order** and select the required option to keep the item in an order.

Flipping Graphic Items

The graphic items in the edit panel can be flipped vertically or horizontally. To flip the graphic items:

1. Select the graphic item.

2. Select **Format > Flip** and select the required option to flip the item.

Distributing Graphic Items

The graphic items in the edit panel can be distributed equally in a horizontal or vertical manner. To distribute graphic items:

1. Select graphic items that are to be distributed.

2. Select **Format > Distribute**.

3. Select **Distribute Horizontally** to distribute the graphic items equally between the leftmost and rightmost graphic item.

   Select **Distribute Vertically** to distribute the graphic items equally between the topmost and bottom most graphic item.

> This section is not applicable for System 800xA 5.1 Feature Pack 4 or later versions.

Horizontal or Vertical Spacing of the Graphic Items

> This section applies to the System 800xA 5.1 Feature Pack release only.

The graphic items in the edit panel can be distributed equally in a horizontal or vertical manner. To distribute graphic items:

1. Select graphic items that are to be distributed.

2. Select **Format > Horizontal Spacing** for distributing the graphic items horizontally.

    – Select **Horizontal Spacing Make Equal** to distribute the graphic items equally between the leftmost and rightmost graphic item.

    – Select **Increase Horizontal Spacing** or **Decrease Horizontal Spacing** to increase or decrease the horizontal spacing respectively, between the graphic items. The spacing is changed based on the value specified in **Gridline Spacing** in **Tools > Options** (see Graphics Builder Settings on page 106).

    – Select **Remove Horizontal Spacing** to remove the horizontal spacing between the graphic items.

    Select **Format > Vertical Spacing** for distributing the graphic items vertically.

    – Select **Vertical Spacing Make Equal** to distribute the graphic items equally between the topmost and bottom most graphic item.

    – Select **Increase Vertical Spacing** or **Decrease Vertical Spacing** to increase or decrease the vertical spacing respectively, between the graphic items. The spacing is changed based on the value specified in **Gridline Spacing** in **Tools > Options** (see Graphics Builder Settings on page 106).

    – Select **Remove Vertical Spacing** to remove the vertical spacing between the graphic items.

Rotating a Graphic Item

Graphics Builder allows the rotation of graphic items based on a specified rotation angle. It also allows to create a copy of the graphic item. Copy of the graphic item is rotated to the specified rotation angle and then added in the edit panel.

To rotate a graphic item:

1. Select the graphic item to be rotated.

2. Select **Rotate By** from **Format > Rotate**.
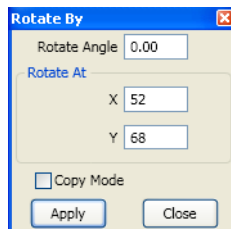
Figure 18. Rotate By

3.   Type the **Rotation Angle** in degrees.

4.   Specify the point of rotation in **Rotate At**. Type the values for X and Y coordinates.

5.   Select **Copy Mode** to create a copy of the graphic item.

6.   Click **Apply** to apply the settings.

To rotate the graphic item 90 degrees to left or right, select **Left** or **Right** from **Format > Rotate**.

Grid and Snap

The edit panel of Graphics Builder contains a grid which helps the user to position and resize the graphic items placed into the edit panel.
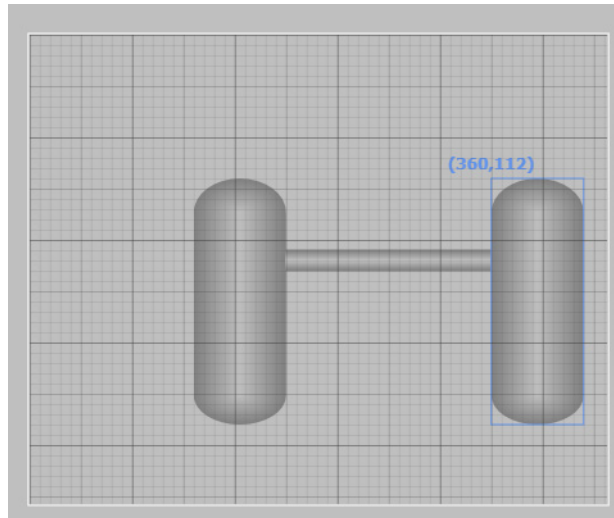
Figure 19. Edit panel with the Grid

The visibility of grid and snapping of graphic items to the grid is controlled using **Zoom Control** or through **Tools > Options**.

Snap to grid allows the user to position the graphic items to the grid.

If snapping is enabled, this setting can be overridden by pressing CTRL during a move or resize operation.

Zoom In and Zoom Out

The Zoom option is used for giving an enlarged or reduced view of the edit panel. Select the options from **View > Zoom**.

Zoom In provides an increased zoom level for the edit panel.

Zoom Out provides a decreased zoom level for the edit panel.

Fit to window fits the graphic aspect to the size of the window.

Home View urns to the default view of the edit panel.

Zoom Control

The existing zoom factor is displayed in Zoom Control. This is placed on the upper-left of edit panel by default.

Figure 20. Zoom Control

The visibility of this option is configured through **Tools > Options**. For more information on this configuration, refer to Graphics Builder Settings on page 106.

Use the slider to enlarge or reduce the view for edit panel.

Click 🔄 to return to the default view of edit panel.

Click ⊞ to view or hide the grid lines in the edit panel.

Click ⊡ to turn off or turn on snapping of the graphic items to the grid.

Copying and Pasting a Graphic Item

To copy and paste a graphic item:

1. Select the graphic item to be copied.

2. To cut a graphic item, select **Edit >Cut**.

    To copy a selected item, select **Edit > Copy**.

3. Select **Edit > Paste**. A copy of the selected item appears on the edit panel.

    A paste operation can be done after selecting **Cut** or **Copy**.

To paste a graphic item in another graphic display/graphic element:

- Execute Step 1 and Step 2 for copying the graphic item.
- Open the graphic display/graphic element to paste the graphic items.
- Execute Step 3 to paste the graphic item. The following dialog appears if there are any dependencies for the copied graphic items (for example, expression variables, or input properties).
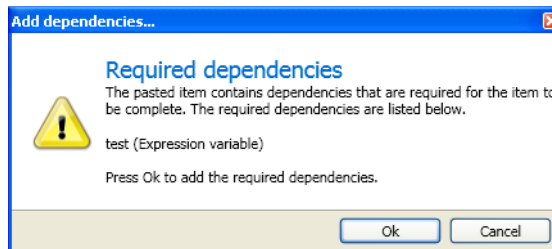
- Click **Ok** to copy the dependencies of the graphic item.

To perform the undo and redo operations, select **Edit** > **Undo** or **Edit** > **Redo**.

Positioning of pasted items depends on how the paste operation is activated.

Selecting **Paste** from the context menu positions the graphic item in relation to the place where right-click was performed.

Selecting **Edit > Paste** pastes the graphic item below the original position of the item.

Toolbox Window

The toolbox in Graphics Builder contains a set of graphic items that are used to configure a graphic aspect.

A toolbox can be created by defining custom Graphics Toolbox and Graphics Toolbox item objects in the **Graphics Structure**. For more information on creating toolbox, refer to Creating Generic Elements on page 641.

Select **View > Toolboxes** to view the toolbox.
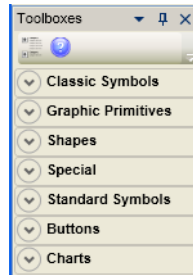
Figure 21. Toolbox

Click to group the toolboxes based on the product name.
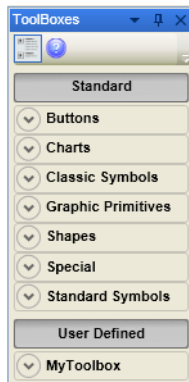
Figure 22. Toolbox after grouping

- The toolboxes from the base is grouped as **Standard**.

- The user defined toolboxes are grouped as **User Defined**.

- The toolboxes of system extensions are grouped as the respective system extension.

The toolboxes to be displayed in **Toolboxes** are selected using **Options** in the **Tools** menu. For more information on the toolboxes, refer to Appendix A, Standard Building Blocks.

Selecting Graphic Items from the Toolbox

This section helps the user to add graphic items into the edit panel. There are four ways to add a graphic item into the edit panel.

1. Click the graphic item. This enables the drawing tool cursor that allows the user to draw the item on the edit panel.

2. Double-click the graphic item to add the item on the edit panel.

3. Select the graphic item and press ENTER.

4. Drag and drop the graphic item.

Element Explorer

Element Explorer allows the user to browse through different structures and select graphic elements to be added into a graphic aspect. It consists of an object browser and an element selector.

The object browser displays a tree structure containing the system structures. The user can browse for the required object.

The element selector displays all the graphic elements included in the selected object.

To select a graphic element from the **Element Explorer**:

1. Select **View > Element Explorer**. The element explorer window appears to the left of edit panel.
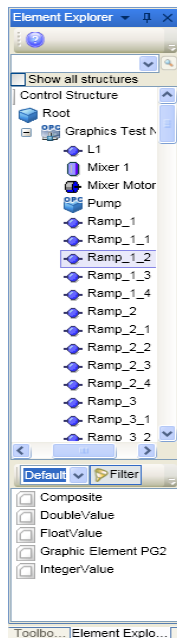
Figure 23. Element Explorer

2. In **Object Browser**, select the structure. The objects available within the structure are displayed. **Element Selector** displays all the elements available within the selected object.

    a. Select **Show all structures** to display all the structures available in the system.

    b. Select **Default** to have a default view of the element list. Select **Preview** to display the preview of all elements belonging to the selected object.

    c. Click **Filter** to filter the hidden graphic elements. This is configured using the **Browsable** property of graphic aspects. For more information on the properties of graphic aspects, refer to Table 14.

3. Select the graphic element that is to be added to the graphic aspect, from the **Element Selector**.

- Select the graphic element and click on the edit panel.

- Double-click the selected graphic element.

- Drag and drop the selected graphic element into the edit panel.

- Select the graphic element and press ENTER.

> The user is allowed to search for a specific element using the Element Explorer. Element Explorer also allows to search for elements using object names. All elements belonging to the specified object will be displayed.
>
> Type the name of the element or object to be searched and click 🔍 .

Feature Pack Functionality _____

> To replace an existing graphic element, select the graphic element in the edit panel and enter the new element name in the **Element** property in the **Properties Window**.
>
> Provide the element name using the following syntax:
>
> <ObjectName>:<GraphicElementName>
>
> The object and the graphic element must exist in the system without any duplicates.

_____

Graphics Item List

A list of all graphic items and input items added to the edit panel is displayed in the graphics item list. This list is in the form of a tree structure having the root node as the graphic aspect. The top node represents the graphic aspect itself. On selecting the aspect, the user can configure its properties in the **Properties** window.

The input items added for a graphic item is displayed as a child node of the respective graphic item.

The element hosted input items are displayed as child nodes of *Element Hosted Input Items*.

By default, this is displayed on the top-right of the Graphics Builder.
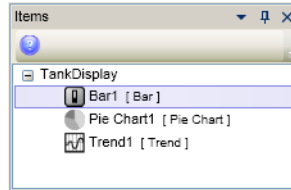
Figure 24. Graphics Item List

The graphics item list is opened from **View > Graphic Items**. The items list has the following features:

- To select graphic items from the list.

- To delete the graphic items.

- To add, delete, and select input items.

Use SHIFT or CTRL to select multiple graphic items.

Right-click a graphic item to display the context menu. Figure 25 shows the context menu of a graphic item.
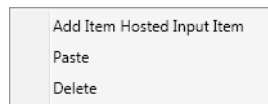
Figure 25. Context menu of Graphic Item

- Select **Add Item Hosted Input Item** to add an input item for the graphic item. Also refer to Adding an input item on page 59 for the procedure to add an input item.

- Copy an item hosted input item and select **Paste** to add this input item for the selected graphic item.

- Select **Delete** to delete the graphic item.

Right-click the graphic aspect to display the context menu. Figure 26 shows the context menu of the graphic aspect.

Add Element Hosted Input Item   ▶
Paste
Force Visible
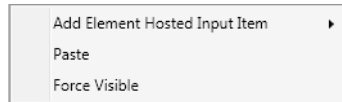
Figure 26. Context menu of Root Node

- Select **Add Element Hosted Input Item** to add an input item for the graphic aspect. Also refer to Adding an input item on page 59 for the procedure to add an input item.

- Copy an element hosted input item and select **Paste** to add this input item for the graphic aspect.

- Select **Force Visible** to force the visibility of all graphic items. Any expression assigned to **Visible** property of the graphic items that evaluates to **False** will be ignored.

The graphic display / graphic element currently being edited, cannot be selected along with other items or deleted from the items list.

Adding an input item

Perform the following steps to add an input item.

1. Select the graphic item.

2. Right-click and select **Input Items** from the context-menu.

3. Click **Create** from **Item Hosted** or **Element Hosted**.

4. Select the required input item from the input item list. For more information on graphics item list, refer to Graphics Item List on page 57.

Properties Window

Graphic items and input items possess various properties that define their appearance and behavior. Each property is associated with a data type. The **Properties** window displays these properties for a selected item.

Complex values or expressions for the properties should be done using the **Expression Editor**. The **Properties Window** is used for quick and efficient editing of expressions.

For example, if a value 70 should be assigned to the *Height* property of an item, use the **Properties Window**. Use the **Expression Editor** to assign any complex expression for this *Height* property.

The values of properties can be changed by typing value for a specific property or by invoking the **Expression Editor** on clicking ◎ . The user can also assign expression variables for properties.

There are three ways to open the Properties window.

1.   Select **View > Properties**.

2.   Right-click a graphic item and select **Properties** from the context menu.

3.   Press <F4>.
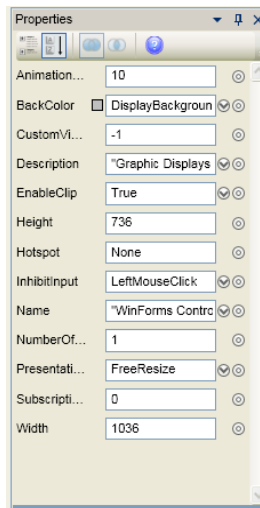
Figure 27. Properties Window

Properties of an item can also be viewed by selecting the graphic item or input item from the graphics item list on the top-right of the window. For more information on graphics item list, refer to Graphics Item List on page 57.

The following are the features of Properties window:

• A toolbar with buttons to sort the properties alphabetically or categorically.

  – **Categorized**, which sorts the properties based on categories. Each property is associated with a category such as Appearance or Behavior.

  – **Alphabetical**, which sorts the properties in an alphabetical order, based on the property name.

• Lists the union or intersection of properties.

• Use ⊚ to open a Type Editor that helps the user to select values for properties.

• Use ⊚ to open the Expression Editor to set expressions for properties.

• Possible to copy / paste expressions from Properties window.

• Possible to write a value or expression for a property.

- Click the property name to list the enumerated values of the property, if any (For example, the **Frame3dEffect** property of a **Text** primitive has values **Flat**, **Raised**, and **Sunken**, which will be listed on clicking the property name).

- A description for each property can be viewed at the bottom of the window or as a tooltip while selecting a property.

Feature Pack Functionality _____

- If the user specifies a value without double quotes (" ") for any property of the **String** data type, double quotes is automatically appended to the specified value if it is not resolved.

  For example, if the user specifies a value *InputValue* without the double quotes to the **Text** property of an item, the value changes to *"InputValue"* if there are no Expression Variables, Input Properties, or any other properties of the invocation object having the same name.

- If the user clicks the value corresponding to a property, the entire value is selected.

- If a property is selected for an item, the same property will remain selected if it exists for another selected item.

  For example, consider the items **Text** and **Scrollable Text** added to a Graphic Display. If the *Font* property of the **Text** item is selected, the same property is selected if a **Scrollable Text** item is selected.

_____

If the user enters wrong value for a property in the **Properties** window, an error message appears.
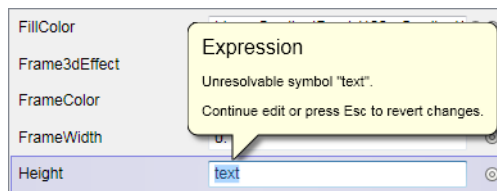
The user can then type the correct value or press ESC to get the previous value.

The following are the keyboard navigations available in the **Properties** window.

- Use the Up and Down arrow keys to move between the different properties.
- Press the letter key that corresponds to the first letter of the property to highlight the property. For example, in a **Text** primitive, press <H> to select the **HAlign** property. Pressing <H> again selects the **Height** property.
- Press <Tab> to move the focus to the text box corresponding to the highlighted property.
- Press <F4> on a highlighted property to invoke the Expression Editor.
- Use the Right arrow key to invoke the Type Editor.

Type Editor

Type Editor is a drop down list that displays enumerated values for the specific property. It provides selectors and editors that are specific to the data type of the property. These selectors and editors are used for a quick entry of values that are specific for the type of the selected property to create complete expressions.

Click ⊘ corresponding to a property to invoke the Type Editor.

The following are the keyboard navigations available for the **Type Editor**.

- Press <Esc> to close the Type Editor.
- Press <CTRL + Tab> to toggle between the tabs.

For example, Figure 28 shows a type editor for Brush data type.
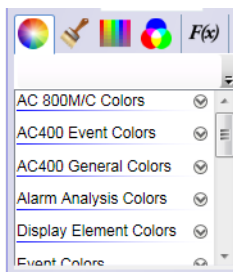
Figure 28. Type Editor for Brush

This type editor contains the following selectors.

- Logical Colors

- Logical Brushes

- Constant Colors

- RGB Editor

- Functions

Logical Colors displays the colors available in the 800xA system. Click ⊚ corresponding to a color category to display the list of colors.

Logical Brushes displays the brushes referenced from a resource library.

- The **Brush Resources** tab lists the available brush resources.

- The **Brush Editor** tab allows the user to create a brush resource.

    – *Color Resources* is same as the **Logical Colors** selector.

    – *Color Editor* is used for creating different brushes such as Solid Color, Linear gradient, Radial gradient, Hatch, or Image.

Constant Colors displays all the colors available in Microsoft Windows.

RGB Editor is used to select a color using the RGB editor.

Functions displays the expression functions available for the data type of the property. Clear the selection in **Display matching types only** check box to display all the expression functions. For more information on expression functions, refer to Expression Functions on page 233.

Complex type editors are invoked through the Expression Editor.

Expression Editor

Expressions are simple constants or complex statements that contain references to expression variables, input properties, object properties, data or resource references. The system data can be accessed through expressions.

For information on the syntax, data types, expression functions, and examples of expressions, refer to Section 4, Expressions.

The **Expression Editor** helps the user to create expressions. It is used for assigning expressions to the properties of graphic items and input items. It is also used for assigning expressions to expression variables.

It also includes the basic and complex type editors when compared to the Properties Window which displays only the basic type editor. For more information on type editors, refer to Type Editor on page 63.
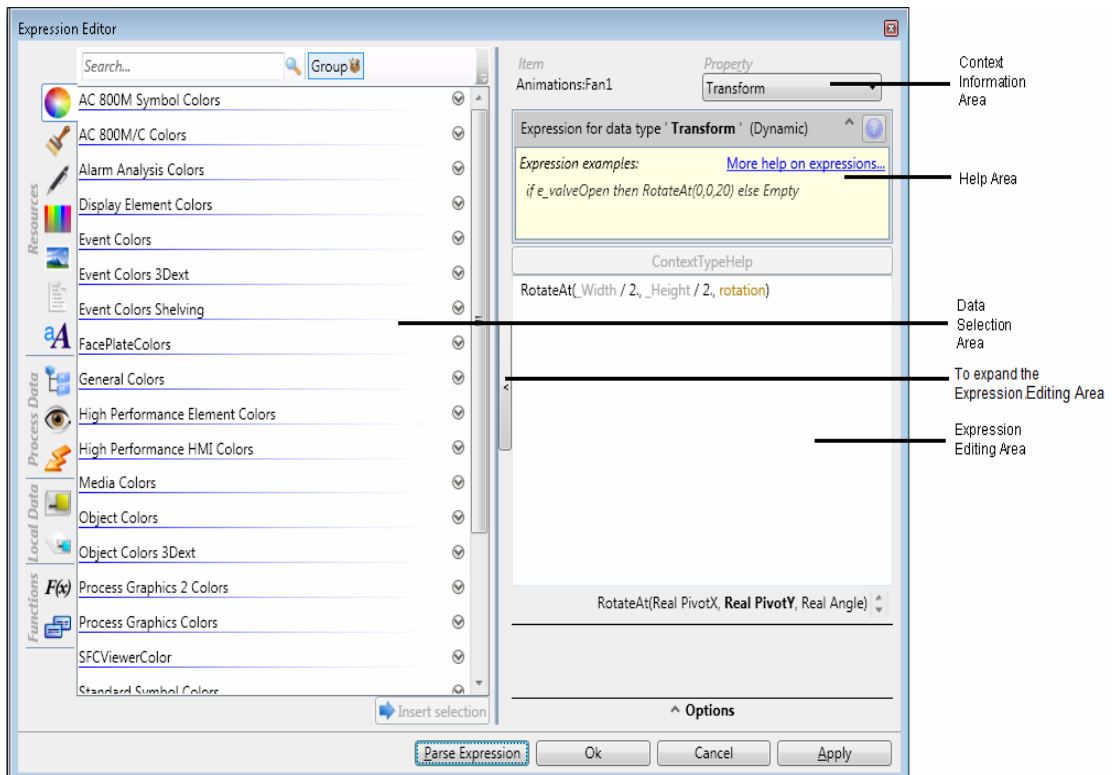
Figure 29. Expression Editor

The **Expression Editor** window is invoked through:

- Properties Window

  For more information on assigning expressions to properties, refer to
  Properties Window on page 60.

- Expression Variables Window

  For more information on expression variables, refer to Expression Variables on
  page 75.

- Usage Window

  Usage window displays the list of graphic items or input items using an
  expression variable. The user is allowed to view or modify an expression using
  this window.

- Press <CTRL + SHIFT + X> on a property text box to open the expression
  editor.

The **Expression Editor** is divided into four areas.

1. Expression Editing Area

2. Data Selection Area

3. Context Information Area

4. Help Area

Invoking the Expression Editor

Graphics Builder allows the user to create expressions and assign them to the
properties of graphic items and input items through the **Properties** window.
Execute the following for assigning expressions:

1. Select a graphic item or input item from the edit panel or graphics item list
   window.

2. Select **View > Properties**. The property window for the selected item appears.

3. Click ⊚ to open the **Expression Editor**. Refer to Expression Editor on page
   64 for adding expressions.

Expression Editing Area

The expression editing area is used to write expressions for a specific property or expression variable.
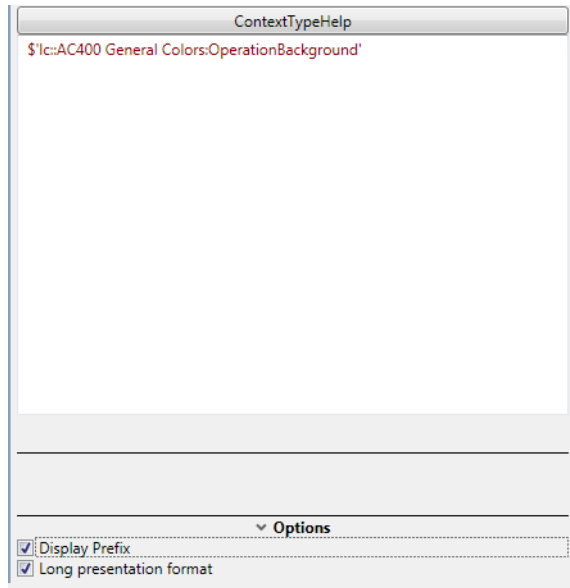
Figure 30. Expression Editing Area

Press <CTRL + Space> to display the enumerated values available for the property.

Press <F4> to open the Type Editor based on the existing data type.

Feature Pack Functionality _____

Click ▌ to expand the Expression Editing Area (see Figure 31).
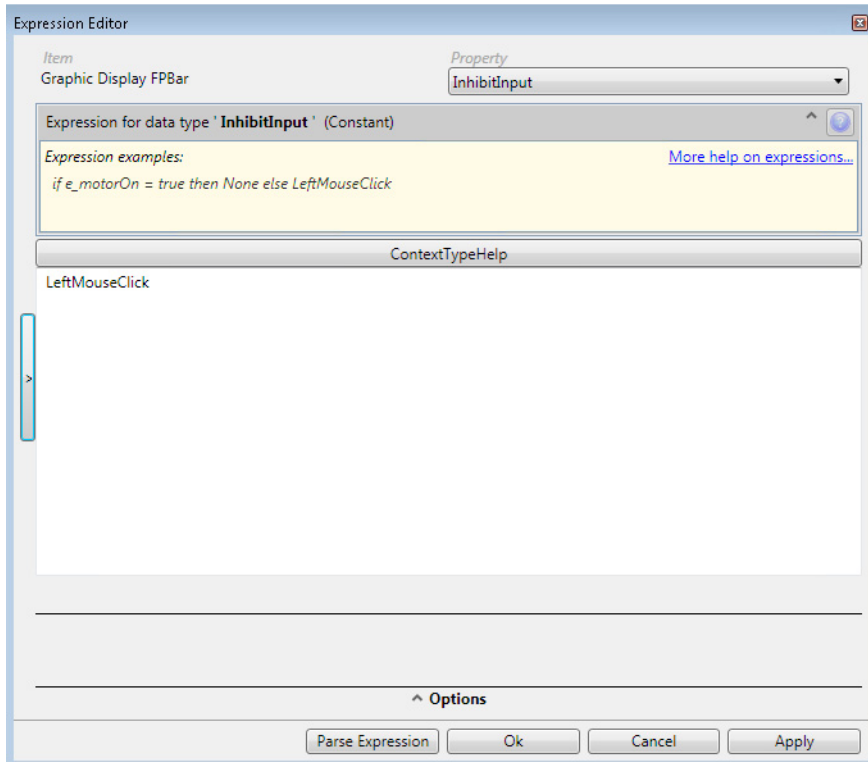
Figure 31. Expression Editing Area - Expanded

The icon changes to   .

Click this to restore the size previously set for the Expression Editing Area (see Figure 32).
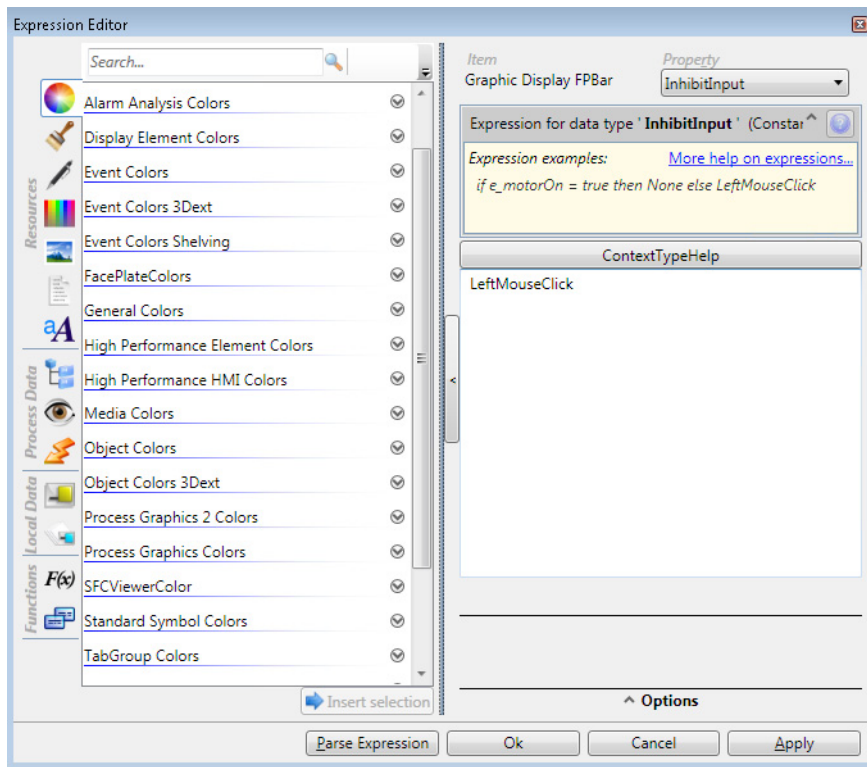
Figure 32. Expression Editing Area - Restored

Click **ContextTypeHelp** to open the Type Editor corresponding to the data type of the property. This is enabled only for the properties containing ⊚ icon.

Options contains a list of formats that controls the appearance of expression in the editing area.

Select **Display Prefix** to display the prefixes before the known symbols. For example, if the expression has a reference to an input property "iptest", the expression will be,

> ip::iptest

where, "ip::" is the prefix used for input properties. For more information on the prefixes used in expressions, refer to Expression Syntax on page 184.

Select **Long Presentation Format** to display the expression in a detailed manner. For example, if the expression contains a reference to an aspect property, the object name, aspect name, and property of the aspect being referred, is displayed in the expression.

When the user writes the expression, it is automatically parsed for errors. The user can also click **Parse Expression** to check the correctness of the expression. The errors contained in the expression is displayed below the Expression Editing Area.

Click the **Apply** button or press ENTER to save the changes to the expression, and assign the expression for the property.

Data Selection Area

The data selection area allows the user to browse for system values, input properties, variables, and resources, which are added to the expression. This area includes:

- Resources

- Process Data

- Local Data

- Functions
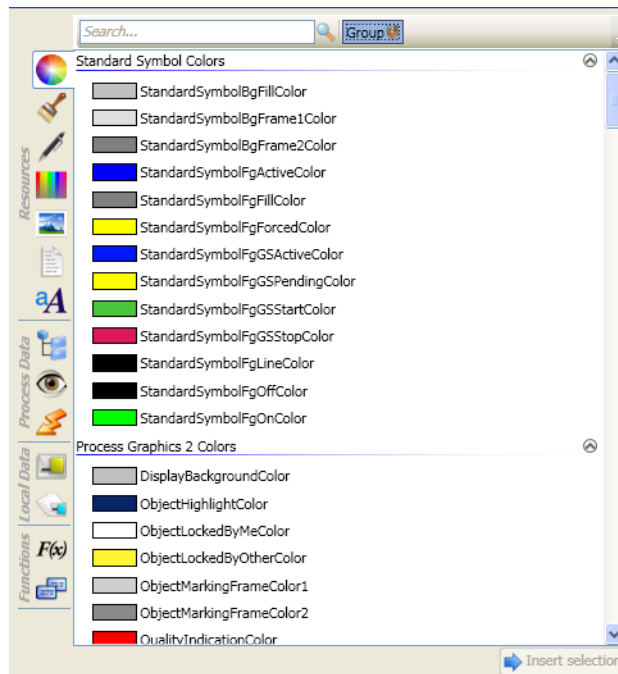
Figure 33. Data Selection Area

The following are the different methods to insert a selection from the data selection area into the expression editing area:

- Select a data and click **Insert selection**.

- Double-click on a selected data.

- Select a data and click ![arrow] corresponding to the selection.

Press F4 to display a browser that is consistent with the data type of the property in the data selection area.

For example, if the property being edited is of data type *Brush*, pressing F4 in the data selection area displays the Logical Brushes tab.

Resources

These browsers help the user to select images, fonts, colors, or NLS texts for the property.

Click **Group** to group or ungroup the resources into different resource groups.

Table 4. Resources

Image / Name	Description
Logical Colors	Colors available in 800xA system are displayed. The logical colors are configured in the workplace. Select the required color category and the list of colors available in the system is displayed.
Logical Brushes	Logical Brushes allow the user to select a brush such as Linear gradient, Radial gradient, Hatch, or Image. This is relevant only for properties having the data type **Brush**. To set the brushes, refer to Linear gradient, Radial gradient, Hatch, Image.
Pen Editor	Pen Editor allows the user to select line styles and line thickness. Brush, dash cap, dash style, line join, and thickness can be specified for the pen. This is relevant only for properties having the data type **Pen**. For more information on pens, refer to Pen on page 198.
Constant Colors	Colors available in Microsoft Windows are displayed.
RGB Editor	RGB Editor allows the user to select a customized color by giving values for **R**, **G**, **B**, where **R** is the value for Red, **G** is the value for Green, and **B** is the value for Blue. The values for all factors can vary between 0 and 255. When the value for **A** is 0, the item is transparent, and when it is 255, the item is opaque.
Images	Images allow the user to select an image. For more information on configuring images, refer to Images tab on page 285.

Table 4. Resources (Continued)

Image / Name	Description
Localized Text	Localized Text lists all the NLS resource libraries defined in the system. For more information on configuring localized texts, refer to Config View on page 280.
Logical Font	Logical Font lists all the fonts defined in the system. It also allows the user to define a font by specifying a font family, font size, style, and font weight. For more information on configuring logical fonts, refer to Fonts tab on page 286.

Process Data

These browsers help the user to select aspect properties, views, or verbs to be referenced for the property.

Click **Objects** to control the visibility of structures and objects.

Click **Aspects** to group or ungroup the aspects into different categories.

Table 5. Process Data

Name	Description
Properties	This allows the user to select a property of an aspect object.
Views	This allows the user to select a view (for example, Config View) of an aspect object. If the aspects on one object have the same name, the aspect category for each aspect is also displayed. This helps the user to distinguish between the PG (Process Graphics) and VBPG graphic aspects.
Verbs	This allows the user to select a verb of an aspect object. A verb is an action that can be performed on an aspect or object, accessed through the context menu. For example, Edit verb for a graphics aspect launches the Graphics Builder and Open verb launches the executable associated with a Windows Application aspect.

Local Data

These browsers help the user to select input properties and variables for the expressions.

Select **Display matching types only** to display the input properties and variables corresponding to the data type of selected property.

Table 6. Local Data

Name	Description
Input Properties	This allows the user to select an input property defined for the graphic aspect.
Variables	This allows the user to select an expression variable, local variable, or out terminal for the expression.

Functions

These browsers help the user to select expression functions or enumerated values for the property.

Select **Display matching types only** to display the functions or enumerated values corresponding to the data type of selected property.

Table 7. Functions

Name	Description
Functions	It lists the functions valid for data type of the property being edited. For more information on functions, refer to Expression Functions on page 233.
Enumeration Values	It lists the enumerated values for data type of the property being edited.

Context Information Area

The context information area displays the name of the graphic item and the property of the graphic item for which the expression is created. Name of the expression

variable is displayed if the user is creating or modifying an expression for an
expression variable.

> The user can select another property of the graphic item to create an expression.
> On selecting a property, the expression applied for the property is displayed in
> the expression editing area.
>
> If there are two or more expression variables, the user can select the required
> expression variable to create an expression.
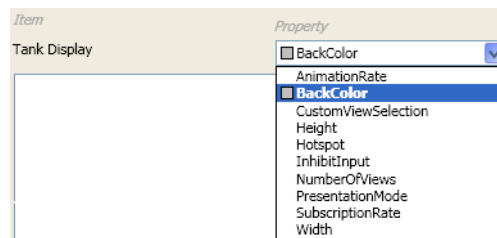
Figure 34. Context Information Area

Help Area

The help area displays examples of expressions that are relevant for the data type of
the selected property.
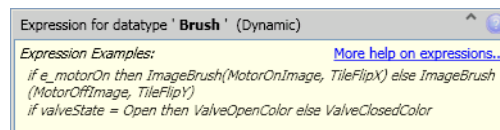
Figure 35. Help Area

Expression Variables

Expression variables are used for storing the results of intermediate expression
calculations. Expression variables store the results of expressions and reduces the

complexity of expressions, by allowing calculations to be performed in intermediate steps. It also enables reuse of frequently used expressions.

Expression variables are also used to store a local state. For example, if an operator clicks a button, some other graphic items become visible in the graphic display. During the button click, a value can be written to an expression variable. The **Visible** property of the other graphic items may have an expression referring to this expression variable.

Execute the following steps to create an expression variable:

1. Select **View > Expression Variables**.
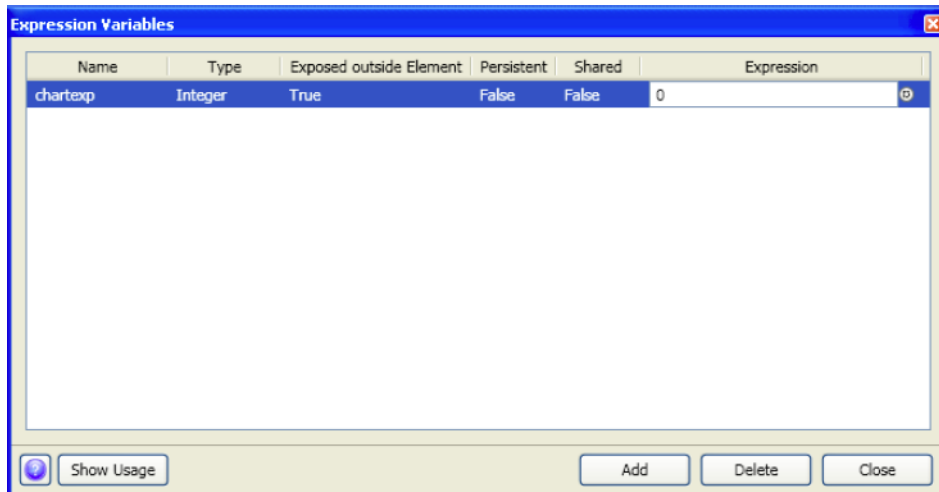
Figure 36. Expression Variables
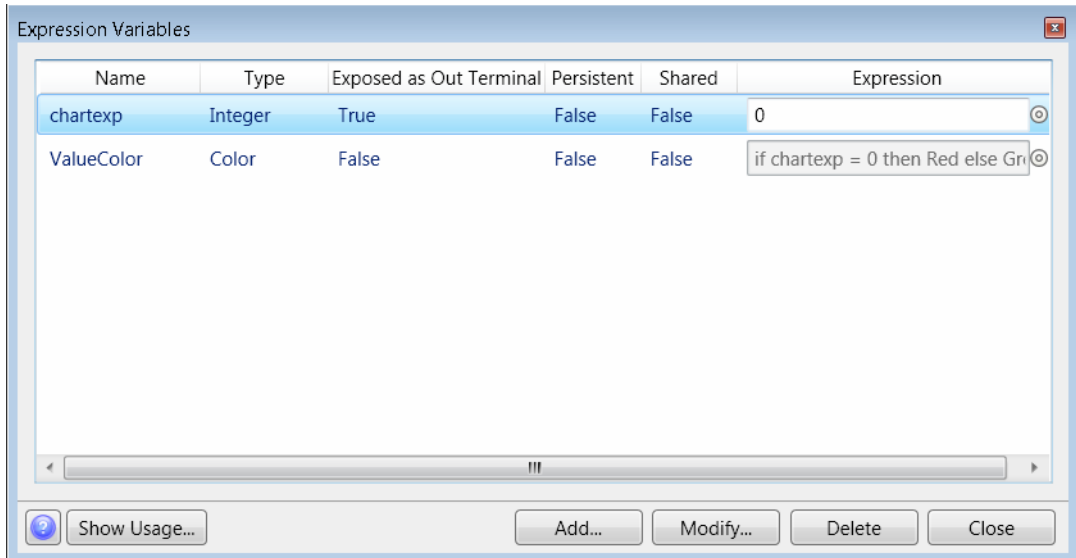
Figure 37. Expression Variables

_____

2.   In the **Expression Variables** window, click **Add** to create an expression variable.
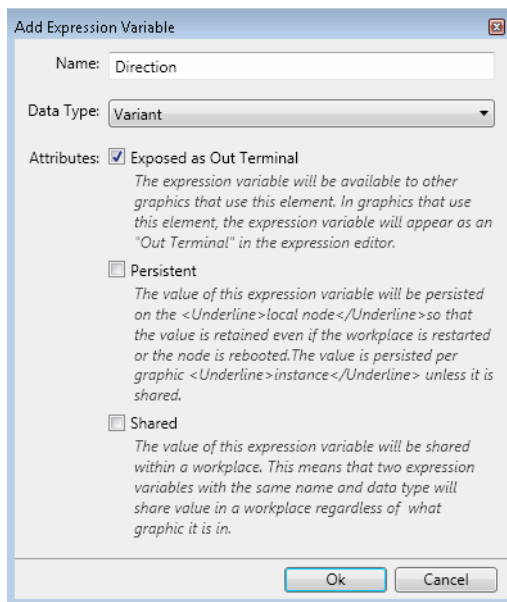
Figure 38. Add Expression Variable

3. Type a **Name** for the variable and select the **Data Type** for the variable. For more information on data types, refer to Data Types on page 190.

4. Select an attribute for the expression variable, as **Exposed as Out Terminal**, **Persistent**, or **Shared** if required. For more information on attributes, refer to Expression Variables on page 131.

5. Click **OK** to return to the **Expression Variables** window.

6. Click ⊚ in **Expression** column to open the Expression Editor for creating expressions. For more information on using the Expression Editor, refer to Expression Editor on page 64.

Select an expression variable from the list and click **Show Usage** to view the list of graphic items that use this expression variable.

To rename an expression variable, select the expression variable from the **Expression Variables** window (Figure 37). Right-click and select **Rename** from the context menu. The user can also use F2 to rename an expression variable.

Feature Pack Functionality _____

Consider the following while modifying or renaming an expression variable:

1. The following warning appears if the attributes **Exposed as Out Terminal**, **Persistent**, or **Shared** are set for the expression variable.
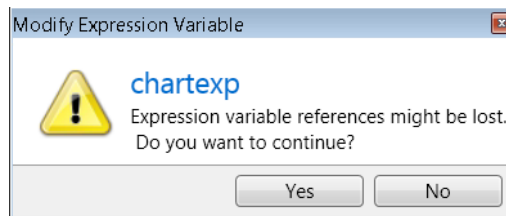
Select **Yes** to continue with modifying or renaming the expression variable.

2. There may be other graphic aspects which refer to expression variables having the attribute **Exposed as Out Terminal** set. If the referenced expression variables are renamed or if the selection of **Exposed as Out Terminal** attribute is cleared, the corresponding graphic aspects have to be manually updated for the references.

Select an expression variable and click **Modify** to modify the expression variable.

_____

Select an expression variable and click **Delete** to delete the variable.

The user cannot delete an expression variable if it is referenced by any property of graphic items or input items.

Usage Window

A graphic aspect includes graphic entities such as input properties, and expression variables.

The Usage Window displays the usage of a specific graphic entity (for example, Input Property) in the graphic aspect. It displays the graphic items and corresponding properties using the graphic entity.

The following graphic entities contain the **Show Usage** function:

• Input Properties

• Expression Variables

• Data References

• Resource References

For example, Figure 39 shows a Usage Window of input properties. Clicking **Show Usage** in **Input Properties (View > Input Properties**), displays the list of graphic items and expression variables using the input property.
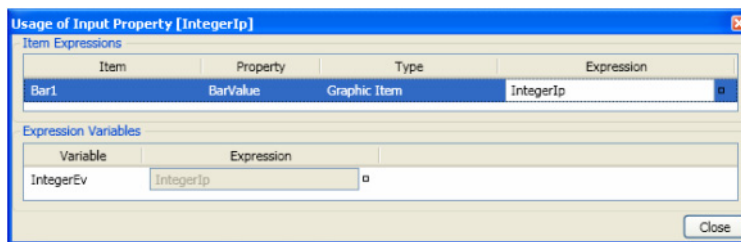
Figure 39. Usage Window

Item Expressions lists the graphic items and input items using the graphic entity. Name of the item, property of the item, type of the item, and the expression assigned to the item property is displayed.

Expression Variables lists the details of expression variables using the graphic entity. The name of the expression variable and its expression is displayed.

The Usage Window also displays the usage of graphic entities if the graphic aspect has multiple views. The items are grouped based on the view.
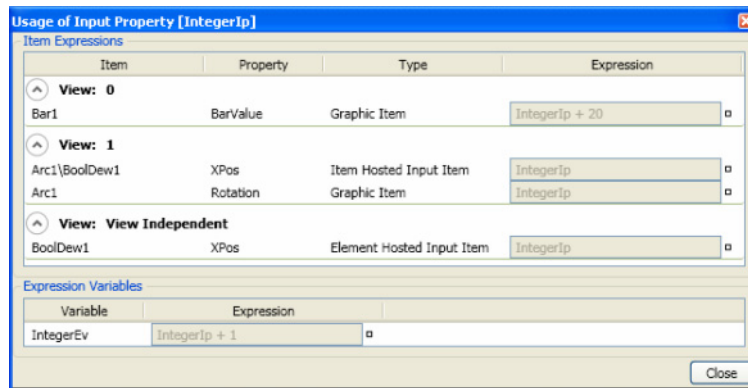
Figure 40. Usage Window with Multiple Views

Selecting an item in **Item Expressions**, selects the item in the edit panel of the respective view and also switches to the view.

The expressions displayed in **Item Expressions** and **Expression Variables** can be directly modified in the **Usage Window**.

Click ⊚ to open the Expression Editor for modifying the expressions. For more information on writing expressions, refer to Expression Editor on page 64.

Input Properties

Input properties are user-defined properties of a graphic aspect. These properties are created within graphic elements or generic elements. The input properties defined on the graphic element are externally visible to any graphic display having the graphic element in it. **Properties** window of the graphic display shows the input properties defined for the element. Values for the input properties can be set through the **Properties** window. Input properties can also be referenced by expressions in the graphic element.

Execute the following steps for creating an input property:

1. Select **View > Input Properties**. The **Input Properties** window appears displaying details of the existing input properties.
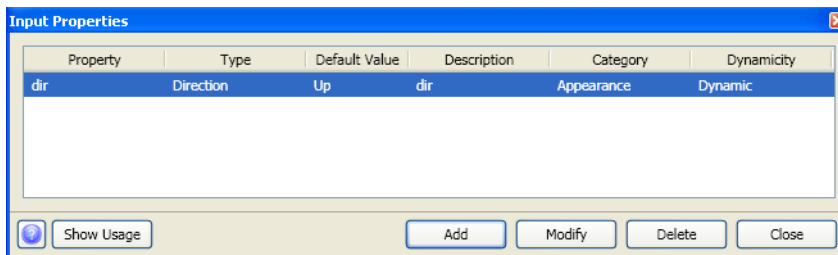
Figure 41. Input Properties

2. Click **Add** and the **Add Input Properties** window appears.
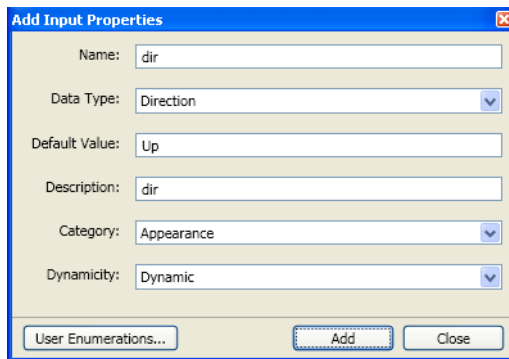
Figure 42. Add Input Properties

3. Type a **Name** for the input property.

4. Select a **Data Type** for the input property. For more information on data types, refer to Data Types on page 190.

5. Type a **Default Value** for the input property.

6. Type a **Description** for the input property. This is displayed as a tooltip or description for the property in **Properties Window**.

    The user can specify a description enclosed in "" (for example, "Vertical line") or a Resource ID (for example, NLS_T12).

7. Select a **Category** to be assigned for the input property.

ℹ️ The user can also create a new category for the input property. Categories are used by the **Properties Window** to organize the properties.

    Select *[New Category]* from the available list and type a name for the category.

8. Select the dynamic behavior of the input property in **Dynamicity**.

ℹ️ **Constant** indicates that the value of input property should be a constant.

    InitOnly indicates that the value of input property can be a constant or an expression referring to input properties that are **Constant** or **InitOnly**. The **InitOnly** values should be calculated during invocation of the graphic aspect.

    Dynamic indicates that the value of input property can be an expression containing references to dynamic data.

    ConstantAtAnchoredLayout indicates that the input property should have a constant value when the **PresentationMode** of the graphic element is *Anchored* or *AnchoredHorizontal* and **AnchorStyle** property of instances of the graphic element are not selected as *Custom*.

9. Click **Close** to save the changes made and close the window.

To add more input properties, continue to enter the details of input property and click **Add.**

The user is allowed to create user defined data types while creating input properties. Click **User Enumerations** to create a user defined data type. For more information on user defined data types, refer to User Enumerations on page 84.

Select an input property and click **Modify** to edit the details of the input property.

Select the input property and click **Delete** to delete the input property. The user cannot delete an input property that is referenced in an expression.

Select an input property and click **Show Usage** to view the list of graphic items that use this input property.

To rename an input property, select the input property from **Input Properties** window (Figure 41). Right-click and select **Rename** from the context menu. The user can also use F2 to rename an input property.

User Enumerations

User enumerations are user defined data types. These are relevant for input properties.

For example, consider a graphic element that represents a fan. If the user requires to control the speed of the fan using a property *FanSpeed*, create a user enumeration with values *Low*, *Medium*, and *High*. Also create an input property having the data type as the newly created user enumeration.

Add this graphic element to a graphic display. The input property of the graphic element appears as a property listed in the **Properties** window. The user can now control the speed of the fan using this property.

Viewing the Enumerations

To view the enumerations, select **View > User Types**. The **User Types** window appears displaying the existing user enumerations.
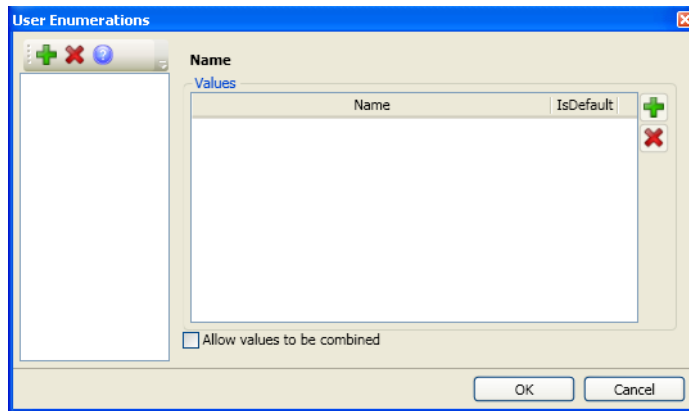
Figure 43. User Enumerations

Creating user enumerations

This section helps the user to create data types. Execute the following steps to create enumerations:

1. Click ![icon] in the toolbar appearing on top of the **User Enumerations** window, to create a new user enumeration. Refer Figure 43.

The user can also right-click the left pane and select **Add** from the context menu to add a new enumeration.
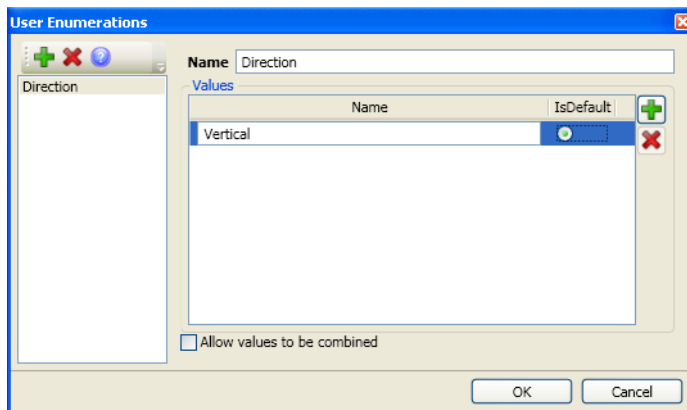
Figure 44. Add User Enumerations

2.   Type a **Name** for the user enumeration.

3.   Type the values for the enumeration in **Values**.

       Click   in **Values** to add new values.

4.   Select **Allow values to be combined** if the user type can have a combination of more than one value.

       For more information on the Enum data type, refer to Enum on page 218.

5.   Select **IsDefault** for a value that should be the default value.

6.   Click **OK** to save the changes.

To delete an enumeration value, select the value and click   in **Values**.

To delete a user enumeration, select a user type and click   on the toolbar appearing on top of the **User Enumerations** window.

To edit a user type or an enumeration value, select the user type or enumeration and type the required value.

Solution Library

A Solution Library supports the reuse of common designs and solutions in the Graphics Builder. It acts as a palette containing solutions that can be copied into the graphic aspect.

A solution includes one or more graphic items. The user can reuse these solutions any number of times for various graphic aspects.

> The solutions in a solution library should not have references to expression variables, input properties, and element hosted input items.

For example, if the user requires to reuse some solutions created in a graphic display, these solutions can be copied into a solution library. On opening the Graphics Builder, the user can view all solution libraries previously created. The required solutions can then be copied into the display.

Select **View > Solution Library** to open the Solution Libraries window.
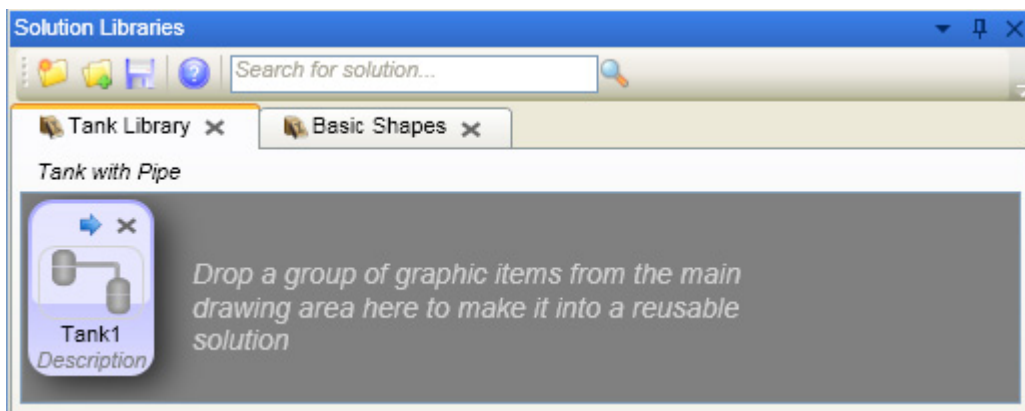
Figure 45. Solution Library

The following are some of the features of a solution library:

• Each solution library is shown in different tabs.

• Solution library can be created or deleted from the Graphics Builder.

- Any number of solution libraries can be opened at the same time.

> To specify a name for the solution library, click the tab header area to type a name.

Adding a solution to a solution library

This section helps the user to add solutions to a solution library. Solutions are added from the edit panel. There are three ways to add solutions into a solution library.

1. Copy the graphic item to be added into the solution library. Right-click the solution library and select **Paste** from the context menu.

2. Drag and drop the graphic item from the edit panel into the solution library.

3. Select the graphic item to be added into the solution library. Right-click and select **Save as Solution** from the context menu.

Adding a solution to the edit panel

This section helps the user to add solutions from a solution library into the edit panel. There are four ways to add solutions into the edit panel.

1. Drag and drop the solution from the solution library into the edit panel.

2. Select the solution to be copied. Right-click and select **Copy** from the context menu. Right-click the edit panel and select **Paste** from the context menu.

3. Right-click the solution to be copied and select **Insert** from the context menu.

4. Click ⇨ on the solution.

Hotspots in a solution

> This section applies to the System 800xA 5.1 Feature Pack release only.

Hotspots help in positioning a solution in a Graphic Aspect. When a solution is added to the edit panel, the hotspot is placed in the position where the user clicks.

By default, the hotspot is located in the middle of the solution. The hotspot appears as a small red circle (see Figure 46).
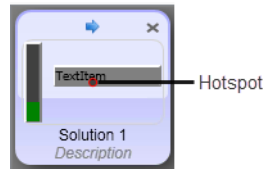
Figure 46. Hotspot in a solution

To change the hotspot, right-click the solution and select **Set Hotspot** from the context menu. The **HotspotDialog** dialog appears.
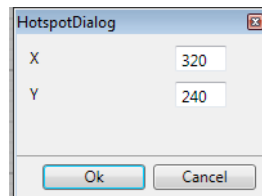
Figure 47. Changing the Hotspot

Set the X and Y coordinates for the hotspot and click **Ok**. The appearance of the hotspot changes in the solution.

Context Menu and Toolbar

Right-click on the solution library tab to access the following context menu.

Figure 48. Context Menu of Solution Library

Table 8. Context menu of a Solution Library

Item	Description
New	To create a new solution library.
Open	To open an existing solution library.
Save	To save the currently opened solution library.
Rename	To rename the solution library.
Close	To close the solution library.
Delete	To delete the solution library. This displays a message to the user to confirm this operation. Click **Yes** to delete the solution. Otherwise click **No**.

Right-click on a solution in the solution library to get the following context menu.

Figure 49. Context Menu of Solution

Table 9. Context menu of a Solution

Item	Description	Toolbar Icon
Insert	To insert the solution into the edit panel.	
Rename	To rename the solution.	
Change Description	To change description of the solution.	
Set Hotspot*	To change the hotspot of the solution.	
Copy	To copy the solution to clipboard.	
Paste	To paste the solution to the solution library.	
Delete	To delete the solution. This displays a message to the user to confirm this operation. Click **Yes** to delete the solution. Otherwise click **No**.	

*   Feature Pack Functionality

The following is the toolbar of a solution library.

Figure 50. Toolbar of Solution Library

Table 10. Tool Bar

Item	Description	Toolbar Icon
New	To create a new solution library.	
Open	To open an existing solution library.	
Save	To save the currently opened solution library.	

To search for a solution library, type the name of solution library in the toolbar and click .

Opening a solution library

Clicking   allows the user to select a solution library to be opened. The **Open Solution Library** window appears.
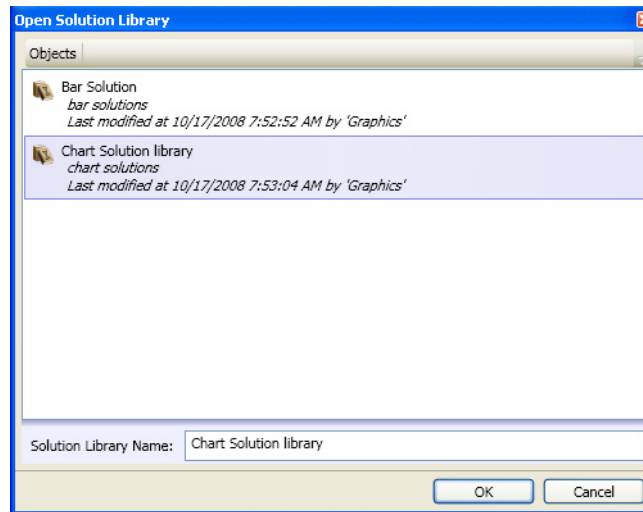
Figure 51. Opening a Solution Library

Select a solution library from the list and click **OK** to open the solution library.

The name of the selected solution library is displayed in **Solution Library Name**.

Click **Objects** to view the tree structure displaying the structures and objects in the system. This allows the user to browse for a solution library in the object tree.

Saving a solution library

Clicking ⊡ allows the user to save a solution library. The **Save Solution Library** window appears.

This window appears only while saving the solution library for the first time.
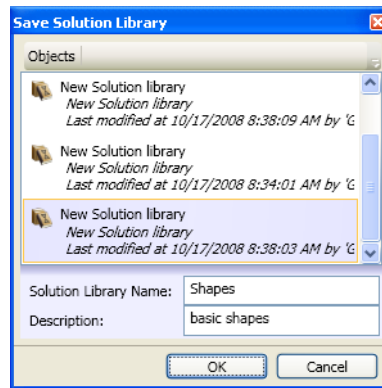
Figure 52. Saving a Solution Library

Type a name to save the solution library in **Solution Library Name**.

Type a **Description** for the solution library if any.

Data References and Resource References

Data References and **Resource References** in the Graphics Builder are used to list the references made from a graphic aspect. It displays states of the references (whether resolved or broken). For more information on different status of references such as Resolved and Broken, refer to Reference Status on page 143.

Data References and **Resource References** also allow the user to replace references made in the graphics aspect. For example, a color is referenced from many graphic items in the aspect. If the user requires to use a different color, this reference can be replaced using the **Resource References** window. All expressions where the color is referenced will be updated.

Select **View > Data References** and the **Data References** window appears. This displays the data references in the graphic aspect, which includes references to graphic elements, aspect object properties, aspect views, or aspect verbs.
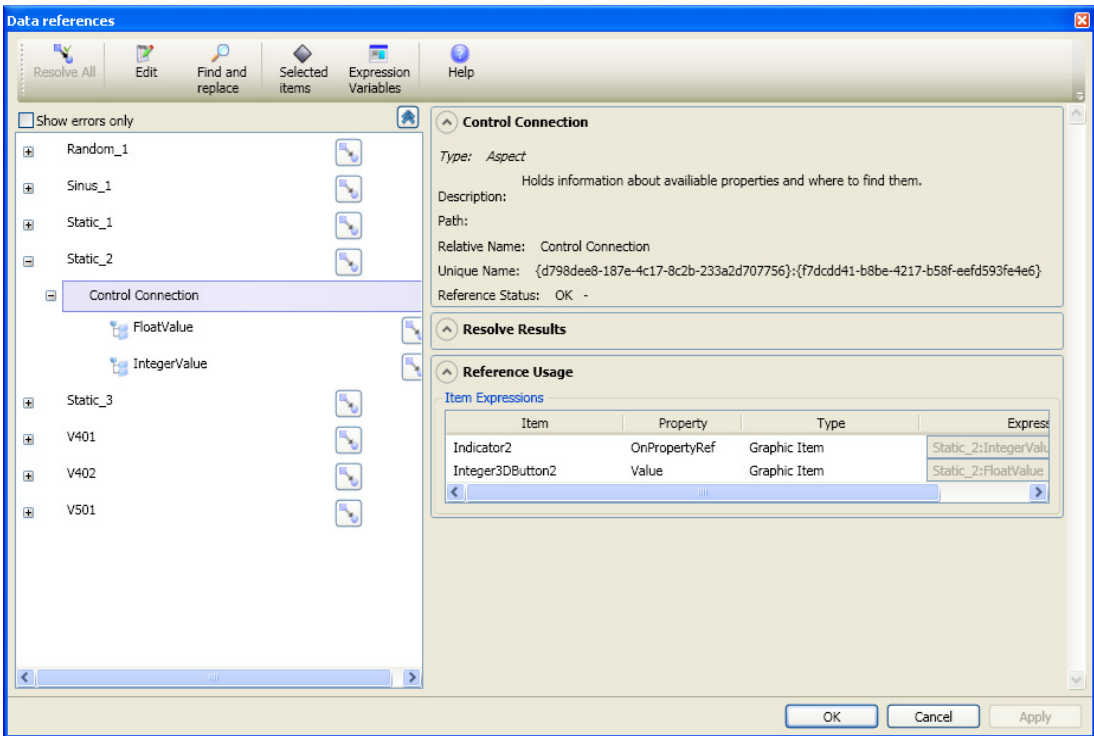
Figure 53. Data References

Select **View > Resource References** and the **Resource References** window appears. This displays the resource references in the graphic aspect, which includes references to colors, brushes, images, fonts, or localized texts.
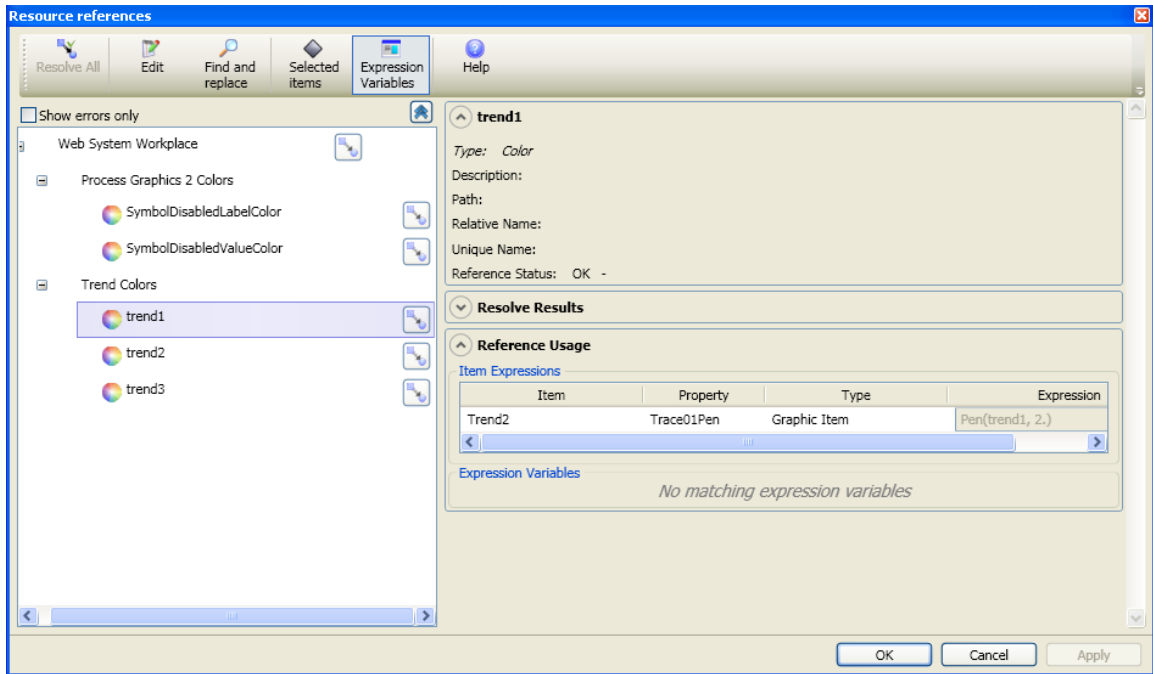
Figure 54. Resource References

The references are displayed in the following levels:

- Object
- Aspect
- Property, Verb, View, or Resource

The following are the details displayed in the **Data References** or **Resource References** window:

- *Tree View* shows the data or resource references made in the graphic aspect in the form of a tree structure. Expand the tree to view the different levels of reference.

  Objects are displayed as root nodes, with aspects and references as child nodes.

- ☒ appears for a bad reference, in the tree view.

- ⚠ appears for an object containing child nodes with bad references.

- Select a reference at any level to display the details of the reference. This appears on the right of the window. The following information is displayed:

  – Name of the selected reference.

  – **Type** of the selection, such as object, property, or color.

  – **Description** for the selection.

  – **Path** where the object is located.

  – **Relative Name** of the object if any.

  – **Unique Name** of the object. This contains the object identifier (GUID).

  – **Reference Status** of the reference. This displays *OK* or a description specifying the error in reference.

  For example, reference status can be *OK*, *Broken*, *Unresolved*, or *SubstatusBad*.

  – *OK* specifies that all references and sub-references are resolved.

  – *Broken* indicates a broken reference.

  – *Unresolved* specifies that the reference has never been resolved to an existing object in the system.

  – *SubstatusBad* specifies that the reference is resolved but there are broken or unresolved sub-references.

- **Resolve Results** shows the result of the resolve operation. It displays whether the resolve operation was successful or a failure. If a reference can be resolved to different items, the list of items will be displayed.

- **Reference Usage** displays the expressions where reference is used. For more information on the usage window, refer to Usage Window on page 79.

Click **Apply** to apply the changes and then click **OK** to close the window.

Click **Cancel** to cancel the changes and close the window.

To correct references in the graphics aspect that is copied from one object type to another, execute the following steps:

1. Select the object in the *Tree View* and click ![icon]. The object browser appears.

2. Select the object type to which the graphics aspect is copied to and click **OK**.

    All references for the selected object will be resolved.

Toolbar

Figure 55 shows the toolbar of the reference window.
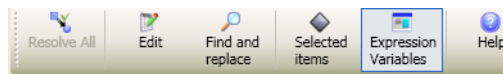
Figure 55. Toolbar of Data References and Resource References

Click **Resolve All** to resolve all the broken references.

> ℹ If **Resolve All** fails to resolve an item, use **Resolve** corresponding to a reference, to resolve individual references. A list of references will be displayed if the reference is ambiguous. The user can select the required reference from the list.

Click **Edit** to set the reference tree to the edit mode. This allows the user to modify object names or references.
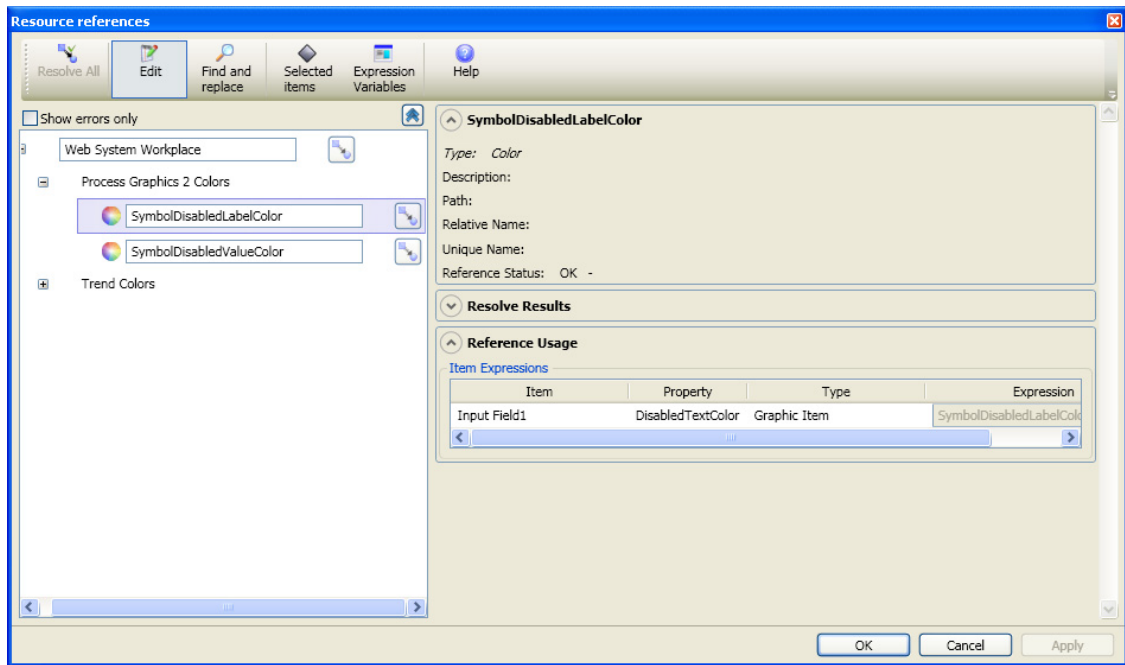
Figure 56. Edit Mode Data References and Resource References

> If the specified reference or object does not exist in the system, the edit box is displayed with a red frame. The user can check the resolve result in case of a red frame.

Click **Selected Items** to display the references from graphic items that are selected in the Graphics Builder. This is relevant if user requires to replace references for a set of graphic items (Otherwise when the user changes a reference, all graphic items will be affected).

Consider the following example:

1. Select the graphic items that should be duplicated.

2. Press Ctrl + C (To copy) and then Ctrl + V (to paste) to duplicate the graphic items.

The duplicated items will be the selected graphic items in the edit panel.

3. Open the **Reference Window** and click **Selected Items**.

    Only the selected graphic items are displayed in the usage window. Changing references will affect only the selected graphic items. Change the references and click **Apply** to update the references for the selected items.

Click **Expression Variables** to view the references made from expression variables in the graphic aspect.

If an expression variable and a graphic item is referencing the same item (for example, an Aspect Object Property) and the user requires to update only the reference made from the expression variable, click **Expression Variables** and click **Selected Items** (with no graphic items selected in the Graphics Builder). Only the references made from expression variables will be shown and only expression variables will be updated even if the same reference is used in a graphic item.

If an expression variable and a graphic item is referencing the same item (for example, an Aspect Object Property) and the user requires to update only the reference made from the graphic item, see that the **Expression Variables** button is not clicked. Only graphic items that use the references will be updated and not the expression variables.

Click **Find and Replace** to search for a specific object or reference. It also allows the user to replace a searched object name or reference with another name. If the specified reference or object does not exist in the system, the edit box is displayed with a red frame.
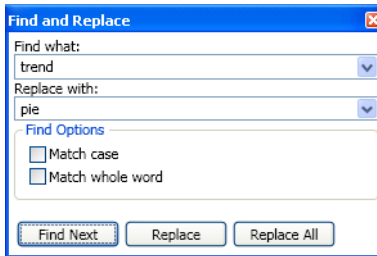
Figure 57. Find and Replace References

–   Type the name to be searched for in **Find what**.

–   To replace the existing name, type the replacement name in **Replace with**.

–   Select **Match case** if the search should be case-sensitive.

–   Select **Match whole word** if the search should exactly match the name to be searched.

–   Click **Find Next** to find the next occurrence of the name.

–   Click **Replace** to replace the name.

–   Click **Replace All** to replace all occurrences of the name.

Context Menu

Right-click on a reference in the *Tree View* to display the context menu of a reference as shown in Figure 58.

Figure 58. Context Menu of Data References and Resource References

- Select **Resolve** to resolve the reference. This is enabled only if the selected reference is a broken or unresolved reference.

- Select **Change** or click 🔧 corresponding to a reference, to change an object name or a reference.

  The appropriate browser is launched depending on the reference type.

  Following are the browsers used for data-entity references:

  – Element

  – Property

  – View

  – Verb

  Following are the browsers used for resource references:

  – Color

  – Brush

  – Localized text

  – Font

  – Image

  In the browser, select the desired item and click **OK**.

- Select **Find and replace** to search for a specific object name or a reference. It also allows the user to replace the object name or reference name. For more information refer to Toolbar on page 98.

- Select **Expand tree** to expand the selected node in the *Tree View*.

Test Data

A graphic aspect can have references to values (for example, an aspect object item property) in the system. The references control the behavior and appearance of the aspect during execution of the aspect in a live environment.

The Test Data dialog is used to test the behavior and appearance of a graphic aspect by assigning values to the properties used by the graphic aspect.

The Test Data dialog displays the following:

- Name of the subscribed aspect object properties.

- Name of expression variables or input properties.

- Data type of each property.

- Value of each property.

- Data quality for aspect object properties.

Select **View > Test Data** and the **Test Data** window appears.
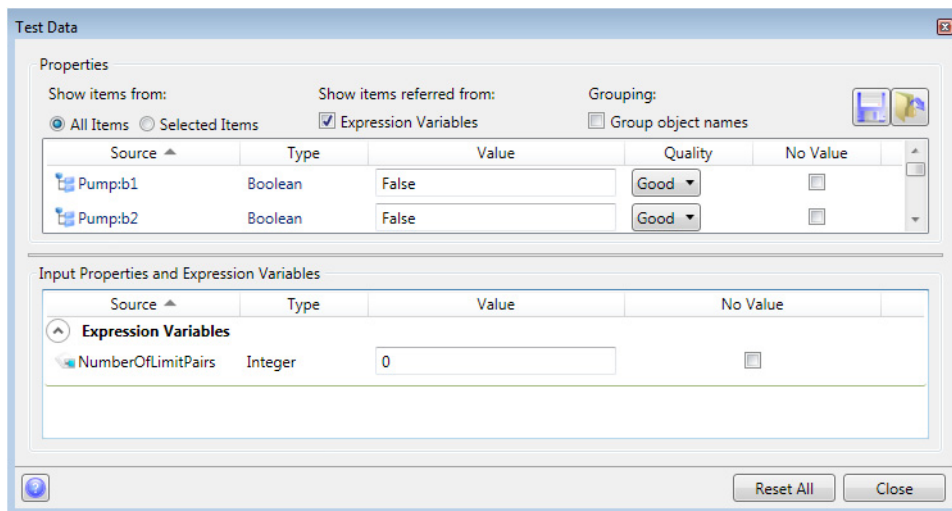
Figure 59. Test Data

Properties displays the properties used by graphic items properties in the aspect.

Show Items helps the user to limit the scope of items to be displayed in this window.

- Select **All** to display the input properties and subscribed properties of all the items in the graphic aspect.

- Select **Selected Items** to display the input properties and subscribed properties of selected items in the graphic aspect.

–   Select **Expression Variables** to display subscribed properties that are referenced by expression variables.

–   Select **Group Object Names** to group the subscribed properties based on the respective object names.

–   Click ![icon] to save the property list to a specific location. The list is saved as a *.xml* file.

–   Click ![icon] to open a property list from any location.

Input Properties and Expression Variables displays the details of all input properties and expression variables assigned for graphic items.

Table 11 describes the columns available in **Test Data** window.

Table 11. Test Data columns

Name	Description
Source	Source name of the subscribed property, name of the input property, or name of the expression variable.
Type	Data type of the subscribed property, input property, or expression variable.
Value	Current value of the subscribed property, input property, or expression variable, expressed as a constant.
	If the data type of an Input Property is **Enum**, a drop-down containing the list of enumerated values will appear. The user can select the required value.

Table 11. Test Data columns

Name	Description
Quality	Data quality of the subscribed property. This can be Good, Uncertain, Bad, or Bad (Uninitialized).
No Value	This check box enables the user to test the behavior of a property reference, input property, or an expression variable when the result is a "No Value". If a property reference is not available, the **No Value** check box is automatically selected in the **Test Data** window. The user cannot specify any value in **Value** field. The user may clear the selection of this check box and then specify any value in the **Value** field. This check box is not enabled in the **Live** mode of Graphics Builder.

In the **Live** mode of Graphics Builder, *Value* is not editable, and the current data quality of the property is displayed in *Quality*.

In the **Edit** mode of Graphics Builder, *Value* is editable, and the user can select the data quality for the property in *Quality*.

Click **Reset All** to reset the values of all properties and variables.

Show Migration Errors

A VBPG graphic aspect can be migrated to a PG graphic aspect using the Graphics Migration Tool. The corresponding PG graphic aspect is created after the migration even if there are any errors occurring during the migration process.

Select **File > Show Migration Errors** to view the errors that occurred during the migration process. This is applicable only for a migrated PG graphic aspect.
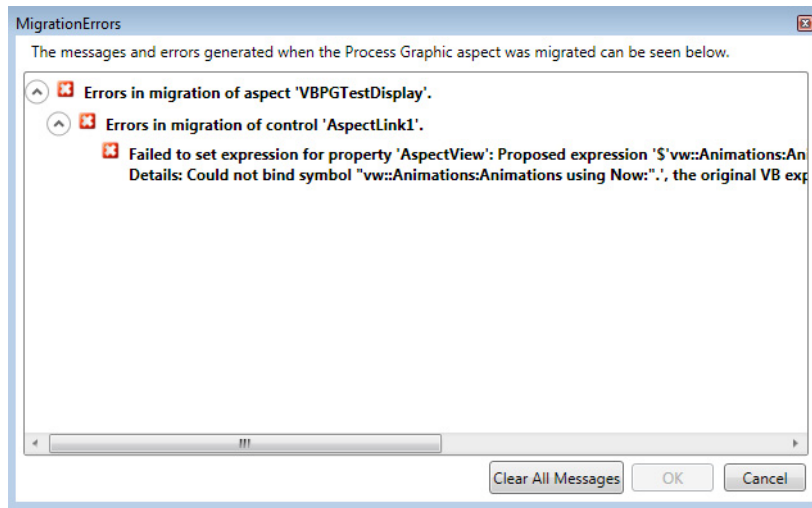
Figure 60. Show Migration Errors

Click **Clear All Messages** to clear all the error messages.

If the graphic aspect is saved without clearing the error messages, the aspect cannot be approved and this is prompted to the user. If the user continues to save the aspect, the aspect is saved and will be in *Aspect unapproved* state.

For more information on migrating graphic aspects, refer to *System 800xA, Engineering, Process Graphics Migration Tool (3BSE049231*).*

Graphics Builder Settings

The Graphics Builder allows the user to configure the settings of Graphics Builder. The settings done for the Graphics Builder will be used when the user again opens the Graphics Builder. This includes the grid settings, toolbar icons, and the background color. The user can select the toolboxes that should appear in the **Toolboxes** window of Graphics Builder.

Select **Tools > Options** to set the display settings. The **Options** window has two tabs, an **Editor** and **Toolbox Order**.
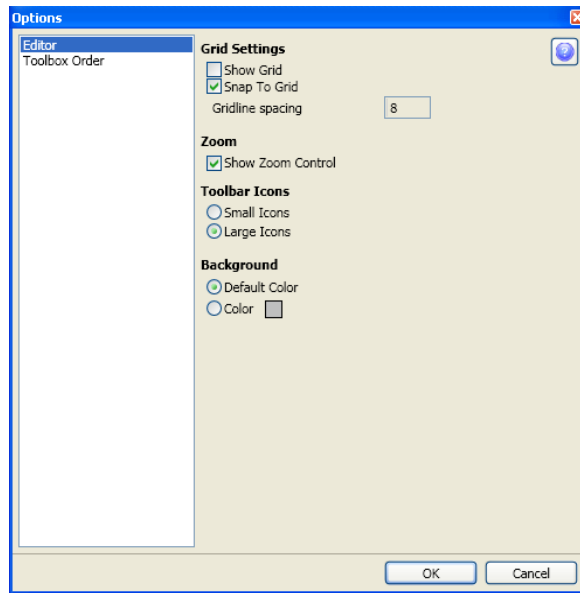
Figure 61. Graphics Builder Settings

Editor

The **Editor** includes the grid settings, background color for the workspace, and the icon settings, as shown in Figure 61. The visibility of the zoom control can also be configured using the Editor.

Editor includes:

- **Grid Settings**

    Select **Show Grid** to display grid in the edit panel.

    Select **Snap To Grid** to enable the snapping of graphic items to the grid.

    Show Grid can also be selected from **Format > Grid**.

    Type a value for **GridLine spacing**. This specifies the distance between grid lines (in pixels).

- **Zoom**

    Select **Show Zoom Control** to view the zoom control toolbar in the edit panel. For more information on zoom control, refer to Zoom In and Zoom Out on page 51.

- **Toolbar Icons**

    Select **Small Icons** to display small icons for items in the main toolbar.

    Select **Large Icons** to display large icons for items in the main toolbar.

- **Background**

    Select **Default Color** to use the default color for the edit panel.

    Select **Color** to select a color for the edit panel using the RGB editor. Specify values for **R**, **G**, **B**, where **R** is the value for Red, **G** is the value for Green, and **B** is the value for Blue. The values for all factors can vary between 0 and 255. When the value for **A** is 0, the item is transparent, and when it is 255, the item is opaque.

Toolbox Order

The **Toolbox Order** displays the list of all the toolboxes. The user can select the toolboxes that are to be displayed in the **Toolboxes** window of the Graphics Builder. This also allows the user to specify the order in which the toolboxes should be displayed in the **Toolboxes** window.
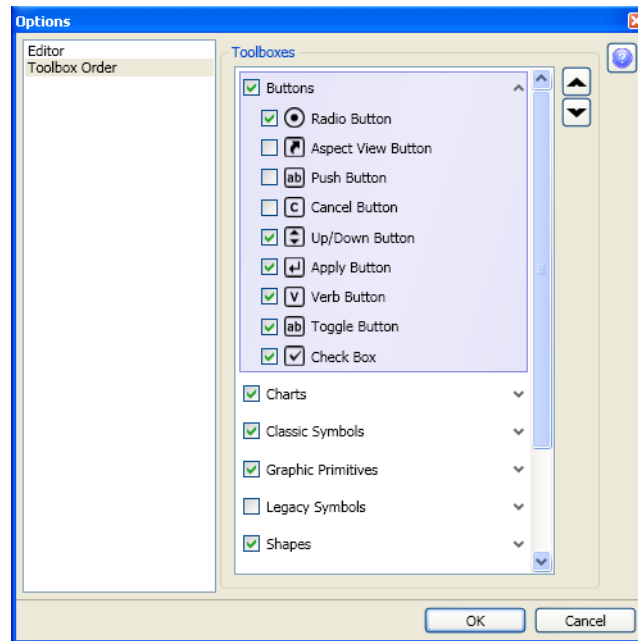
Figure 62. Toolbox

Select the toolboxes that should appear in the **Toolboxes** window. This selects all the items belonging to the selected toolbox, by default.

Click ⌄ corresponding to a toolbox to view the list of toolbox items. The user can also select the items that should appear in the toolbox group.

Click **OK** to save the changes.

Use ⬆ and ⬇ to order the toolboxes.

Display Documentation

The Display Documentation tool captures the information of graphic displays in an Excel spreadsheet. By default, this tool shows a list of **Graphic Display PG2** aspects belonging to a selected object and child objects.

This tool also captures the information of **Graphic Element PG2** and **Generic Element PG2** aspects.

This tool locates and extracts data from desired graphic display aspects within the system structure. The following details of the graphic display aspect are presented:

- Object to which the graphic display aspect belongs to.

- Name of the graphic display aspect.

- Data references.

- Resource references.

- Property names, and values.

- Aspect screen dumps.

There are two ways to invoke this tool.

1. In the Graphics Builder, select **Tools > Display Documentation Tool**.

2. Select **Display Documentation** from **Start > Programs > ABB Industrial IT 800xA > Engineering > Utilities**.
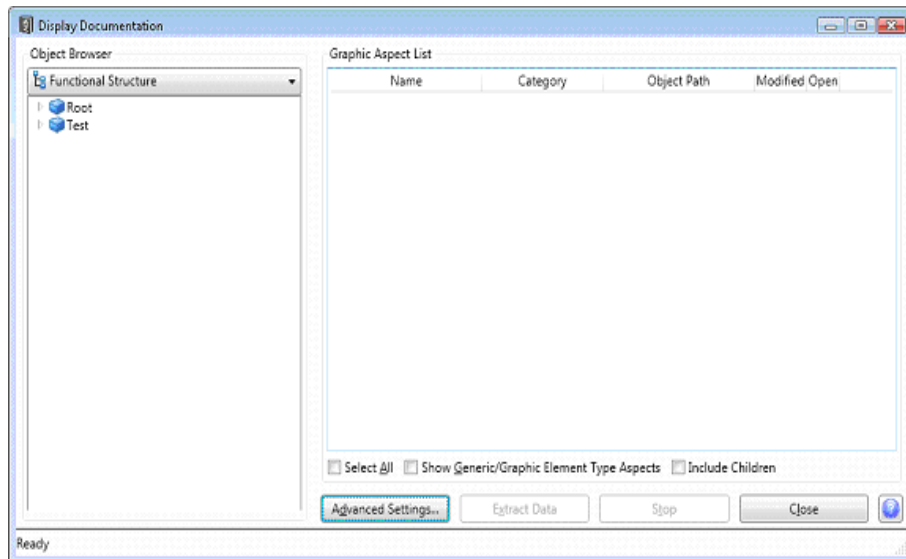
Figure 63. Display Documentation

Select a system structure to display the objects. The objects appear in the form of a tree structure.

Browse through the structure to select the required object. All graphic aspects and the child aspects belonging to the select object appears in **Graphic Aspect List**.

Select **Include Children** to include the child aspects in the **Graphic Aspect List**.

Select **Show Generic/Graphic Element Type Aspects** to display generic elements and graphic elements belonging to the selected object.

Click **Extract Data** to extract data for a selected graphic aspect.

Click **Stop** to stop extracting the data into the excel file.

Select **Select All** to select all the aspects appearing in the aspect list.

Extracting Data for a Graphic Aspect

The Display Documentation tool allows the user to extract data for a selected aspect. The data for a graphic aspect is extracted into an excel file.

Close all the excel files before extracting data.

The excel file contains two main tabs, **Data** and **Screenshot**, as shown in Figure 64.

• **Data** tab displaying the information of the graphic aspect.

• **Screenshot** tab displaying the snapshot of the graphic aspect.

Each primitive displayed in the **Data** tab contains a unique ID referred as index. This index is mapped to the corresponding primitive in the **Screenshot** tab.

Figure 64. Excel file containing Extracted Data of a Graphic Aspect
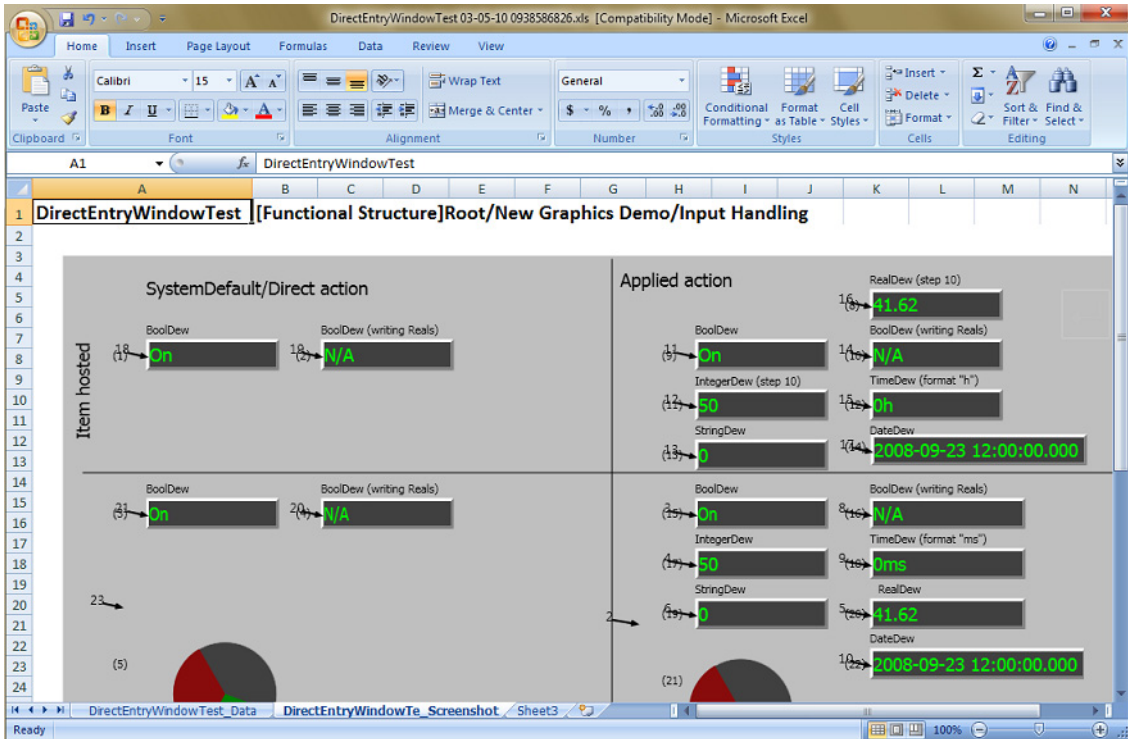
Figure 65. Excel file containing Snapshot of a Graphic Aspects

The following are the ways to specify the path for storing the excel file:

1. Type the path in **Folder Path** in the **Advanced Settings** dialog as shown in Figure 66.

2. To browse for a required folder, click **...** and the **Browse for Folder** window appears. Select the folder and click **OK**. The path appears in **Folder Path**.

Click **Stop** to stop the process of extracting data.

To open the excel file containing the extracted data of graphic aspects, click **...** in **Open** column of **Graphic Aspect List**.

The user is allowed to configure the data to be displayed in the excel file. Click **Advanced Settings** to do the configuration.
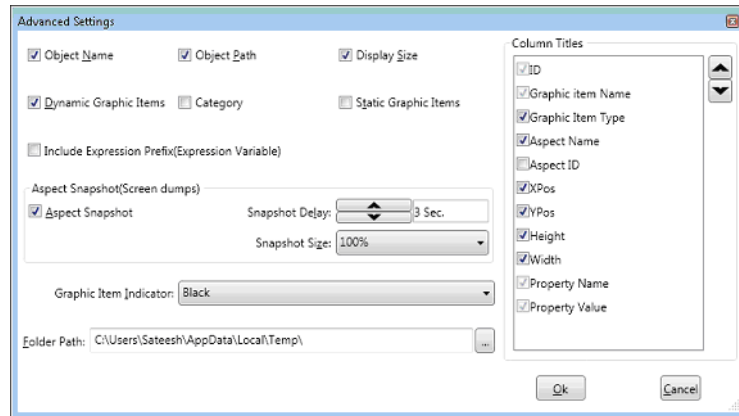
Figure 66. Advanced Settings

Following are the configurations to be done in Advanced Settings:

1. Select **Object Name** to display the name of the object to which the aspect belongs to.

2. Select **Object Path** to display the location / path of the object.

3. Select **Display Size** to display the height and width of the graphic aspect.

4. Select **Dynamic Graphic Items** to list the graphic elements and generic elements that have references to properties.

5. Select **Category** to display the aspect category.

6. Select **Static Graphic Items** to list the graphic items that do not have property references.

7. Select **Aspect Snapshot** to display a snapshot of the graphic aspect in the excel file.

8. In **Snapshot Delay**, specify the time delay (in seconds) for capturing the aspect snapshot. Use ⬆ and ⬇ to increase or decrease the time. By default the time delay is 3 seconds.

9. In **Snapshot Size**, specify the size (in percentage) to capture the snapshot in the excel file.

10. Each graphic item displayed in the **Data** tab is tagged to the appropriate graphic item in the **Screenshot** tab. **Graphic Item Indicator** specifies the color in which the items are marked in the snapshot.

11. Select **Capture Referenced Graphic/Generic Elements Data** to extract the data of all graphic elements added to the graphic display. The data of each graphic element is captured in separate tabs in the excel file.

There will be different **Data** tabs for each graphic element in the excel file.

12. The columns to be displayed in the excel file is selected in **Column Titles**. Use ⬆ and ⬇ to specify the order to display the columns.

The user is allowed to rename the columns appearing in **Column Titles**. Select a column and press F2 to type a new column name.

13. Click **OK** to save the settings done.

Reference Documentation

The Reference Documentation tool generates reference documents containing information of generic elements and graphic elements. This document contains the following information:

• Description about the graphic element, which is provided by the user of this tool.

• Input properties and the details of the properties of graphic element such as the name.

• All user enumeration types defined within the graphic element and the values.

This tool shows a list of **Graphic Element PG2** aspects belonging to a specific system structure and object. This list includes graphic elements, generic elements, and faceplate elements.

To invoke this tool in the Graphics Builder, select **Tools > Reference Documentation Tool**.
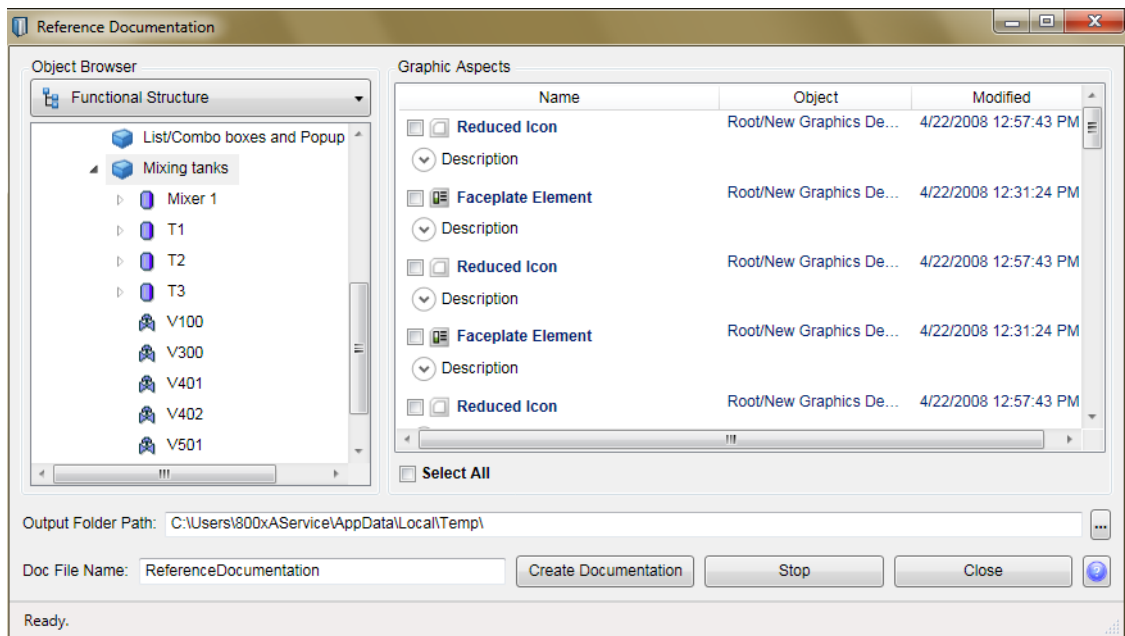
Figure 67. Reference Documentation

In **Object Browser**, select a system structure to display the objects. The objects appear in the form of a tree structure.

Browse through the structure to select the required object. All graphic elements, generic elements, or faceplate elements, belonging to the select object appears in **Graphic Aspects**.

Select the **Select All** check box to select all the aspects appearing in the aspect list.

The Reference Documentation tool allows the user to extract data for a selected element into a word document. The following are the ways to specify the path for storing the word document:

1. Type the path in **Output Folder Path**.

2. Click **...** to browse for the required folder. In the **Browse for Folder** window, select the folder and click **OK**. The path appears in **Output Folder Path**.

Type the document name in **Doc File Name**.

Click **Create Documentation** to start capturing the data for the selected graphic aspect.

Click **Stop** to stop capturing the data into the word document.
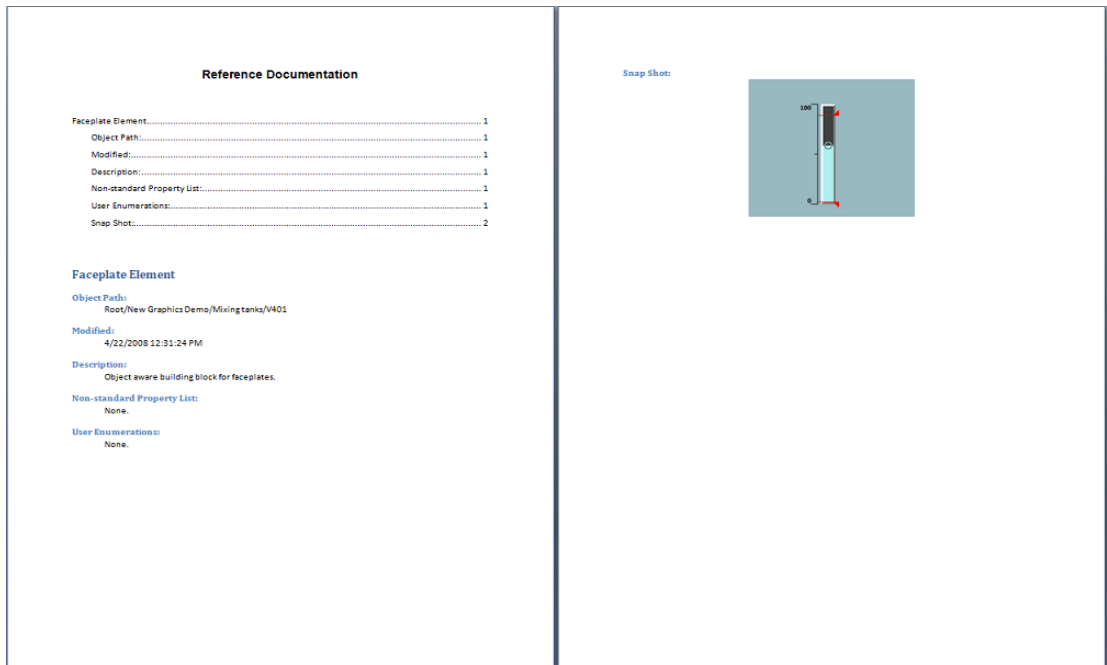
Figure 68. Word Document containing the Extracted Data

Following details are captured in the word document:

- Name of the graphic element.

- Path of the object to which the element belongs to.

- Property list of the element.

- User enumerations defined for the element, and the enumerated values.

- Snapshot of the element.

Section 3 Graphic Aspects

The Graphic Aspects section describes the aspect types implemented by Process Graphics 2 Aspect System. The following are the different aspect types:

- Graphic Display PG2

- Graphic Element PG2

- Generic Element PG2

- Solution Library PG2

For more information on aspect types, refer to Aspect Types in Process Graphics on page 122.

This section also describes the structure of graphic aspects, the concepts of object aware and generic graphic elements, the references and input handling for graphic aspects.

A graphic aspect includes graphic items and input items. Graphic aspects also contain input properties, and expression variables. Appearance and behavior of graphic aspects are defined by:

1. Adding graphic items and input items to the aspect.

2. Changing property values for graphic items and input items.

Figure 71 shows the structure of a graphic aspect. For more information, refer to Structure of a Graphic Aspect on page 125.
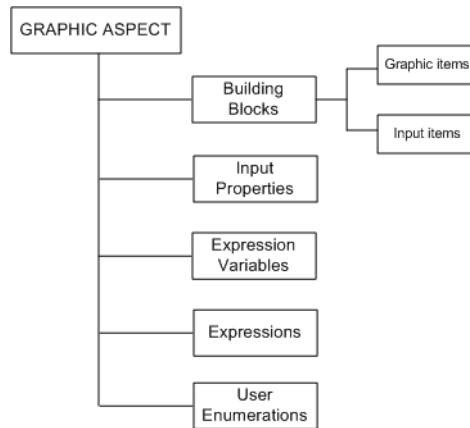
Figure 69. Structure of a Graphic Aspect

Aspect Types in Process Graphics

The Process Graphics 2 aspect system provides four different aspect types.

- Graphic Display PG2, which is the graphic aspect used directly by process operators.

- Graphic Element PG2, which is an object aware building block.

- Generic Element PG2, which is a generic building block.

- Solution Library PG2, which is a graphic aspect that supports copying or pasting solutions for graphic displays in a Graphics Builder.

ℹ️ A solution library is different from other aspect types. Aspects of **Graphic Display PG2**, **Graphic Element PG2**, and **Generic Element PG2** are referred as graphic aspects. For more information on solution libraries, refer to Solution Library on page 87.
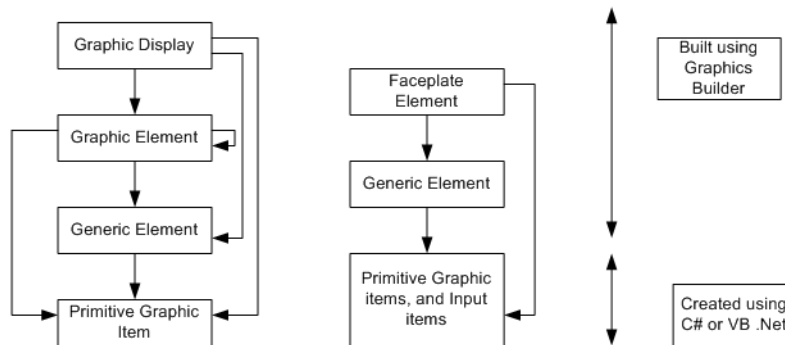
Figure 70. Configuration with layered approach

Figure 70 briefly describes about the components of a Graphic Display, Graphic Element, and Faceplate Element. A Graphic Display can contain primitive graphic items, generic elements, and graphic elements. For more information on the components, refer to Structure of a Graphic Aspect on page 125.

Aspects created from each aspect type are configured using the Graphics Builder.

Graphic Displays are invoked directly by the process operator for monitoring and controlling the graphics during the run time.

Graphic Elements and Generic Elements are building blocks that are used while configuring graphic displays, other graphic elements, or faceplates.

Each aspect type is categorized into different aspect categories that define the usage of a graphic aspect.

Table 12 describes predefined aspect categories.

Table 12. Aspect Categories

Aspect Type	Aspect Category	Description
Graphic Display PG2	Graphic Display PG2	Aspects of this category are used directly by the process operators for monitoring and controlling the graphics during the run-time.
	Navigation Display PG2	Aspects of this category are used to define display menus.
	Object Display PG2	Aspects of this category are used to define detailed information of an object.
Graphic Element PG2	Faceplate Element PG2	Aspects of this category are used in faceplates.
	Graphic Element PG2	Aspects of this category are used as building blocks for other graphic aspects.Graphic elements are building blocks which present a specific object or all objects. A graphic element contains references to properties of the represented object.

Table 12. Aspect Categories (Continued)

Aspect Type	Aspect Category	Description
Generic Element PG2	Generic Element PG2	Aspects of this category are used as building blocks for other graphic aspects.
		These elements are generic because they are not restricted in presenting a specific object or different objects of a particular type. Generic elements may not contain references to aspect object properties or other data entity references.
		A generic element can access object properties through late binding and can have input properties of data type **PropertyRef**.
Solution Library PG2	Solution Library PG2	Each solution library created using the Graphics Builder is an aspect of this category.
		The users configuring a graphic display use solution libraries to copy existing solutions into the display being built.

Structure of a Graphic Aspect

A graphic aspect includes the following components:

- Building Blocks.

- Input Properties.

- Expression Variables.

- Expressions.

- User Enumerations.
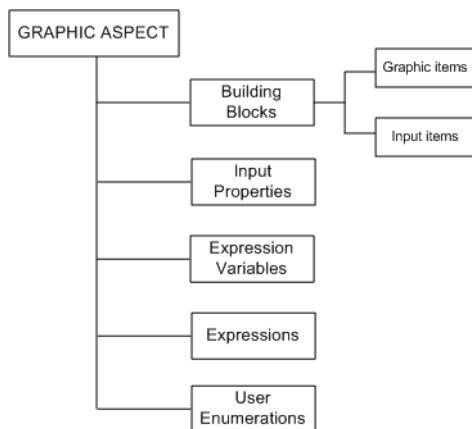
Figure 71. Structure of a Graphic Aspect

Building Blocks

Building blocks are used to build a graphic aspect. Graphic items and input items are referred as Building Blocks. Graphic items include graphic elements and generic elements.

Graphic items define a graphical representation of the graphic aspect.

Building blocks are built-in (that is, defined within Process Graphics) or user-defined.

A built-in building block is a primitive, a graphic aspect, or an input item. Primitives include graphic items such as Rectangle, Pie, Text, and Cone. A user-defined building block is created only using the Graphics Builder.

Built-in primitives cannot be opened in Graphics Builder.

Generic graphic items are created using the toolbox and input items are created using the context menu of the edit panel.

Graphic items and input items possess certain properties like Name, Fill Color, Value. The values of properties are set according to the requirement. The property value can be a constant or a dynamic expression.

Built-in toolboxes contains graphic aspects that can be used as templates for creating user-defined building blocks.

For example, the *Buttons* toolbox contains built-in button implementations. They are added to graphic displays or graphic elements. Buttons work in a direct mode or applied mode and help the operators to interact with the system. A button in an applied mode performs the function only if there is an apply operation. A button in a direct mode performs the function directly when it is pushed. Some of the buttons used in the Process Graphics are Checkbox, Radiobutton, Push button, and UpDown Button.

Toolboxes can also be system defined. This includes toolboxes included in the base installation such as Classic Symbols, Graphic Primitives, Shapes, Special, Standard Symbols, Buttons, and Charts. It also includes the toolboxes corresponding to the loaded system extensions.

For information on creating toolboxes, refer to Creating Toolbox and Generic Elements in Graphics Structure on page 642.

For more information on different building blocks, refer to Appendix A, Standard Building Blocks.

For more information on different input items, refer to Input Items on page 488.

Input Items

Input Items define the input behavior of the graphic aspect. It facilitates implementation of graphic aspect that allows the user to interact with the system.

Input Items are building blocks that are added for a graphic item. This supports triggering of different actions on clicking the item. Input items support the execution of different functions as the result of mouse or other events (for example, a value can be written to an object property when the mouse button is released). Input items are *Element Hosted* or *Item Hosted*.

An *Item Hosted* input items is assigned for a selected graphic item. A graphic item may also be an instance of a graphic element or a generic element. The actions to be

triggered are invoked on clicking the corresponding graphic item. An Item Hosted input item is assigned for a selected item.

An *Element Hosted* input item is assigned for the graphic aspect. Click anywhere in the aspect (except for the areas including the input sensitive items) to invoke the actions to be triggered. For example, click within the background area or on the contained static items to invoke the element hosted input items in a graphic display. Another example is the implementation of buttons. All buttons contain an element hosted input item that defines the click behavior of the respective button. For example, *PushButton* contains an element hosted *PropertyWriterEx*. The *PropertyWriterEx* is the object that performs write operation when the user clicks on a PushButton.

> **i** While copying a graphic item containing name references to *Element hosted* input items from one graphic aspect to another, these input items will not be automatically copied or pasted in the destination element. A separate copy and paste operation is required for the *Element hosted* input items.

Input items include Direct Entry Windows (DEWs) and other input items. This section describes about the input items which provide a click behavior for the implementation of graphic aspects and buttons. For more information on DEWs, refer to Direct Entry Windows on page 504.

More than one input item can be added to a graphic item or aspect. All input items pertaining to a specific event are executed. The execution order is arbitrary.

Assigning several input items to a host. It is possible to assign more than one input item to a host item. Several actions are invoked based on a single event, but often only one of the attached input items has the **Enabled** property set to *True* at the time. For example, a CheckBox might contain an input item that defines the action to be executed while selecting the checkbox and another action to be executed while clearing the checkbox.

Triggering actions. Input items such as Property Writer, Aspect View Invoker, Object ContextMenu Invoker, and Verb Invoker possess a property called **Trigger** which is used for triggering the action. An action is triggered when the value of this property changes from *False* to *True*.

ℹ **Trigger** property is applicable only for element hosted input items.

If a dynamic expression is assigned to the **Trigger** property, other actions are not executed.

Table 13 explains the events that are triggered. These events also indicate the values of the **ActionExEvent** enumeration.

Table 13. Trigger events

Event	Description
OnActivate	Execution takes place for a graphic element shown in the preview area of the Plant Explorer workplace when the preview area is opened.
OnCreate	Execution takes place when the graphic element is invoked.
OnDeactivate	Execution takes place for a graphic aspect in preview area of the Plant Explorer workplace when the preview area is made invisible.
OnDestroy	Execution takes place when the graphic element is closed.
OnMouseUp	Execution takes place while releasing the mouse button.
OnNotTopOfTabs	Execution takes place when the graphic element is on a tabbed faceplate and the tab is on the top and another tab is selected to be on top, that is, the graphic element is hidden.
OnTopOfTabs	Execution takes place when the graphic element is on a tabbed faceplate and the tab is selected as the top tab.
OnDemand	Some graphic items, such as ListView, may request execution of a particular input item by specifying the name of the input item. This property should be set for the input item which is the target for such invocation.

Mouse triggered actions. The input items such as Property Writer, Aspect View Invoker, and Verb Invoker possess properties called **MouseButton** and **ModifierKeys** that determine which mouse button triggers the action and whether combination of modifier keys (SHIFT, CTRL, ALT and Windows) should to be pressed to enable the action.

The **ModifierKeys** property is a set enumeration to support combinations of modifier keys. The following are prerequisites for a mouse triggered action:

• The specified mouse button is clicked with the cursor pointing at the hosting item.

• The specified modifier keys are pressed.

• **Enable** property is *True* when the mouse button is clicked.

• **Enable** property is still *True* when the button is released.

Releasing the mouse button triggers the action.

Input Properties

Input properties are user-defined properties of a graphic aspect.

The semantics of an input property is defined by adding expressions to the graphic aspect. For example, add a text primitive to a graphic element and create an input property named **Font** of data type *Font*. Assign this input property to the **Font** property of text primitive. The newly created input property **Font** appears as a property of the graphic element (after adding the graphic element to a display).

Input properties are visible to the user after adding the graphic elements to a graphic display. The user can view the input properties through **Properties** window of graphic display in the same way as properties of built-in items are viewed. Values of input properties can be changed in the **Properties** window. For more information on changing the property values, refer to Properties Window on page 60.

For more information on defining input properties, refer to Input Properties on page 81.

> Input properties are relevant only for generic and graphic elements. They are not relevant for graphic displays and faceplate elements.
>
> When graphic aspects are invoked in an environment where input properties do not receive any values from the host, the default values of the input properties are used (for example, when a graphic element is viewed directly in the workplace, or when a faceplate element is used in a faceplate).

Expression Variables

Expression variables are used in graphic aspects to store the results of intermediate calculations. A name, a data type and an expression defines an expression variable. These variables store the result of expressions and reduce the complexity of expressions by performing calculations in intermediate steps.

Expression variables can be exposed to outside elements in a display. These expression variables do not have an expression. They are called empty expression variables due to the absence of expressions. These variables get values from input items or data entry windows.

An expression variable with an expression that is not dynamic, can be shared (on the workplace), that is, if two graphic aspects should share an expression variable, both should contain a definition of an expression variable with the same name and same type. For example, a faceplate and a graphic element can share a state using the same expression variable that is shared.

An expression variable with an expression that is not dynamic, can be persistent. The values exist on the workplace and will be retained during invocation of graphic aspects and during restart of the workplace. Value of a non-shared persistent expression variable is persisted for the element instance, that is, each instance of an element has its own persistent data.

For more information on creating expression variables, refer to Expression Variables on page 75.

Expressions

Expressions are used for setting property values and values on expression variables.

Examples of expressions

The following is an example of a legal expression:

```
$'AI.117:Control Connection:VALUE' * 14.2
```

This expression multiplies *14.2* with the *ControlConnection:VALUE* property of *AI.117* object.

The following is an expression returning a color by selecting one of three logical colors:

```
if $'Pump:Control Connection:ALARM_BLK' then
   blockedSymbol
else if $'Pump:Control Connection:ABOVE_HI_LIM2' then
   highAlarmSymbol
else
   processEvent
```

The symbols *blockedSymbol, highAlarmSymbol, and processEvent* are logical colors.

Expressions refer to sub properties of a transmitted property. Sub properties are available which return:

• Quality information about transmitted value

• Time stamp

• Properties defining whether a write operation can be performed towards the property. If not, the reason is specified.

The following is an example of an expression referencing a sub property:

```
if $'Motor14:Control Connection:UNIT'#IsBad then
    OPCError
else
    NormalText
```

This expression evaluates to *OPCError* (which can be an expression variable or an NLS text) if *IsBad* sub property of the *UNIT* property of object *Motor14* is true. Otherwise the expression evaluates to the *NormalText*.

	For more information on creating expressions, refer to Section 4, Expressions.

User Enumerations

User enumerations are user-defined data types. These are relevant for input properties.

Enumerations are pure enumerations or set enumerations. Pure enumerations permit one value at a time. Set enumerations permit values that are combinations of values from the enumeration.

For example, consider a set enumeration *FontStyle* with value *Italic, Regular, Strikeout,* and *Underline*. An example of a FontStyle value is Italic|Strikeout, that is, a value that specifies italic and strikeout font styles. This is a combination of two values, Italic and Strikeout.

	For more information on defining user enumerations, refer to User Enumerations on page 84.

Session Handling

Process Graphics 2 supports input session handling (often called faceplate session) to work in conjunction with faceplates. There are several input and graphic items that interact with the session framework to achieve the desired functionality. Following are some of the examples of input items that use session handling.

• Apply and Cancel buttons

• Direct entry windows

• Push Button

• SessionApplyCancel input item used in the non faceplate session control buttons, Apply Button and Cancel Button.

Some of the input items, for example, Push Button, possess a property called *Action* of data type *Action* enumeration. The *Action* enumeration accepts the following values:

- **Direct** - Action of the building block is executed directly when the building block is activated.

- **Applied** - Action of the building block remains pending after the building block is activated until an apply operation is performed. For example, on clicking the Apply button.

- **SystemDefault** - With this setting the action of the building block is *Direct* when the *AppliedButtonAction* user profile is *False*.

Properties of a Graphic Aspect

The properties for a graphic aspect is set through the Graphics Builder. Table 14 describes the properties of graphic aspect.

Table 14. Properties of Graphic Aspects

Property	Type	Description
AnimationRate	Integer	AnimationRate is relevant for graphic aspects that refer to the local variable _*Now*. It specifies the time interval (in milliseconds) between updates of the graphic aspect to react to the changing _*Now* variable.
BackColor	Brush	The background of the graphic aspect.
Height	Integer	The height of the graphic aspect. This controls the height of design area when the aspect is opened in Graphics Builder. For elements, this property keeps the default size while creating an instance of the element in Graphics Builder.

Table 14. Properties of Graphic Aspects (Continued)

Property	Type	Description
Width	Integer	The width of the graphic aspect. This controls the width of design area when the aspect is opened in Graphics Builder. For elements, this property keeps the default size while creating an instance of the element in Graphics Builder.
PresentationMode	ENUM	Controls the resizing and the layout strategy for the graphic aspect. This can be FreeResize, NoResize, AnchoredHorizontal, Anchored, or KeepAspectRatio. For more information on layout strategies, refer to Resizing and Layout Strategies on page 148.
Hotspot	Point	A point in the graphic aspect that is snapped to the grid point. Hotspot helps in positioning a graphic aspect when it is placed into another graphic aspect.
FocusStrategy	KeyboardNavigation	Controls the direction of focus movement and if focus should be automatically moved to next item after an apply operation. The direction can be **RightThenDown** or **DownThenRight**.
SubscriptionRate	Integer	Specifies the time interval (in milliseconds) to request subscription for the graphic aspect. The default value of this property is 0. This signifies that the accessed property determines the subscription rate. This property in the display is valid for the subscriptions in the display itself, that is, for the aspect object properties subscribed to in the display, and not for the subscriptions in the instantiated graphic elements.

Table 14. Properties of Graphic Aspects (Continued)

Property	Type	Description
NumberOfViews	Integer	The number of presentation views to be implemented by the element.
CustomViewSelection	Integer	An expression is used to determine the presentation view to be used for certain instance of the graphic aspect, through this property.
InhibitInput	ENUM	Inhibits selected or all the standard input handling functions of a graphic aspect. It is relevant only for object aware graphic aspects. It takes values such as All, None, ChangeCursor, DragSource, EnableInputDefaultFalse, LeftMouseClick, ObjectHighlight, ObjectLocking, ObjectMarking, ObjectTooltip, and RightMouseClick. These values are set enumeration values. Select *All* to inhibit all the functions. To inhibit none of the functions, select *None*. For more information on input handling, refer to Standard Input Handling on page 164.
EnableClip	Boolean	Controls drawing of a graphic item within the borders of element. If it is set to *True,* an attempt to draw the item outside borders is clipped.

Table 14. Properties of Graphic Aspects (Continued)

Property	Type	Description
Browsable	ENUM	Determines if the element is visible by default in the browsers of Graphics Builder. Change this property for a graphic element that should not be visible by default to other users. This can be Browsable, Obsolete, or Hidden.
Publish	ENUM	Determines if the element is published for use by the users building graphics. Change this for a graphic element that should not be available outside a library. This can be Published or Internal.

Handling Views for Graphic Elements

Process graphics allows the user to affect the appearance of a graphic element by changing the element properties. Following are the ways to define a custom appearance for a graphic element:

•   Using **Visible** property, to control the display of graphic items.

•   Multi view elements.

Multi view elements support different appearance of the graphic element, by having more than one view. Each appearance have a separate view.

Number of views required for an element is defined by **NumberOfViews** property of the graphic aspect.

Following are the configuration data stored separately for each view:

•   Set of graphic items.

•   Input items hosted by graphic items.

Select the view to be configured from **Views** in the toolbar.

Following are the data which are independent of the view selected.

- Set of input properties.

- Set of expression variables.

- Set of element hosted input items.

Execute the following steps to create a view.

1. Create a Graphic Element or Generic Element.

2. Right-click on the newly created element and select **Edit** from the context menu. This opens the Graphics Builder for this element.

3. In the **Properties** window, increase the value in **NumberOfViews** (For example, change the value to 3).

The user cannot reduce the number of views by decrementing the value in the **NumberOfViews** property. The following warning message appears if the user attempts to reduce the value.
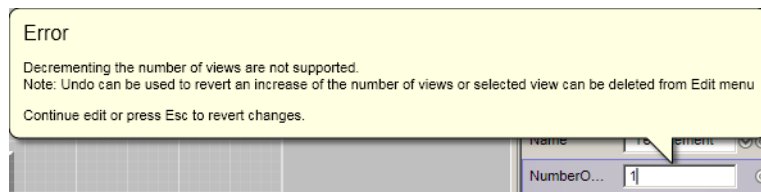
For more information on deleting a view, refer to

Supporting view selection for multi view element instance

There are two ways available for the user to select a view to be used in an instance of a multi view element.

1. Default view selection.

2. Add one or more properties to control custom view selection.

Default view selection is achieved when a dynamic expression is not applied to **CustomViewSelection** property of the element. Instances of multi view elements possess a property called **View**. This allows the user to enter the view to be displayed. The value of this property should be between zero and one less than the value in **NumberOfViews** property of the element.

Custom view selection is implemented by writing an expression in the **CustomViewSelection** property of the element. This expression should evaluate to an integer value which is between zero and one less than the value in **NumberOfViews** property of the element.

For example, consider a graphic element containing two views. One view supports horizontal orientation and the other supports vertical orientation. To provide this function:

1. Add a user enumeration type called *Orientation* having values *horizontal* and *vertical*.

2. Add an input property called *Orientation* having the data type **Orientation**.

3. Set **NumberOfViews** to *2*.

4. Assign the following expression to the **CustomViewSelection** property of the graphic element.

```
if Orientation = vertical then
    0
else
    1
```

Input properties in view selection should have the **Dynamicity** property set to **InitOnly**. This prevents the user from changing a view during runtime.

Deleting a view for multi view element

The user can delete any view which is available in a multi view element. This is possible only if the current graphic aspect is not used in any other aspect.

Do not delete a view when the graphic aspect is used in any other aspects. If the user proceeds with the deletion, the outcome is undefined with respect to the view presented by the instances of the graphic aspect.

It is recommended to cancel the operation and replace the content of the respective view with a text that the view is obsolete and the user should select another view of the graphic aspect.

Execute the following steps to delete a view:

1. Select the view to be deleted from **Views** in the toolbar.

2.   Select **Edit > Delete Current View** to delete the currently selected view. This prompts for a confirmation. Click **OK** to proceed with the deletion.
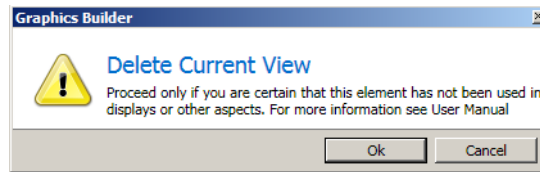
Figure 72. Confirmation for deleting a view

The first view is displayed and the value of **Views** in the toolbar is **0**. The value in **NumberOfViews** will be reduced by 1.

CustomViewSelection is used to select a view using a set of enumeration values. For more information on **CustomViewSelection**, refer to Supporting view selection for multi view element instance on page 138.

The user can delete this view and replace it with another view. Execute the following steps:

1. Delete the enumeration values corresponding to the deleted view.

2. Set the default value of the enumeration to a value corresponding to the replacement view.

This operation should be done only if it is acceptable that the deleted view is automatically substituted by a fixed replacement view.

Object Aware Elements

Aspects of **Graphic Element PG2** and **Graphics Display PG2** aspect types are called object aware aspects or elements. The term "object aware" is used because these aspects represent the object for which they are aspects.

A process operator, who finds an object aware element in a display, considers element as the "object", and not as a graphic element representing the object.

Object aware aspects are created on object types or concrete objects. Concrete objects are the objects in **Functional Structure**, **Control Structure**, or any other non-administrative structure.

Defining object aware graphic aspects on object types has the advantage that aspects are used not only for one object, but for all instances of the object type. Object aware graphic aspects are placed in concrete structures when reusability by inheritance is not desired.

Graphic elements are often placed on object types to use them as building blocks representing any instance of the object type.

Graphic displays are often placed in concrete structures because they represent sections of a plant for which reuse by inheritance is not possible. Displays are also used on object types when reuse by reference is desired. In this case, the **Object Display PG2** aspect category should be used.

> An instance of an object aware element can be created in an object aware aspect but not in generic aspect.

The graphic aspects created on object types should be set to be inherited.

Object aware aspects possess the capabilities to have references to data-entities in addition to references to resources (images, fonts, and brushes). For more information on references, refer to References on page 142.

For an example of creating object types, refer to on page 639.

Generic Elements

A graphic aspect of the aspect type **Generic Element PG2** is referred to as generic element. Generic elements are used in graphic elements, graphic displays, or other generic elements.

A generic element has the capability to contain references to resources (images, fonts, and brushes) but not references to data-entities.

Generic elements must be available in a toolbox to facilitate creation of instances from these elements. For more information on toolbox, refer to Toolbox Window on page 53.

Generic elements are created directly in the **Graphics Structure** or in a library.

Generic elements created in a library cannot be modified by the user. These elements should be inserted in the **Graphics Structure** to be made available to the users.

Generic elements created in the **Graphics Structure** are displayed in the **Toolboxes** of the Graphics Builder. For more information on creating generic elements, refer to Creating Generic Elements on page 641.

> **i** Generic elements cannot be created in any other structures other than the ones specified in this section.

References

Graphic aspects contains two categories of references.

• Data references

• Resource references

Data references are references to aspect object properties and references to views, verbs, and instances of graphic elements.

The following are some examples of data-entity references:

• This expression is a reference to the *DoubleValue* property of the *ControlConnection* aspect in the *Sinus_5* object.

```
$'Sinus_5:Control Connection:DoubleValue'
```

• This expression is a reference to the *Config View* (Configuration view) of the *Control Connection* aspect.

```
$'MixerMotor:Control Connection:Config View'
```

Resource references are references to logical colors, brushes, images, fonts, or NLS text. The following are examples of resource references:

• This expression is a reference to a logical color.

```
EventColors:unackHighAlarm
```

• This is an expression that changes the font of the text displayed based on the value of *IsAlarmActive* property.

```
if Pump:AlarmGlobalProperties:IsAlarmActive = True then
        Font ("Comic Sans MS",30,Italic,Bold)
else
        Font ("Arial",10,Regular,Bold)
```

Reference Status

A reference may be Resolved, Unresolved, or Broken. The term "Non-resolved" is used to identify a reference that is Unresolved or Broken.

Unresolved state applies only to graphic element references and in the context of off-line engineering. An unresolved reference is created by typing the name for referenced entity rather than identifying an existing entity. These references can be created without considering the existence of the referenced entity. An unresolved reference should always be resolved (for example, using the reference window) before it becomes operational.

A resolved reference is a reference by which a specific data-entity or resource can be identified. These reference possess the following qualities:

•   The referenced entity is identified even if there are other target entities having the same name as referenced entity.

•   A referenced entity can be renamed or moved without affecting the references.

A broken reference was a resolved reference but the referenced entity is removed. It is a reference that was previously resolved but cannot identify the referenced entity.

A broken reference can be repaired using any one of the following steps:

•   Allow the referenced entity to re-appear by installing it.

•   Create a new entity with the same name as the previous entity and perform a resolve operation.

•   Change references by selecting a new entity without relying on the rescue information.

An unresolved or a broken reference is resolved through the Reference Window. It uses the Name of the object / aspect property in the reference to search for the existence the system. If it is not available in the system, the user has to browse for the object / aspect property.

Reference Window (Data References and Resource References) is the primary tool for maintaining references within graphic aspects. This helps the user to modify the data and resource references made in the graphic aspect. For example, if a color is referenced for many graphic items, and the user requires a different color to be used, the Resource Reference window is used. All the expressions referencing the color are updated. For more information on Reference Window, refer to Data References and Resource References on page 94.

Reference Tool

The Reference tool can be used to view the references of a single graphic or non-graphic aspect, or references of several aspects on the same object. Using the Reference tool, the user can change the target of references, automatically resolve the broken references, and approving the non-approved aspects.

Reference Handling

Graphic aspects store data and resource references. References from one aspect to the other must be resolved for the successful execution of the aspects.

Rescue Information

Rescue information are additional details stored for references. It is used:

•   To present a non-resolved reference to the user.

•   To support resolve operations (that is, operations that resolve a reference to an entity which is identified by its name).

This additional information of references contains the following.

•   Names of the referenced object and aspect.

•   Object type of the referenced object.

–   Object type is used only while referring to a data-entity that is defined by a derived aspect. It is not used for resource references.

A derived aspect is the aspect for which the origin is at an object type and not an instance object being part of the current reference. It is created automatically when creating the instance object based on a template aspect on the object type. The template aspect is marked "Inheritable" or "Copy" to create the derived aspect.

References become broken when graphic aspects are imported into another system where the target entities do not exist. Rescue information is imported for each reference and helps in resolving broken references. This information is used by the consistency checker to update the source aspect with a reference to a new target.

Approve State

Create a graphic display with references and copy it to another object. The copied graphic display will be in an unapproved state. Data references of a copied graphic aspect should be reconnected (if required) to the appropriate target entities before opening the graphic aspect in a real workplace. Consider the following example.

An engineer creates a graphic display for an object "Machine room 1" representing a machine room. The machine room contains two machines, "Machine X" and "Machine Y". Buttons in the graphic display are used to turn the machines on and off.

The engineer creates a copy of "Machine room 1" display on the object "Machine room 2" representing another machine room.

The references for the copied display point to "Machine X" and "Machine Y" in "Machine Room 1" and the display still operates on those entities.

To prevent operation on wrong objects, a graphic aspect can be set into an unapproved state. A graphic aspect will be set to an unapproved state when it is copied or moved.

An authorized user should confirm the correctness of all references using the Reference Tool or Graphics Builder by setting the graphic aspect into an approved state.

When the aspect is in an unapproved state, the display cannot be viewed during runtime. A standard view of the display is visible to the user instead of a real display. This informs the user that the display is not approved. In this situation, user should open the display in Graphics Builder to check the correctness of references and save the display to set it into an approved state.

The consistency checker generates warnings for the aspects which are not approved. When the consistency checker resolves a broken reference, the **Approve** state for the aspect is set to Non-Approved. Only the users having the privileges of an *Application Engineer* are allowed to approve references. For more information on approving references, refer to *System 800xA, Tools (2PAA101888*).*

Approve the aspects using the **Save** in Graphics Builder or using the Reference Tool.

Off-Line Engineering

Off-line engineering supports building of displays before creation of the objects to be presented by these displays.

Prerequisites for offline engineering are:

- Machine with the 800xA system installed (controllers are not necessary).

- Object types including graphic elements installed.

- Displays built solely based on graphic elements to represent objects.

Execute the following steps to build the display:

1. Set the graphic display in **Work Offline** mode. Select **File > Work Offline**.

2. Using the element browser, select a graphic element from the object type of the intended instance object

3. Create an instance of the selected element.

4. Open the Type Editor for the Element property of the element instance and enter name of the intended target.

5. After creating objects and importing graphic display into the target system, resolve the references using Reference Tool or Graphics Builder.

Indication of Broken Graphic Element References

When a graphic aspect contains an instance of a graphic or generic element that does not exist in the system, the element instance is replaced with a place holder. The place holder is drawn as a filled rectangle in magenta color containing a white cross as shown in Figure 73.
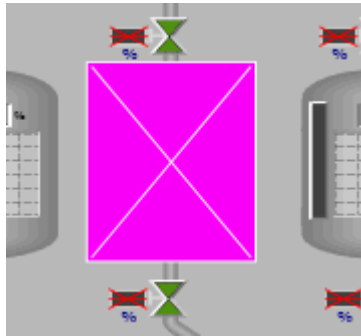
Figure 73. Indication of Broken Graphic Element Reference

The place holder is shown in the following cases:

1. For a graphic element when the element's invocation object is missing.

2. If the element itself is missing.

For more information about the missing element, open the graphic aspect in Graphics Builder and select the place holder. In Properties window, information about the used element and its property values are found.

To resolve the reference to another object when the invoked object is missing, the Data References window in Graphics Builder can be used. See Data References and Resource References on page 94.

Reference Tool can also be used to resolve the references. See Reference Tool on page 144.

If the referenced element is missing in the system, it can be imported using the Import/Export tool. The graphic aspect will use the imported element next time the graphic aspect is invoked.

Library Handling

A graphic display is not updated after upgrading a major version of a library. Consider the following example.

1. Create a library and an object type containing a graphic element in the **Object Type Structure**.

2. Release the library and create a new version. For more information on creating libraries, refer to *System 800xA, Control, AC 800M Configuration (3BSE035980*)*.

3. Update the graphic element in the new version of the library. In the **Functional Structure**, the graphic display containing this graphic element is not updated.

Restart the workplace after upgrading the library. This updates the graphic display.

Late Binding

Late Binding is used for locating system entities such as aspect object properties during runtime. A reference is retrieved using the entity names for identification. These references cannot be stored. It should be retrieved when a graphic aspect is invoked.

Early binding identifies the target system entities in the Graphics Builder and also stores references to the system entities while saving the graphic aspect. Early bound references are controlled using the Reference Map.

Late binding is supported by expression functions. For more information, refer to Functions for Late Binding on page 258.

Resizing and Layout Strategies

The *PresentationMode* property of a graphic aspect controls the resize behavior of instances of the aspect and the type of layout strategy to be used by the aspect.

Table 15 gives the different presentation modes for a graphic aspect.

Table 15. Presentation Modes

Presentation Mode	Description
Anchored	Layout of the aspect is calculated based on anchoring rules. Anchored layout depends on the value of *AnchorStyle* property of the graphic items in the aspect. Refer to Anchored Layout on page 151 for more information. This is relevant for graphic elements and graphic displays.
AnchoredHorizontal	This is applicable only for graphic elements. Refer to Horizontal Anchoring on page 156 for more information.
FreeResize	Layout of the aspect is calculated by applying a linear transformation when the actual size of aspect is different from the default size. Scale factors in vertical and horizontal directions are independent. This is relevant for graphic elements and graphic displays.
KeepAspectRatio	Layout of the aspect is calculated by applying a linear transformation when the actual size of aspect is different from the default size. Scale factors in vertical and horizontal directions are identical. It is calculated based on the actual width or height of the aspect. This is relevant only for graphic displays.
NoResize	The graphic aspect is not transformed. This is relevant only for graphic displays.

Layout Strategies

Process Graphics supports some layout formats which control the layout of a graphic aspect. The following are the layout strategies:

- Linear Transformation

- Anchored Layout

- Horizontal Anchoring

- Custom Layout

Linear Transformation

Linear Transformation is applied for a graphic aspect when the **PresentationMode** property of the aspect is set to *FreeResize* or *KeepAspectRatio*.

KeepAspectRatio. If the Presentation Mode is *KeepAspectRatio*, the size of the graphic display is changed to use the available space keeping the aspect ratio of the display, that is, the vertical or horizontal size is the limiting size. The presentation of the graphic display is centered in the non limiting dimension. The graphic items included in the graphic display are resized proportionally with respect to the width and height during a horizontal or vertical resize of the graphic display.

The background color set for the graphic display is applied to all the windows displaying the graphic display (in the Plant Explorer workplace).

Mouse interaction is possible only in the area that displays the graphic display. This is applicable for all the windows displaying the graphic display. For example, consider the graphic display shown in Figure 74. Mouse interaction does not function in the area where the background color is applied.

Execute the following steps to align the graphic items within a background image for the **Presentation Mode** *KeepAspectRatio*.

1. Add a rectangle primitive (**Shapes > Rectangle**) to the graphic display.

2. Set the height and width of this primitive equal to that of the graphic display. For more information on adding primitives to the display, refer to Selecting Graphic Items from the Toolbox on page 55.

3. Set the value of **BackColor** property of the display to a color.

4. Set the value of **FillColor** property of the rectangle primitive to an image brush.

5. Add the required graphic items to the display and align them within the rectangle.

The background color of the display will be applied to the workplace preview area of the display as shown in Figure 74.
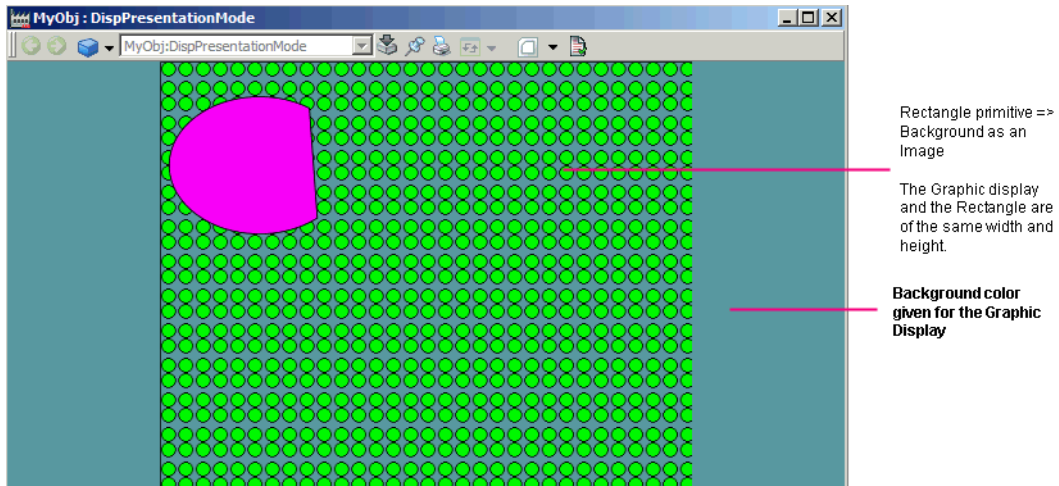
Figure 74. Aligning graphic items within a background image

FreeResize. Free resize of a graphic aspect leads to the following effects:

*   Circle primitives in the aspect do not appear the same after a resize of the aspect.

*   The text fonts become distorted as the text shrinks depending on the resize of the aspect.

Do not use Linear Transformation with free resize if these effects are not acceptable.

Anchored Layout

Anchored Layout is applied for a graphic aspect when the **PresentationMode** property of the aspect is set to *Anchored*. By setting this property, each graphic item in the aspect will have an additional property **AnchorStyle**. The following are

effects on graphic items having anchored layout if **AnchorStyle** is set to any value other than *Custom*.

- **Rotation** property is removed from graphic items.

- Properties which affect the size and position of graphic items should have only constant values. Such properties are **XPos**, **YPos**, **Width**, **Height**, and **PointList**.

The **AnchorStyle** property describes how the edge of the graphic item is placed in relation to the edge of the container (that is, the graphic aspect containing the graphic item). Table 16 gives the different values of this property.

Table 16. AnchorStyle Property values

Value	Description
All	Each edge of the graphic item anchors to the corresponding edge of the container.
Bottom	The graphic item anchors to the bottom edge of the container. The graphic item maintains its distance to the bottom when the height of element instance changes.
BottomLeft	The graphic item anchors to the bottom and left edge of the container.
BottomLeftRight	The graphic item anchors to the bottom, left, and right edge of the container.
BottomRight	The graphic item anchors to the bottom and right edge of the container.
Custom	This sets a custom layout for the graphic item. Refer to Custom Layout on page 158 for more information.
Left	The graphic item anchors to the left edge of the container. The graphic item maintains its distance to the left side and its size does not change when width of the element instance changes.

Table 16. AnchorStyle Property values (Continued)

Value	Description
LeftRight	The graphic item anchors to the left and right edge of the container. The graphic item maintains distance to both edges and changes size when width of the element instance changes.
None	The graphic item is not anchored to any edges of the container. The graphic item does not change its size and it maintains the distance to horizontal center of the container. Horizontal center of the container is a vertical line which is in the middle of the left and right sides of the element instance.
Proportional	The graphic item is resized corresponding to the container, based on the size of the item and the X, Y dimensions. An item having this anchor style value behaves similar when a linear transformation is applied, that is, the size and position of graphic items are transformed in relation to change in size of the containing element instance. For example, if the graphic item is a text item, the font size is not changed even when the size of text item changes. Another effect is that the width of lines do not change.
Right	The graphic item anchors to the right edge of the container. The graphic item maintains its distance to the right side and its size does not change when width of the element instance changes

Table 16. AnchorStyle Property values (Continued)

Value	Description
Top	The graphic item anchors to the top edge of the container. The graphic item maintains its distance to the top when the height of element instance changes.
TopBottom	The graphic item anchors to the top and bottom edge of the container. The graphic item maintains its distance to the bottom and top when the height of element instance changes.
TopBottomLeft	The graphic item anchors to the top, bottom, and left edge of the container.
TopBottomRight	The graphic item anchors to the top, bottom, and right edge of the container.
TopLeft	The graphic item anchors to the top and left edge of the container.
TopLeftRight	The graphic item anchors to the top, left, and right edge of the container.
TopRight	The graphic item anchors to the top and right edge of the container.

Table 17 describes the effect of changing the height of the container for different values of **AnchorStyle**. This is described for the vertical dimension.

Table 17. Summary of AnchorStyle property

Value of AnchorStyle	Distance between graphic item top and container top	Height of graphic item	Distance between graphic item bottom and container bottom
Top	Fixed	Fixed	Changed
Bottom	Changed	Fixed	Fixed

Table 17. Summary of AnchorStyle property (Continued)

Value of AnchorStyle	Distance between graphic item top and container top	Height of graphic item	Distance between graphic item bottom and container bottom
Top + Bottom	Fixed	Changed	Fixed
Not Top and Not Bottom	Changed	Fixed	Changed

The effect of anchoring an item to **bottom**, but **not to top**, is that the distance between bottom of the graphic item and bottom of the container is fixed when the height of the container changes. The height of the graphic item does not change. The Y position of the graphic item has to be changed to achieve this effect.

Anchoring to **top**, but **not to bottom** works in a similar way. The distance between top of the graphic item and top of the container does not change when height of the container changes. The height of the graphic item does not change. The Y position of the graphic item also does not change.

Anchoring to **top** and **bottom** ensures that the distance between top of the graphic item and top of the container or bottom of the graphic item and bottom of the container changes when height of the container changes. The height of the graphic item is changed to facilitate this.

The last anchoring permutation in the vertical position is **not top** and **not bottom**. In this case, changing height of the container has the following effects.

• The height of the graphic item does not change.

• The absolute Y position of the graphic item maintains its distance to the center of the element.

Anchoring in the horizontal dimension works the same as in the vertical dimension. Anchoring in the vertical and horizontal dimensions work independently of one another and it should be easy to deduct the effect of combinations of vertical and horizontal anchoring.

A graphic element will have font and/or line or frame width properties. Layout policies which incorporate linear transformation affect font size and line/frame width when an instance of the element is changed. The advantage of the anchored

mode is that a linear transformation is not applied and font size and line/frame width remain consistent.

Setting proportional anchoring style for a graphic item maintains the relative size and position of the item in the X and Y dimension, when resizing the container. Table 18 describes the effect of size change for proportional.

Table 18. Proportional Anchoring Style

Layout Strategy	Size and Placement of graphic items	Inner Geometry
Linear Transformation	Size transform applied	Size transform applied
Proportional Anchoring	Size transform applied	Size transform not applied

Horizontal Anchoring

Horizontal Anchoring is applied for a graphic aspect when the **PresentationMode** property of the aspect is set to *AnchoredHorizontal*. Horizontal anchoring is relevant only for graphic elements and effective only for instances of graphic elements. In horizontal anchoring mode, each graphic item in the aspect will have an additional property **AnchorStyle**.

The **AnchorStyle** property describes how vertical edges of graphic items are anchored to the corresponding vertical edges of the graphic element instance containing the graphic item.

Changing width of the graphic element instance affects layout of the element instance depending on the **AnchorStyle** property of graphic items.

Changing height of the element instance applies a linear size transform where the size factor is equal to the relative height change. The size transform is equally applied in vertical and horizontal dimensions.

Consider the following example for horizontal anchoring.

1. Create a graphic element containing a text item and status box horizontal placed above the text item.

2. Set the **PresentationMode** as *AnchoredHorizontal*.

3.   Set the **AnchorStyle** for the text item as *LeftRight* and the status box horizontal
      as *Left*.

4.   Add the graphic element into a graphic display.

Figure 75 shows the default layout of the graphic element.
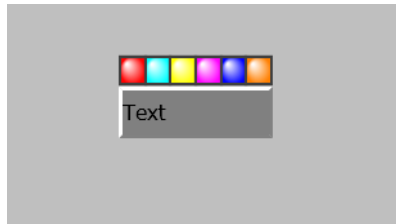
Figure 75. Default Layout

Figure 76 shows the layout of the graphic element after increasing the width of the
element.

Figure 76. Layout after increasing the Width of the Element

Figure 77 shows the layout of the graphic element after increasing the height of the
element.
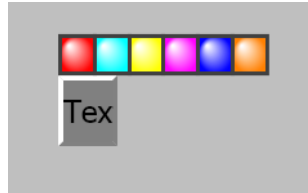
Figure 77. Layout after increasing the Height of the Element

To increase the width of the element by providing more space for the text, set the **AnchorStyle** property for the text primitive as *LeftRight*.

Table 19 gives the different values of this property.

Table 19. AnchorStyle Property values

Value	Description
Left	The graphic item anchors to the left edge of the container.
Right	The graphic item anchors to the right edge of the container.
LeftRight	The graphic item anchors to the left and right edge of the container.
None	The graphic item is not anchored to any edges of the container.

Custom Layout

Custom layout can be selected for graphic items in graphic aspects using anchored layout. Custom layout is created by performing the following:

1. Set the **AnchorStyle** property to *Custom*.

2. Write dynamic expressions for properties which control the size and position of graphic item, such as **XPos**, **YPos**, **Width**, **Height**, **Rotation**, and **PointList**.

A custom layout graphic item is different from other items because it has a **Rotation** property and is permitted to attach dynamic expressions to layout properties. For more information on the **AnchorStyle** property, refer to Anchored Layout on page 151.

When the size of an element instance is the default size, every item in the element appears at the configured position. If there is any deviation in size of the element instance, then the size and position of each contained graphic item is recalculated based on the **AnchorStyle** value.

Layout expressions typically refer to the *_Width* and *_Height* local variables to calculate new positions for graphic items. These variables change when size of the element instance changes.

Pixel Snapping

Graphic displays in Process Graphics are rendered using Microsoft WPF. The WPF technology uses anti-aliasing when rendering. This results into a better look and feel without any jagged edges. It is a limitation, that is, in some situations (for example, horizontal and vertical lines), the result may become less sharp. One way to minimize this effect is by creating graphic displays in their correct size. Also set the **SnapsToDevicePixels** property to **True**. This has a slight performance cost and only exist on some of the primitives.

Handling Mouse Events

Input items can possess a property **Enabled** which specifies whether the input items should be executed or not.

When the graphic item is a graphic element or generic element, both item and element hosted input items are executed.

When the graphic item is an instance of a graphic element with **EnableInput** as **True**, the following actions are executed.

•   Defined by object awareness.

•   Not disabled by **InhibitInput** property.

•   Defined by input items.

Mouse Event Consumer

A graphic item may or may not be a mouse event consumer. If it is not a mouse event consumer, it is click transparent.

A graphic item is a mouse event consumer for the following reasons.

1. The graphic item is inherently a mouse event consumer, for example, a list item.

2. The graphic item contains item hosted input items.

3. The graphic item is an instance of an element aspect containing element hosted input items.

4. The graphic item is an instance of an element aspect containing expressions that refers to any of the mouse variables.

5. The graphic item is an object aware element instance for which input is enabled.

The background of a graphic display is always a mouse event consumer. Mouse events are consumed by a mouse event receiver. Only one mouse event receiver can react on mouse events.

Mouse Event Receiver

Several graphic items in a graphic aspect can be mouse event consumers, but only one graphic item can be a mouse event receiver at a time.

Feature Pack Functionality _____

An exception to this is when the **RetainObjectAwareness** property of an input item is set to *True*. For more information, refer to Retain Object Awareness on page 162.

_____

Mouse event receiver is a graphic item (or the graphic display background) which responds to the mouse events by executing mouse actions.

Following are different mouse actions.

•   Action of item hosted input items.

- Execution of expressions because of the changing states of mouse variables, if the graphic element is an element item.

- Action of element hosted input items.

- Standard input handling functions, which are performed by object aware elements.

All actions which are not disabled are executed. This is not true for the following situations.

- All mouse events are consumed when drag/drop operation (a part of standard input handling) is executed.

- Some graphic items which are inherently mouse event consumers, cannot be hosts for input items.

Finding a mouse event receiver

A mouse event receiver is:

- The mouse event consumer that has captured the mouse during a mouse capture event.

- One of several mouse event consumers over which the mouse pointer hovers.

There is always a mouse event receiver when the mouse is over a graphic display. If the mouse is not over a graphic item which is a mouse event consumer, the display background becomes the mouse event receiver.

The capture item is the mouse event receiver even if the mouse is moved to a position where there can be another mouse event receiver if the mouse was not captured. The mouse event receiver does not change from the point till the user clicks a mouse button after all mouse buttons are released. Consider the following.

For a capture item, the *_MouseOver* local variable indicates that the mouse is over it. When the user moves the cursor outside the capture item, the *_MouseOver* variable is set to *False*. Moving the cursor over another mouse event receiver does not set this variable *True* because this item is not the mouse event receiver.

Retain Object Awareness

This section applies to the System 800xA 5.1 Feature Pack release only.

The standard behavior of Process Graphics concerning the mouse input is that exactly one graphic item is the mouse event receiver. This is not the desired behavior in all cases.

Some users require a graphic element to contain embedded graphic items with input items. For example, having a tooltip for a graphic item in a graphic element, that is different from the tooltip of instances of the graphic element.

Embedded graphic items possessing input items, are mouse event consumers. The standard behavior is that only one graphic item can be the mouse event receiver. When pointing at an embedded mouse event consumer, it takes over input from the hosting element instance. The following are the effects:

- The marking of the graphic element instance disappears.

- The mouse cursor is not a hand but a pointer.

- Left click does not invoke the default action of represented objects.

- Right click does not invoke the context menu of the objects.

These effects are not acceptable in many scenarios.

All input items have the property **RetainObjectAwareness**. If this property is set to *True* to an input item of an embedded graphic item, the behavior changes such that the hosting element instance maintains input when pointing at the embedded item. The following are the effects:

- The graphic element instance is presented as marked and the mouse cursor is a hand.

- Left click invokes the default action of the objects, and any actions performed by additional input items that are triggered on a left click.

- Right click starts by invoking the actions performed by additional input items that are triggered on a right click and then opens the context menu of the objects.

- An additional tooltip item defines a tooltip that replaces the default tooltip of the graphic element instance when pointing at an embedded graphic item hosting the tooltip item.

If the embedded graphic item has more than one input item, it is sufficient that one input item has the **RetainObjectAwareness** property set to *True* to achieve the effect.

The expression function **IsMarked()** allows graphic elements to implement a user-defined marking, that is, without using the built-in default marking. This function returns *True* not only when pointing directly to the element instance but also when pointing at an embedded graphic item being marked with **RetainObjectAwareness**.

Capturing the mouse

A graphic item, which is a mouse event consumer, or a display background is said to capture the mouse when a mouse button is clicked anywhere over the graphic display. The display remains in the capture state until the mouse buttons are released. Graphic item or graphic display which captures the mouse is called the capture item.

The local variable *_MouseCaptured* specifies whether the capture is in progress. This variable evaluates which mouse button is pressed. A capture state is retained till all the mouse buttons are released but the state of the variable *_MouseCaptured* stores its initial value on commence of a capture operation.

_MouseButtonState local variable is used to check the current state of the mouse buttons while the mouse is captured.

Click Transparency

Graphic items which are not event receivers are click transparent or mouse event transparent. Mouse events are not consumed by a click transparent graphic item.

An object aware element instance may be a mouse event consumer or click transparent. An object aware element instance is a mouse event consumer if the element and the parent aspects have **EnableInput** as **True**. Otherwise the element instance is click transparent.

Property value can control click transparency only for object aware element instances.

For example, consider a *PushButton*. It is a mouse event consumer as it contains element hosted input items. It has a property called **Enabled** which controls whether the button responds to the mouse up event or not. Disabling this button does not make it click transparent.

Consider a *PushButton* placed on an object aware element. It is a mouse event consumer even if the object aware element instance has the **EnableInput** property set to **False**. The *PushButton* executes the functions unless it is disabled. The **IsInputEnabled** function can be used to control whether the *PushButton* is enabled or not. This function returns when the objects aware element and the parent elements have **EnableInput** as **True**.

Standard Input Handling

Standard input handling is implemented by object aware graphic aspects. It implements the presentation of information or performance of operations on objects represented by object aware graphic elements.

The features of standard input handling are:

*   Default action is defined to invoke the default aspect of the presented object. The default aspect is typically a Faceplate. This is invoked by a left-click on the element/display.

*   Right-click on the element/display invokes context menu of the object.

*   Object Locking may be used when there are many operators to control the objects. Graphic element is surrounded by a white frame, which indicates that the object is locked.

*   Object marking is applied when the cursor is over an instance of a graphic element.

    –   An object marking rectangle appears on the element.

*   A tooltip showing the name of the aspect object appears.

*   The cursor changes to a hand icon.

*   A graphic item can act as a drag source.

*   Object highlighting adornments are presented for the graphic elements for which invocation object becomes the "highlighted object".

> **i**   Standard Input Handling is relevant only for object aware elements.

> 💡   User profiles for standard input handling are set through **Graphics Profile Values PG2** aspect in the **User Structure**.

Invoke Default Action

Left-click on a graphic aspect invokes the default action of the target object. The default action is typically to invoke a faceplate of the object.

Selecting **InhibitInput** property as **LeftMouseClick** does not allow a left-click operation on the graphic aspect. This does not invoke the default aspect.

Invoke Object Context Menu

Right-click on a graphic aspect invokes the context menu of the target object. Following are the functions added by Process Graphics, which are available in the context menu of an object.

- **Edit** to launch the Graphics Builder to edit a graphic aspect.

- **Diagnostics** to invoke the Diagnostics window. For more information on diagnostics, refer to Section 6, Diagnostics Window.

- **Acknowledge All Visible Alarms** to acknowledge alarms in a graphic aspect using a single command. For more information on acknowledging alarms in a graphic aspect, refer to *System 800xA, Operations (3BSE036904*).*

Selecting **InhibitInput** property as **RightMouseClick** does not allow the user to right-click on the graphic aspect. This does not invoke the context menu of the object.

Feature Pack Functionality _____

Execute the following steps to restrict the access of **Acknowledge All Visible Alarms** for specific users:

1. Select the **Graphics Profile Definition 2** aspect from **Aspect System Structure > Process Graphics 2** .

2. In the **Item Definition** tab, add a new profile by specifying the following
    values:

    –   **Id** as *DisplayAcknowledgeAllVisibleAlarms*
    –   **Name** as *DisplayAcknowledgeAllVisibleAlarms*
    –   **Description** as *Controls whether menu item Acknowledges All Visible
        Alarms is displayed in context menu*

    Click **Add** to add this profile and then click **Apply** to save the changes.
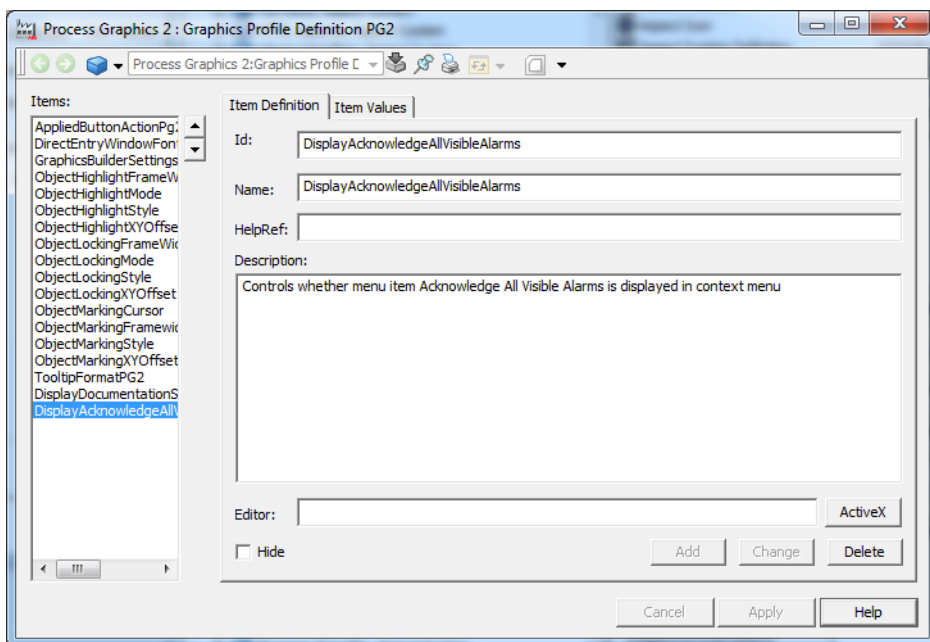
Figure 78. Graphic Profile Definition - Item Definition tab

3. In the **ItemValues** tab, select **Value Type** as *Boolean* and select the **Value**
    check box.

    Click **Apply**.
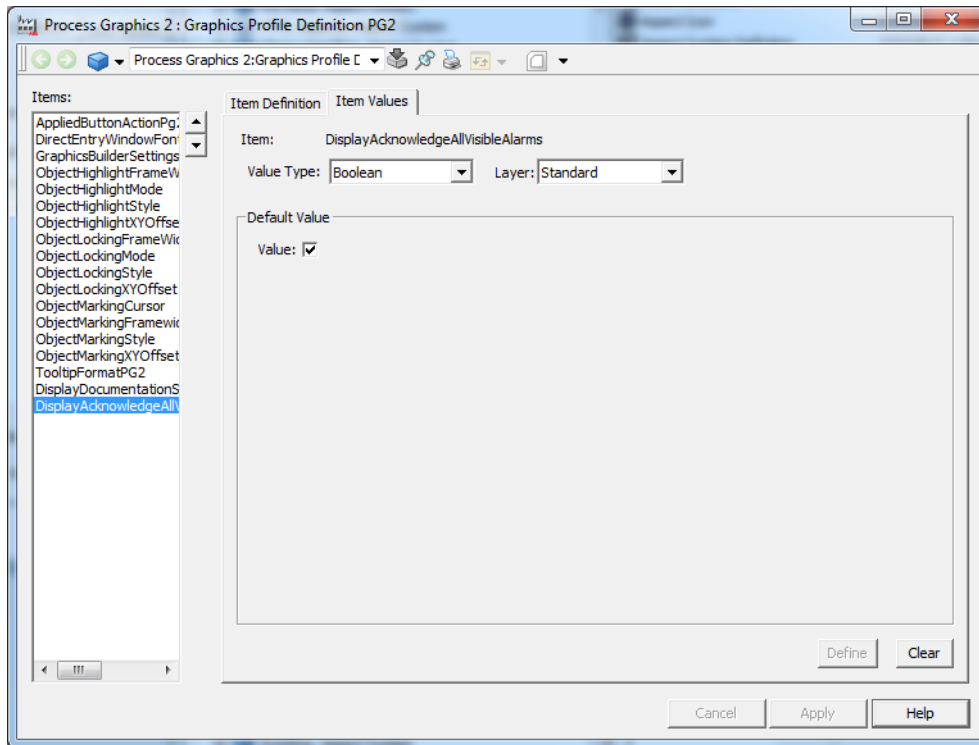
Figure 79. Graphic Profile Definition - Item Values tab

4. Select **Graphic Profile Values PG2** aspect for specific users (**User Structure > User Groups > [user name]**) who should not be able to view **Acknowledge All Visible Alarms** from the context menu.
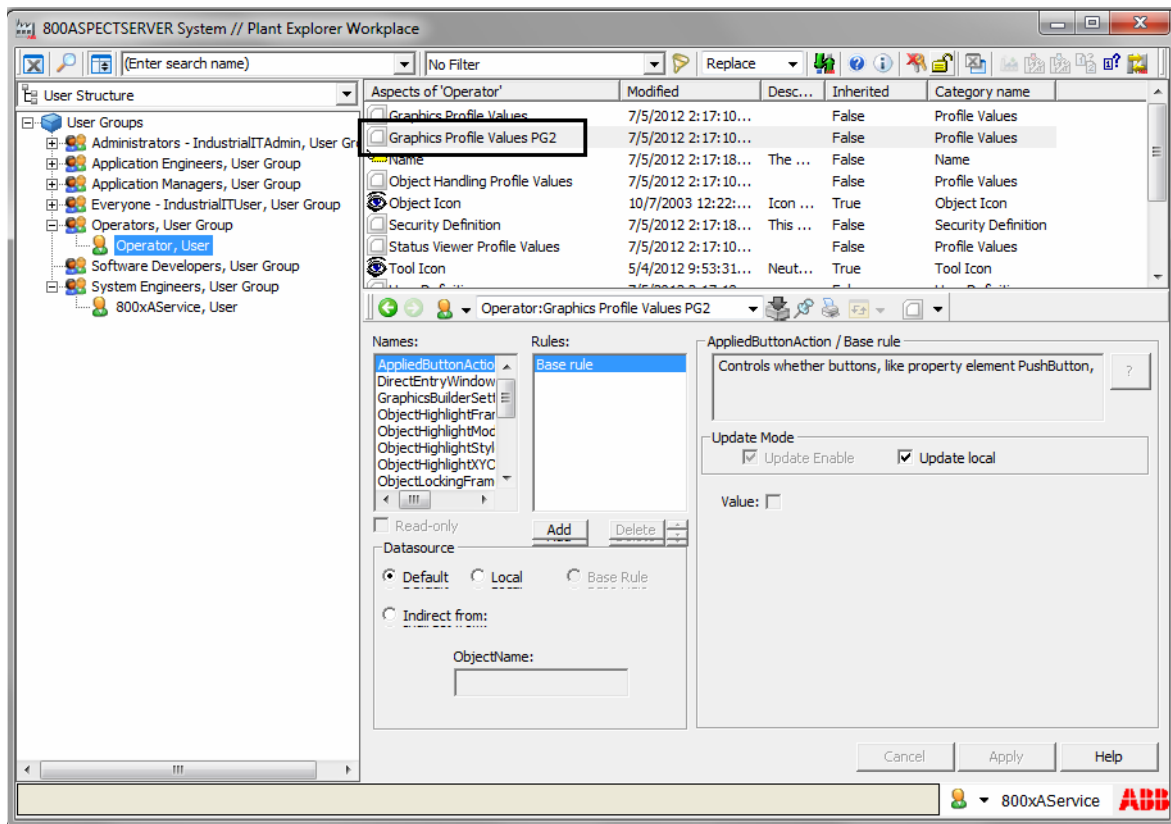
Figure 80. Graphic Profile Values aspect for the Operator in the User Structure

5.   In the profile **DisplayAcknowledgeAllVisibleAlarms**, select the **Datasource** as *Local* and remove the selection of the **Value** checkbox.

Click **Apply**.
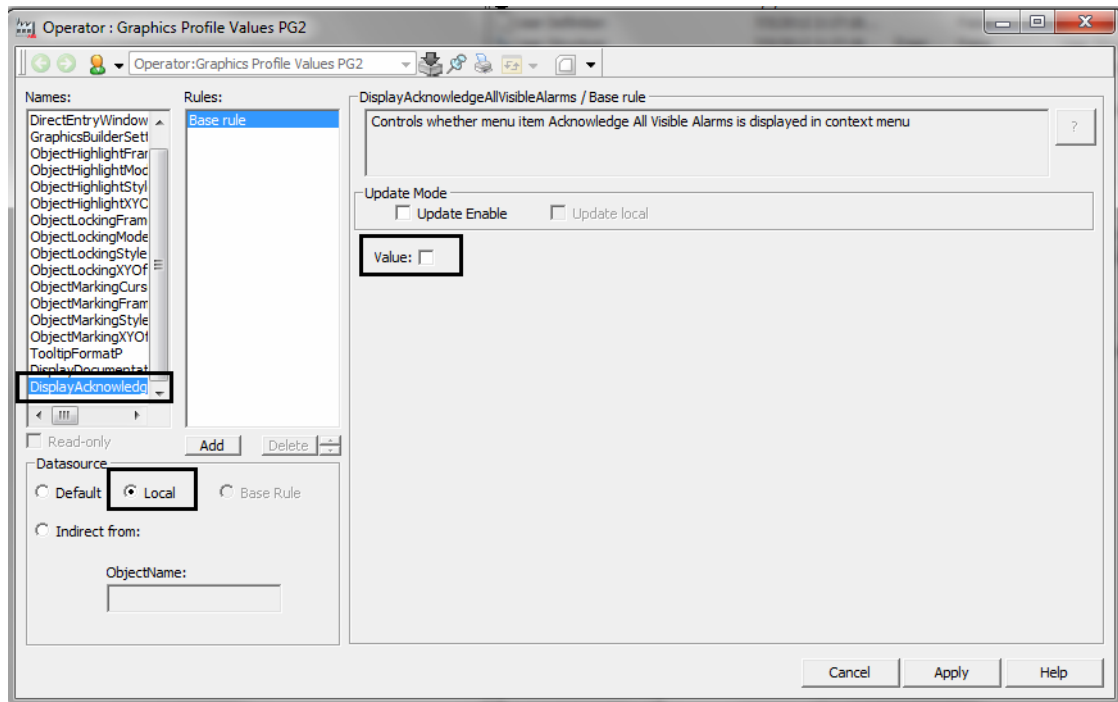
Figure 81. Updating the Graphic Profile Value for Acknowledging All Visible Alarms

> The **Acknowledge All Visible Alarms** will not be visible in the context menu for these users.

Drag Source

An instance of an object aware element is a drag source. For example, the user can drag objects presented in a display to a trend viewer. The trend viewer presents trend data from the dragged objects.

Selecting **InhibitInput** property as **Drag Source** does not allow any drag operations.

ChangeCursor

The cursor changes from 🔍 to 🖑 while pointing to an object aware graphic element.

The change of cursor while pointing at an element is disabled by selecting **InhibitInput** property as **ChangeCursor**.

Table 22 describes the user profiles used for change cursor.

Table 20. User Profiles for change cursor

Name	Description	Values
ObjectMarkingCursor	Cursor type used to point an object aware element.	0 = system default (arrow) 1 = hand (default value)

ObjectTooltip

A tooltip displays the object name of the corresponding aspect object to which the object aware element points to.

Selecting **InhibitInput** property as **ObjectTooltip** will not enable a tooltip for an aspect.
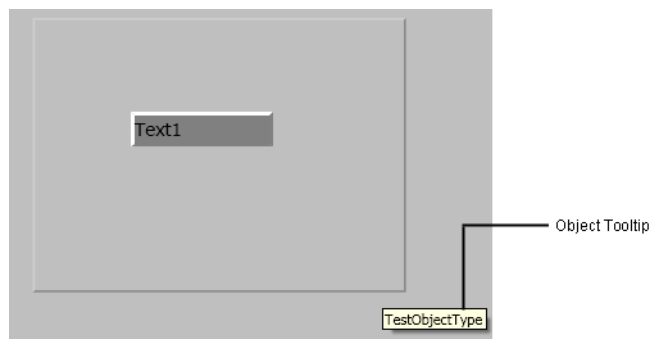
Table 22 describes the user profiles used for object tooltip.

Table 21. User Profiles for object tooltip

Name	Description	Values
TooltipFormatP	Determines how to present the tooltip of a graphic element.	%ObjectName% = name of the object. %ObjectDescription% = object description. %ObjectPath% = full path of the object. \n = a line break

Object Marking

Object marking includes selection of a graphic aspect, cursor changes, and object tooltips.

Object marking is applied while selecting an object aware graphic element placed in a graphic display or another graphic element. Marking is not done on the outermost aspect.

Selecting **InhibitInput** property as **ObjectMarking** will not enable marking for an aspect.
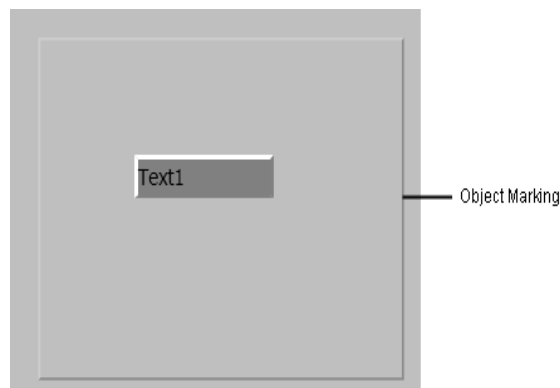
Table 22 describes the user profiles used for object marking.

Table 22. User Profiles for object marking

Name	Description	Values
ObjectMarkingFramewidth	Framewidth of object marking adorner (in pixels).	Default value = 2
ObjectMarkingStyle	Determines how marking should be represented.	0 = no representation 1 = Classic3DFrame (default value)
ObjectMarkingXYOffset	Offset for the marking frame (in pixels).	Default value = 7

Object Highlighting

Object highlighting works in two modes:

• In *Highlight Follows Faceplate Focus* mode, the highlighted object is set from the activated faceplate or when faceplate becomes the focused window.

There can be many faceplates open at the same time. Object highlighting enables all the object aware elements that correspond to the currently active faceplate. These belong to the same aspect. This reduces the risk of operating the wrong object when many faceplates are open.

• In *Highlight Follows Mouse Movements* mode, the highlighted object is set when the cursor hovers on the object aware element. This mode does not show any object marking adornments.

Object highlight also reduces the risk of operating the wrong object when using hot keys configured to operate on highlighted object. When the highlight mode is *Highlight Follows Faceplate Focus*, it is not required to move the cursor over an object to apply hot key commands on the object.

Selecting **InhibitInput** property as **ObjectHighlighting** disables highlighting of aspects.
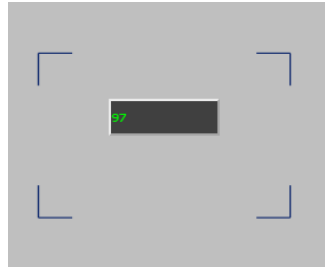
Table 23 describes the user profiles used for object highlighting.

Table 23. User Profiles for object highlighting

Name	Description	Values
ObjectHighlightFrameWidth	Framewidth of the highlighting rectangle (in pixels).	Default value = 2
ObjectHighlightMode	Determines how the highlight should be triggered.	0 = no highlighting 1 = Highlight follows faceplate focus (default value) 2 = Highlight follows mouse movements
ObjectHighlightStyle	Determines how the highlight should be represented.	0 = no representation 1 = Using a rectangular frame 2 = Using hooks (default value)
ObjectHighlightXYOffset	Offset for the highlighting rectangle (in pixels).	Default value = 2

Object Locking

Object Locking is used when there are many operators to control the objects. Access rights are given only for one operator to operate the objects.

To indicate that the object is locked, the graphic element is surrounded by a white frame for the user who has locked the object and the graphic element is surrounded by a yellow frame for other users.

Selecting **InhibitInput** property as **ObjectLocking** will not enable the object locking function for an aspect.
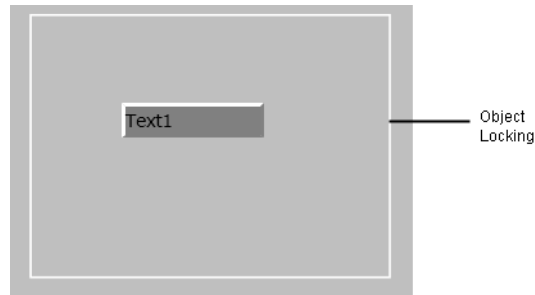
By default, object locking is not enabled. Enable object locking through the user profile **ObjectLockingMode** as described in Table 24. For more information on configuration of lock server, refer to *System 800xA, Administration and Security (3BSE037410*)*.

Table 24 describes the user profiles used for object locking.

Table 24. User Profiles for object locking

Name	Description	Values
ObjectLockingFrameWidth	Framewidth of the locking adornment (in pixels).	Default value = 2
ObjectLockingMode	Enable or disable object locking adornment.	0 = Off (default value) 1 = On

Table 24. User Profiles for object locking

Name	Description	Values
ObjectLockingStyle	Determines how the locking should be represented.	0 = no representation 1 = Solid (default value) 2 = Dash 3 = Dot 4 = DashDot 5 = DashDotDot 6 = Hash rectangle
ObjectLockingXYOffset	Offset for the locking rectangle (in pixels).	Default value = 5

Standard Input Handling used in a Composite Object Type

An object type containing formal instances of another object type is termed as Composite Object Type.

All instances of object aware elements (that is, instances of **Graphic Element PG2** aspect type) possess a property called **EnableInput**. This property can be used to determine the input strategy to be used for a complex graphic element.

The **EnableInput** property of the graphic aspect controls the input handling of the aspect. An input operation cannot be performed for the graphic aspect if value of this property is *False*.

Input strategies are of two types:

•   Tightly coupled

•   Loosely coupled

Consider the example described in Figure 82 having *Reactor* as the composite element. The *Reactor* object contains *Motor* and *Valve* objects with graphic elements on each object. *Reactor* also has a graphic element that includes *Motor* and *Valve* graphic elements.

For tightly coupled, input operation can be performed on *Reactor* but not for *Motor* or *Valve*. Do not set **InhibitInput** for *Reactor* but set **EnableInput** as **False** for *Motor* and *Valve*.

For loosely coupled, input operation cannot be performed *Reactor* but it can be performed on *Motor* and *Valve*. Set **InhibitInput** for *Reactor* and set **EnableInput** as **True** for *Motor* and *Valve*.

Following is an example of the definition and usage of composite object type. Refer Figure 82.

In this example, *AI* and *Reactor* are object types. *AI* contains aspects of type **ControlConnection** and **Graphic Element PG2** and *Reactor* contains aspect of type **Graphic Element PG2**.

Reactor contains two objects, *Motor* and *Valve* that are instances of *AI* object type.

The *Reactor* object type is a composite object type as it contains two instantiated objects of the object type *AI*. *Reactor* is the parent object, *Motor* and *Valve* are the child objects.
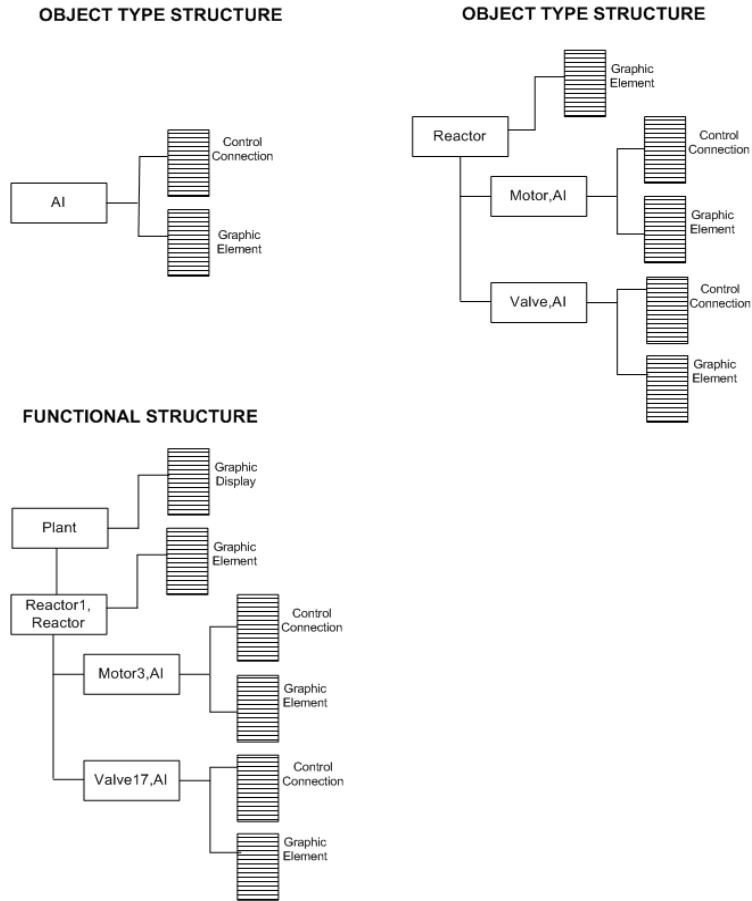
Figure 82. Composite Object Types

Figure 82 contains the following objects.

– *AI:GraphicElement* presents data by using generic building blocks and expressions referring to *AI:Control Connection*.

– *Reactor:GraphicElement* presents data by using generic building blocks and expressions referring to *Motor:Control Connection* or by creating instances *Motor:Graphic element* and *Valve:Graphic Element*.

– *Reactor1:GraphicDisplay* presents data by instantiating *Motor3:Graphic Element* and *Valve17:Graphic Element* or by using expressions referring to *Motor3:Control Connection* and *Valve17:Control Connection*.

AI and *Reactor* are object types.

Motor and *Valve* are instances of *AI*.

Reactor1 is an instance of *Reactor*.

Figure 83 shows the **Object Type Structure** after creating object types and instances of objects.
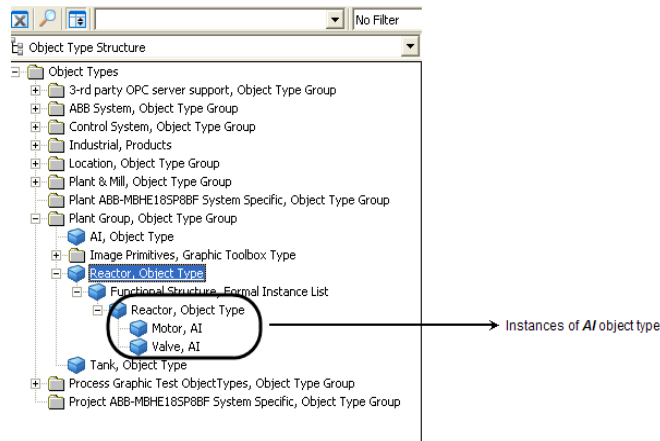
Figure 83. Object Type Structure

Figure 84 shows the **Functional Structure** after creating instances of objects.
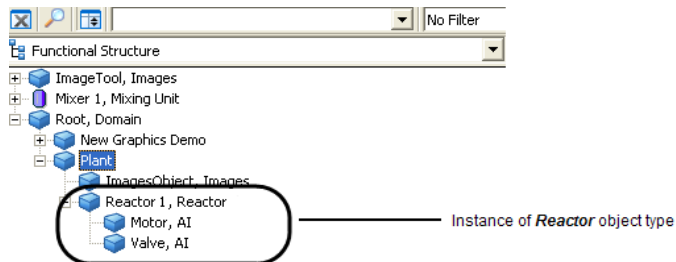
Figure 84. Functional Structure

Security

Graphic aspects support security. A security check is performed to prevent reading or writing of data without the required permissions.

Operation/Permission mapping is a mapping between operations and permissions for the built-in aspect categories. This shows the access rights required for a user to execute different operations. The user requires read permission to view the graphic aspect and configure permission to modify the aspect as shown in Figure 85.
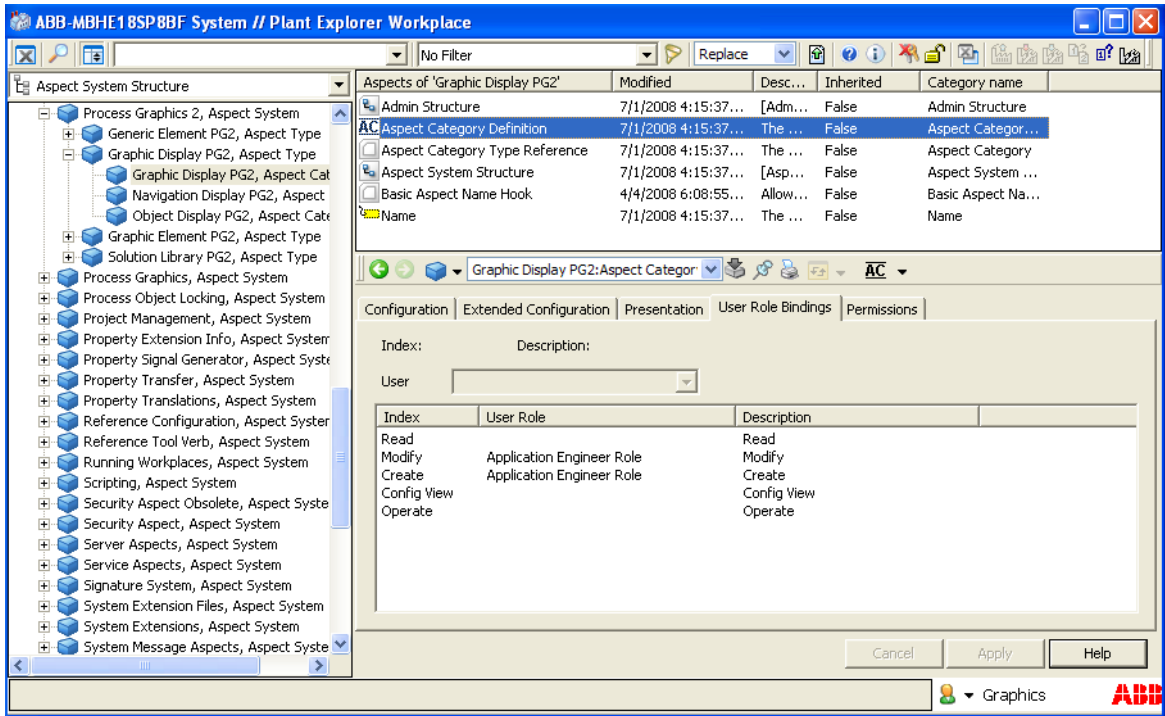
Figure 85. Permission/Operation Mapping on a Graphic Aspect

Table 25 gives the different security operations in graphic aspects. The security operations control the privileges for the user to access different views or to configure the aspect.

Table 25. Graphic Aspect Security Operations

Operation	Description
Read	Not having this permission prevents the graphic aspect from being displayed to the user.
Modify	Not having this permission disables *Edit* in the context menu of aspects.

Printing a Graphic Aspect

The generic Workplace Print function can be used to print graphic aspects but the result may not be satisfactory. This section describes an alternative print function for graphic aspects which performs better.

To print a graphic aspect, right-click the background of a graphic aspect and select **Print** from the context menu (see Figure 86).
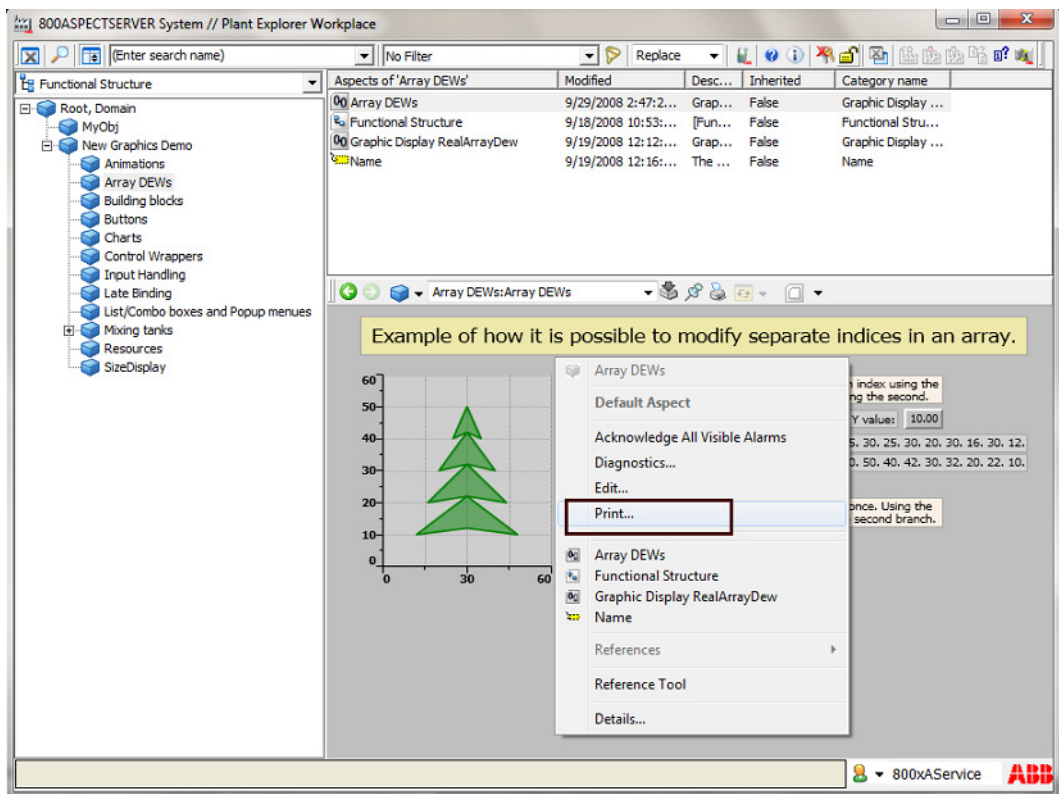
Figure 86. Print option in the Context Menu

The **Print** dialog appears along with a Print Preview of the graphic aspect (see Figure 87).
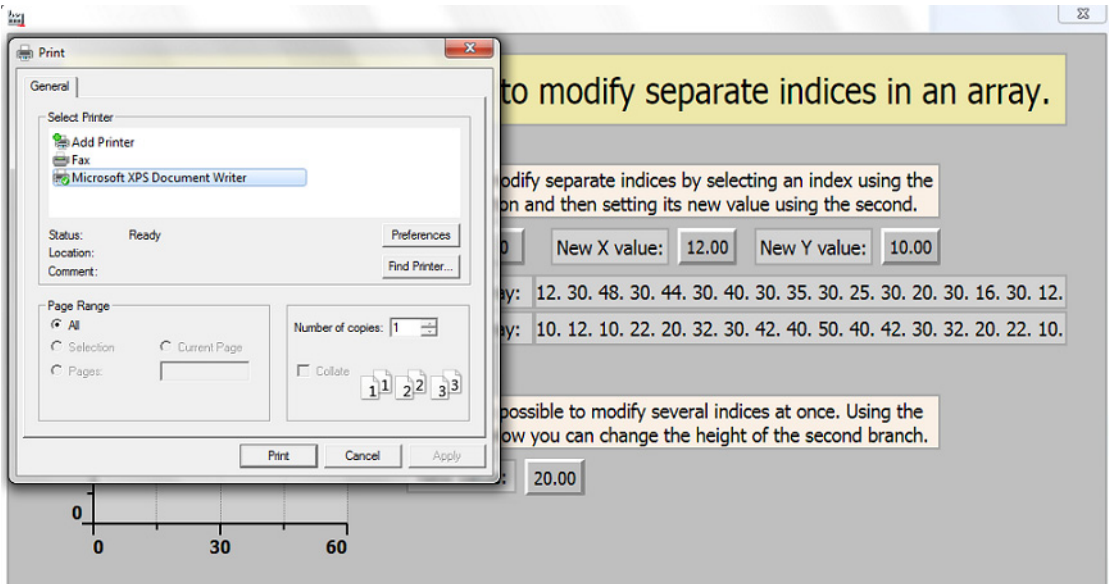
Figure 87. Printing a Graphic Aspect

Select the required printer and click **Print** to print the graphic aspect.

> To modify printout colors, open the file *printercolors.map* from *C:\Program Files\ABB Industrial IT\Operate IT\Process Portal A\config* in a notepad and follow the instructions specified in this file.

Section 4 Expressions

Expressions allow the user to subscribe to system data and to perform calculations based on system data. The following are the examples of system data that an expression can refer to.

- Aspect object properties.

- Colors.

- Expression variables.

- Input properties.

Expressions are assigned to properties of graphic items and input items used in the graphic aspect or to expression variables. Expression variables are used for reducing the complexity of individual expressions, which can also be reused for creating other expressions. For more information on creating and assigning expressions, refer to Expression Editor on page 64.

Execution of an expression takes place automatically when any subscribed system data referred to by the expression changes. An expression gets executed,

- When the graphic element is initially loaded.

- When the values of aspect object properties that are referred in the aspect is changed.

- When the values of expression variables or input properties that are referred in the expression is changed.

- When the local variables, mouse variables, and input item out terminals are changed.

Expression Syntax

An expression can contain operators, operands, and functions. An expression performs calculation based on parameter values and yields one value being the result. Each value is of a well defined data type. Examples of data types are Real, Integer, Boolean, Color, and String.

This section describes the syntax to be used while creating an expression.

- Use **if-then-else** statement for conditional execution. Refer to Conditional Statements on page 189 for the syntax.

- Use operators in the expressions for calculations. Refer to Operators on page 185 for information on the different operators.

Examples of Expressions

Example 1:

```
if objName:Value > 75 then
   alarmColor
else
   RGB (0,255,0)
```

where *objName:Value* represents a reference to the *Value* property of the object, by the name of *objName* and *alarmColor* is the name of a logical color. *RGB* is a function which takes 3 values as arguments and gives a color based on the given values.

This expression evaluates to *alarmColor* if the *Value* property exceeds 75. Otherwise it evaluates to *RGB (0,255,0)*.

objName:Value and *alarmColor* are known as symbols. Symbols are human readable representation of references to entities such as properties, logical colors, and expression variables.

There may be several variations of a symbol. For example, consider the following expression.

```
$'lc::Event Colors:blockedSymbol
```

The following are the variations of the above expression.

- blockedSymbol

    This represents the name of a logical color.

- $' Event Colors:blockedSymbol'

- lc::blockedSymbol

- $' lc::Event Colors:blockedSymbol'

The above specified expression formats are used for clarity or uniqueness of the symbol.

Example 2:

```
ip::myCol
```

where *myCol* is an input property having a color as its value. This value is assigned to the *FillColor* property of a graphic item; *ip* is the prefix that designates *myCol* being an input property. This example can also be written as follows.

```
myCol
```

myCol is also known as a symbol. Symbols are human readable representation of references to entities such as properties, logical colors, and expression variables.

Example 3:

```
$'lc::Event Colors:blockedSymbol'
```

where *blockedSymbol* is a logical color. In this example, symbol quoting is used because there is a space in the name *Event Colors*. For more information, refer to Symbol Quoting on page 265.

This example can also be written as follows.

```
blockedSymbol
```

Expressions containing references to data entities or resources can be written without specifying prefixes. Table 66 gives the list of prefixes.

Operators

Operators are used while creating expressions. The operators which are used in expressions are:

- Unary Operators

- Binary Operators

- #

- []

- if-then-else

is used for invocation of properties and methods of data types. For more information on data type methods and properties, refer to Other Data Types on page 219.

[] is the index operator. This operator is applicable only for array data types and Integer data type. The index operator is zero based. This operator is used to evaluate the individual bits on integer values. For example, *MyInteger [3]* returns *True* if the fourth bit of *MyInteger* is set to 1, else it returns *False*. The least significant bit has the number 0. Using the index operator on an integer is equivalent of using *#Bit( )* method, that is, *MyInteger#Bit(3)* also gives the same result as that of *MyInteger [3]*.

Unary operators are applied to a single operand. Table 26 gives the list of unary operators.

Binary operators use two operands. Table 27 gives the list of binary operators.

The bitwise logical operators (&, |, ^) perform boolean logic on corresponding bits of two integral expressions and return a compatible integral result with each bit conforming to the Boolean evaluation.

The conditional logical operators (&&, ||) perform boolean logic on two boolean expressions. The expression on the left is evaluated, and then the expression on the right is evaluated. Finally, the two expressions are evaluated together based on the boolean operator. It returns a boolean result corresponding to the type of operator used.

Data types contain properties and methods, which can be accessed using the **#** operator. The following is the syntax for accessing the properties and methods.

- *value#property* for properties

- *value#method (parm1,...., parmN)* for methods

Examples:

- *pr::MyStringValue#Length* uses the *Length* property of the *String* data type to retrieve the length of the string value *pr::MyStringValue*.

- *pr::MyRealArray#GetValue(0)* uses the *GetValue(index)* method of the *RealArray* data type to retrieve the first element of the real array *pr::MyRealArray*.

Table 26. Unary Operators

Operator	Description	Applicable Data Types
+	Positive sign for the operand, that is, a no operator.	Integer, Real
-	Negative sign for the operand	Integer, Real
!	Not operator	Boolean
~	Bitwise complement	Integer

Table 27. Binary Operators

Operator	Description	Applicable Data Types
+	Addition	Integer, Real, and String
-	Subtraction	Integer, Real
*	Multiplication	Integer, Real, Transform
/	Division	Integer, Real
%	Modulo (Remainder)	Integer, Real
= (==)	Equality	Integer, Real, String, and Boolean
!=	Not equal	Integer, Real, and Boolean
<	Less than	Integer, Real
<=	Less than or equal	Integer, Real
>	Greater than	Integer, Real

Table 27. Binary Operators (Continued)

Operator	Description	Applicable Data Types
>=	Greater than or equal	Integer, Real
&&	And	Boolean
||	Or	Boolean
&	Bitwise And	Integer
|	Bitwise Or	Integer
^	Bitwise Exclusive Or	Integer
<<	Shift left	Integer
>>	Shift right	Integer

Table 28. Miscellaneous

Operator	Description
#	Member invocation operator
[]	Index operator

Operator Precedence

Each operator has a defined precedence when compared to other operators. Precedence controls the order in which an operator is executed when compared to another operator. When an expression contains more than one operator, the execution takes place based on the precedence of the operators.

Parentheses are used for manually controlling the order of execution.

Table 29 displays the list of operators in the order of their highest precedence.

Table 29. Precedence of Operators

Operator
#, []
Unary+, -, ~,!
*, /,%
+, -
>>, <<
<=, >=, <, >
=,!=
&
^
|
&&
||

Conditional Statements

if-then-else is the conditional statement, which is used in expressions. The syntax of the conditional statement is,

```
if condition then
    true expression
else
    false expression
```

where *condition* refers to the condition to be checked, *true expression* is the expression to be evaluated if the condition is true, and *false expression* is the expression to be evaluated if the condition is false.

```
if a>10 && a<100 then
```

```
        a+1
    else
        0
```

where *a* is an expression variable. In this expression, the current value of the expression variable is incremented only if it is greater than 10 and less than 100. Otherwise the value of this expression is 0.

Multiple conditions can also be checked by using more than one **if-then-else** statement. This is called *Nested If*.

Data Types

Any value handled by an expression is of a particular data type such as Integer, Real, Boolean, String, or Color. The following are some examples where data types apply.

• Expression Variables.

• Input Properties.

• Properties of the graphic and input items.

• Parameters of expression functions.

Data types are categorized as the following.

• *Value types* as specified in Table 30.

• *Reference types* (only references to system entities) as specified in Table 31.

Table 30. Value Types

Type Name	Description	Constant Value Format Examples
Real	Floating point value and handled as a 64 bit number. This data type also supports *NaN* (Not a number) and *Infinity* values.	1.2, -2.3, 1.2E-34
Integer	64 bit signed integer Constant values of an integer can be represented in a decimal format or hexadecimal format.	12, -14, 0xFFA8
String	Set of unicode characters *\n* indicates a newline character and *\t* indicates a tab character in a string constant. ** indicates a \ character in a string constant.	"text" "firstlinetext\nsecondlinetext"
Boolean	True or False values	True, False
DateTime	Specifies a date / time value	DateTime (2008, 6,1 2, 10, 0, 0, 123) DateTime (2008, 6,1 2, 10, 0, 0) DateTime (2008, 6,1 2)
GradientStop	Specifies a color and an offset. This applies to a GradientBrush.	GradientStop (red, 0.8)
Transform	Describes a geometric transformation, which can be used to define a dynamic transformation, that is, an animation	Move (x,y) Empty
Rotation	Describes the angle of rotation or shearing applied to a graphic item	45 Shear (xAngle, yAngle)
Point	Specifies a coordinate	Point (x,y) None

Table 30. Value Types (Continued)

Type Name	Description	Constant Value Format Examples
PointList	Specifies a list of point values that controls the shape of the graphic item	"13,19 65,22 98,12.5"
Font	Describes the font. This includes the font, font style, font weight, and font size.	Font(name, size, style, weight)
Color	Specifies the color.	RGB(127,127,127)
Brush	Specifies the color for the filled area of the graphic item. The following are the functions included for this type. • SolidBrush • HatchBrush • ImageBrush • ShadeBrush • LinearGradientBrush • RadialGradientBrush • Transparent	Transparent()
Pen	Specifies the color and width of the line appearing in the graphic item. The following are the functions included for this type. • Pen • DashDotPen	Pen (Brush, Width) Empty
Image	Reference to an image resource.	

Table 30. Value Types (Continued)

Type Name	Description	Constant Value Format Examples
Rectangle	Specifies a rectangle	Rectangle (x,y,width,height) Rectangle (Size(width,height)) Rectangle (Point(x0,y0), Point(x1,y1)) Rectangle (Point (x,y), Size(width,height))
Enum	List of enumerated values.	Name of the Enum value

Variant	Data type which permits values of the following data types.
	
	   • Integer
	   • Real
	   • String
	   • Boolean
	   • DateTime
	   • DateTimeArray
	   • IntegerArray
	   • StringArray
	   • BooleanArray
	   • RealArray
		3.14
		"A string"
		12
		True
		False

BooleanArray	Specifies an array of boolean values.	"false true false"
IntegerArray	Specifies an array of integer values.	"12 13 14"
RealArray	Specifies an array of real values.	"12 14.6 3.9"
StringArray	Specifies an array of strings.	" 'string1' 'string2' "

Table 30. Value Types (Continued)

Type Name	Description	Constant Value Format Examples
DateTimeArray	Specifies an array of datetime values.	MakeDateTimeArray (DateTime (2008, 6, 12), DateTime (2007, 1, 1, 12, 0, 0))
PropertyRefArray	Specifies an array of property references.	
ViewRefArray	Specifies an array of view references.	
BrushArray	Specifies an array of brushes.	MakeBrushArray (red,white)
IntegerTuple*	Specifies a value that defines the Row and Column properties. Both properties are of Integer data type.	Tuple (0,0)

*   Feature Pack Functionality

Table 31. Reference Types

Type Name	Description	Constant Value Format
PropertyRef	Reference to an aspect object property or expression variable.	Refer to Table 68 for the syntax for giving references.
ViewReference	Reference to a view of an aspect.	
VerbReference	Reference to an aspect or object verb.	
ObjectRef	Reference to an aspect object.	

Font

The Font data type is used to define the style of text. This contains Font name, Font size, Font style, and Font weight.

The following are the different font styles:

•   Regular

- Italic

- Underline

- Strikeout

Font style is a set enumeration. It can accept a combination of values from the enumeration. For example, Italic | Underline, that is, the font style can be Italic and Underline.

For example, Font ("Tahoma", 13, Regular, Normal)

The different font weights used are Black, Bold, DemiBold, ExtraBlack, ExtraBold, ExtraLight, Heavy, Light, Medium, Normal, Regular, SemiBold, Thin, UltraBlack, UltraBold, and UltraLight.

Transform

The Transform data type contains a set of functions, which are used to animate graphic items. It can perform move, rotate at, and scale operations and combinations of all these using the * operator.

This data type supports the following functions:

- **ScaleAt (Real x, Real y, Real xPos, Real yPos)** performs a scale operation around the point xPos and yPos.

- **RotateAt (Real angle, Real xPos, Real yPos)** performs a rotation based on the angle specified, with respect to the coordinate values xPos and yPos.

- **Move (X,Y)** transforms the graphic item X distance units horizontally and Y distance units vertically.

- **Transform (D1,D2,D3,D4,D5,D6)** transforms the graphic item based on the transformation matrix defined by D1 to D6.

- **Empty** represents no transformation on the graphic item.

Do not configure static graphic item placement using the Transform functions.

Rotation

Rotation is done by using a scalar real value or using the function *Shear*. The syntax for *Shear* is,

```
Shear(X,Y)
```

where *X* is the value for horizontal angle and *Y* is the value for vertical angle. This function alters the vertical lines of the item and then the horizontal lines based on the X and Y values.

A scalar real value is applied equally to horizontal and vertical angles.

Brush

Brush values are used to paint the areas of an item or to shade the item.

The following are the different types of brushes used.

- Solid Brush

- Hatch Brush

- Shade Brush

- Image Brush

- Linear Gradient Brush

- Radial Brush

- Radial Gradient Brush

- Logical Brush

For more information on the functions, refer to Color on page 200.

Solid Brush

Solid Brush is a solid color.

Hatch Brush

Hatch Brush is used to shade a graphic item with a pattern. The syntax for Hatch Brush is,

```
HatchBrush (HatchStyle, Color1, Color2)
```

where *HatchStyle* is the type of shading to be applied, *Color1* is the color for the pattern, and *Color2* is the color for shading the item.

Shade Brush

Shade Brush is used for 3D presentation of colors for the graphic item. The syntax for Shade Brush is,

```
ShadeBrush (ShadeStyle, Color1, Color2)
```

where *ShadeStyle* is the type of shading to be applied, *Color1* and *Color2* are the colors used for shading the item.

Image Brush

Image brush uses an image to define what to draw when the brush is used. The syntax for Image Brush is,

```
ImageBrush (image, fillStyle)
```

where, *fillStyle* specifies the style of the image brush and can have values None, Stch, Tile, TileFlipX, TileFlipXY, or TileFlipY.

Linear Gradient Brush

Linear gradient brush is used for shading graphic items. A linear gradient brush is defined using gradient stops.

```
LinearGradientBrush (Angle,s1,...,sN)
```

where, *Angle* specifies the direction in which the shading is done. *s1* to *sN* are gradient stops.

A gradient stop contains a color and an offset. For example, GradientStop (Red, 0.8).

Radial Brush

Radial brush creates a brush suitable for coloring an elliptic graphic item.

```
RadialBrush (CenterX, CenterY, FocusScale, InnerColor,
OuterColor)
```

where, *InnerColor* and *OuterColor* specify the colors for shading the item, *CenterX* and *CenterY* specify the starting position to use the inner color (For example, *0.0,0.0* sets the origin at the top left corner, *1.1,1.1* sets the origin at the bottom right corner). *FocusScale* specifies the strength of the inner color.

Radial Gradient Brush

Radial gradient brush creates a brush suitable for coloring an elliptic graphics item. A radial gradient brush is defined using gradient stops.

```
RadialGradientBrush (CenterX, CenterY, OriginX, OriginY,
RadiusX, RadiusY, s1, ..., sN)
```

where, *CenterX* and *CenterY* specify the position to start the first gradient stop, *OriginX* and *OriginY* specify the position to end the last gradient stop, *RadiusX* and *RadiusY* specify the angle for the shading, *s1* to *sN* are gradient stops.

A gradient stop contains a color and an offset. For example, GradientStop (Red, 0.8).

Pen

Pen is used for controlling the properties of lines in the graphic item. Properties like color, width, and style of the lines is controlled by using Pen.

The line properties in the Polygon, PolyLine, FilledPath and other primitives is controlled by the Pen property.

The following is the syntax of the functions that yield values of Pen type.

```
Pen (brush, width)
```

where *brush* defines a color or texture used while drawing and *width* is the thickness for the line, appearing in the primitive.

```
DashDotPen (brush, width, dashstyle, dashcap, linejoin)
```

where *brush* is the color of the line, *width* is the thickness for the line, *dashstyle* is the dash style, *dashcap* is the cap style for dashes and dots, and *linejoin* is the style for the line joins.

Table 32. Dash Styles

Name	Presentation
Solid	
Dash	
Dot	
DashDot	
DashDotDot	

Table 33. Dash Caps
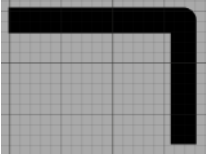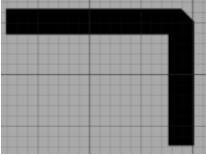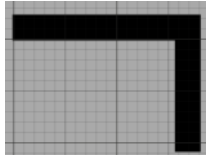
Name	Presentation
Flat	
Round	
Triangle	
Square	

Table 34. Line Joins

Name	Presentation
Round	
Bevel	
Miter	

Color

The Color type is used for assigning colors for the graphic items. The two types of color values are direct colors and logical color references.

A color can also be used for properties of data type *Brush*.

A Logical color is a color resource being identified by the color name. For example, *AlarmStateColor*.

In addition, there are named colors. Named colors are data type symbols. For example, *Red, Blue, Black*.

Direct colors are used to define the combination of Red, Green, Blue colors. The following functions are used to define direct colors.

- *RGB (R, G, B)*, where *R* is the value for Red, *G* is the value for Green, and *B* is the value for Blue. The values can vary between 0 and 255. Based on the specified values, a single color is urned.

  For example, RGB (255, 0, 0) returns the color Red.

- *ARGB (A, R, G, B)*, where *A* is the opacity factor to be given for the shade, *R* is the value for Red, *G* is the value for Green, and *B* is the value for Blue. The values for all factors can vary between 0 and 255.

  When the value for *A* is 0, the item is transparent, and when it is 255, the item is opaque.

  For example, ARGB (150, 0, 0, 255) returns the color Blue and will be partially transparent.

Table 35 shows the properties of Color data type.

Table 35. Properties of Color Data Type

Data Type	Property	Return Type	Description
Color	A	Integer	Returns the transparency factor.
	R	Integer	Returns the red component of the color.
	G	Integer	Returns the green component of the color.
	B	Integer	Returns the blue component of the color.

Real

The Real data type handles real values of 64 bits. Constant values of this data type can be in decimal format (for example, 1.234) or exponential format (for example, 1.2E-3). This data type also supports the *NaN* (Not a number) and *Infinity* values.

There are two *Infinity* values, namely, positive infinity and negative infinity. Any operation encountering a *NaN* value yields *NaN*.

The following are examples of operations that operate on or yield *Infinity* and *NaN*:

- 1/0 => Positive infinity
- -1/0 => Negative infinity
- 1/Infinity => 0
- 0/0 => NaN
- Infinity/Infinity => NaN

Expressions may yield *NaN* or *Infinity* values. This may not be a problem, however, graphics replaces the *NaN* and *Infinity* values with a neutral number (1.0) when assigned to properties of input items and graphic items. This is the default behavior and it avoids unexpected behavior of the Graphics Builder and runtime graphics.

If the values *NaN* and *Infinity* need to be handled different from this default behavior, the functions **IsNaN (Real)** and **IsInfinity (Real)** can be used to test the *NaN* and *Infinity* values respectively. This allows them to be replaced with values appropriate to the situation.

For example, create two expression variables *ev1* and *ev2*. In an expression where ev1/ev2 may evaluate to *Infinity*, the user may use an expression like the following.

```
If IsInfinity (ev1/ev2) then
   3.14
else
   ev1/ev2
```

Another way to handle *NaN* and *Inifnity* values is to turn off the visibility of a graphic item (using the **Visible** property) which would otherwise yield an erroneous presentation when *NaN* or *Infinity* values (being replaced with 1.0) are assigned to properties.

Table 36 shows the properties of the Real data type.

Table 36. Properties of Real Data Type

Data Type	Property	Return Type	Description
Real	IsInfinity	Boolean	Returns *True* if the real value is a positive or a negative infinity.
	IsNaN	Boolean	Returns *True* if the real value is not a number.
	IsNegativeInfinity	Boolean	Returns *True* if the real value is a negative infinity.
	IsPositiveInfinity	Boolean	Returns *True* if the real value is a positive infinity.
	IsNumber	Boolean	Returns *True* if the real value is a number, that is, it is not NaN, NegativeInfinity, or PositiveInfinity.

Integer

The Integer data type handles integer values of 64 bits. Integer constants can be in decimal format (for example, 1234) or hexadecimal format (for example, 0x1F07B).

This data type supports the index operator which returns *True* if the bit at the index is 1. Otherwise it returns *False*. An index of 0 represents the least significant bit. The expression 2[0] returns *False* but 2[1] returns *True*.

The following are the methods for the Integer data type:

•   ToBinaryString (Integer Start index, Integer Length, Boolean Suppress leading zeros)

This method returns a binary string. *Start Index* is the start position of the binary string (the position of the least significant bit is zero), *Length* is the length of the urned binary string. If the integer value is greater than zero and *Suppress leading zeros* is **True**, all zeros before 1 (in the first position) is removed.

For example, consider an integer with value 56 (= 0x38), that is, binary string "111000", with leading zeros suppressed, with the following expression:

```
ToBinaryString (1,3,False)
```

The result of this expression will be "100".

Consider the following expression:

```
ToBinaryString (2,3,False)
```

The result of this expression will be "110".

• Bit (Integer BitNumber)

This method returns *True* if the bit corresponding to the specified bit number (position of the least significant bit is 0) is set.

String

The String data type handles strings of unicode characters. A constant value of the String data type is enclosed within double quotes (for example, "Text").

If the string constant contains a **"** character, then it should be substituted with the **\"** combination. If the string constant contains a \ character, it should be substituted with ****.

A newline character is substituted with **\n** and a tab character is substituted with **\t**.

Table 37 shows the properties of the String data type.

Table 37. Properties of String Data Type

Data Type	Property	Return Type	Description
String	Length	Integer	Returns the number of characters in the string.

LocalizedText

The LocalizedText data type is used to handle values of properties where the value can be a string or a reference to an NLS text.

HistoryReference

The HistoryReference data type is used for referencing an aspect object property with an option to specify a specific history log.

Path

The Path data type is used for the **Filled Path** graphic item to define closed or filled shapes, multiple shapes, and curved shapes. This is accomplished by specifying a string of commands. Uppercase commands use the absolute coordinates while lowercase commands use the relative ones.

- Move command - M *startpoint*

- Line - L *endpoint*

- Horizontal line - H *endpoint*

- Vertical line - V *endpoint*

- Cubic Bezier curve - **C** *controlpoint1 controlpoint2 endpoint*

- Quadratic Bezier curve - Q *controlpoint endpoint*

- Smooth cubic Bezier curve - S *controlpoint endpoint*

- Smooth quadratic Bezier curve - T *controlpoint endpoint*

- Elliptical Arc - A *size rotationangle isLargeArcFlag sweepDirectionFlag endpoint*

- Close command - Z

For more information, refer to FilledPath on page 420.

PropertyRef

The value of a PropertyRef data type is any of the following:

- Reference to an aspect object property

  This allows a graphic item to write to the target property.

- Reference to an expression variable

This is possible if the data type of expression variable is PropertyRef or Variant compatible data type (refer to PropertyRef referencing an expression variable on page 208).

• Null (that is, no target)

Operations cannot be performed.

For information on the syntax of PropertyRef constant values, refer to Table 67 and Table 68. Table 38 describes the properties of PropertyRef data type.

PropertyRef Constants

A constant value of a PropertyRef being a reference to an aspect object property, is represented by a symbol like the following:

```
MyObject:MyAspect:PropertyName
```

Symbols representing PropertyRef constants are interped in one of following ways:

• As the property reference represented by the symbol.

• As the value of the referenced property.

The parser selects one of the above interpations based on the data type being expected at the position of the symbol.

If the expected data type is **PropertyRef**, then the value of the symbol is the property reference being represented by the symbol.

If the expected data type is Value type, then the value of the symbol is the value of the referenced property. The data type of the property is the data type of the referenced property after converting it to corresponding graphics data type. For more information on data type conversions, refer to Data Type Conversions on page 226.

If the expected data type is different from the data type of the referenced property, an attempt is made to perform an implicit data conversion. If no implicit data type conversion exists, then a type mismatch is reported to the Diagnostics Window. This situation may arise when the data type of the referenced property has been changed.

It is possible that the data type of referenced property changes after the configuration time. Since this is considered less frequent, it is required that the property reference is manually updated using the Data References window.

Retrieving property values using non constant PropertyRef

The section PropertyRef Constants describes how the value of a referenced property is accessed through a PropertyRef which has a constant value being known at the time of configuration. It is also possible to retrieve the value of referenced property when it is not the case, but this requires the usage of a property to the PropertyRef data type to inform expressions of the expected data type.

The following are examples of expression clauses which yield PropertyRef values not being known at configuration time:

*   if <condition> then

    PropertyRef1

    else

    PropertyRef2

*   <Name of input property of type PropertyRef>

*   <Name of expression variable of type PropertyRef>

*   Call to LateBoundPropertyRef function

The above expression clauses never yield the value of the referenced property because of not constituting constant values. The reason is that the data type of the referenced property is not automatically known at the time of configuration. It is possible, though, to retrieve the value of the referenced property by applying, to the PropertyRef clause, a property which defines the expected data type. The following properties retrieve the value from the referenced property:

*   **pr#Value** - yields a Variant value.

*   **pr#BoolVa**l - yields a Boolean value.

*   **pr#IntVal** - yields an Integer value.

*   **pr#RealVal** - yields a Real value.

- **pr#StringVal** - yields a String value.

Diagnostic messages are produced while applying a "value property" to a PropertyRef value which is Null. Diagnostic messages are also produced for all properties except #Value if data type of the property value cannot be converted to the required data type.

PropertyRef referencing an expression variable

In an expression, an expression variable is referenced by a symbol such as ev::MyExpVar. The following possibilities exist when the symbol is expected to yield a value of data type PropertyRef:

- If the data type of expression variable is PropertyRef, then the symbol yields a value being the value of the expression variable

- If the data type of expression variable is Variant compatible, then the symbol yields a value being the reference to the expression variable

An error is prompted if the expression variable is of any other data type.

Data types of which the values can be hosted by a Variant value, and Variant itself are said to be Variant compatible. Variant compatible data types are:

- Real and RealArray

- Integer and IntegerArray

- Boolean and BooleanArray

- String and StringArray

- DateTime and DateTimeArray

- Variant

Table 38. Properties of PropertyRef

Name	Data Type	Description	Value if PropertyRef is Null	Value when Target is Expression Variable
DataQuality	Enum	Returns *Good* if a correct value is delivered. *Uncertain* states that the subscription facility has delivered a value with an uncertain data quality. *NotPermittedToRead* may be provided when supported by the 800xA subscription facility. Values of this property can be Good, Uncertain, Bad, NotPermittedToRead, NotInitialized, or PropertyNotFound.	NotInitialized	Good
QualityDetail	Enum	Provides the quality word of the subscribed property, as delivered by the 800xA subscription facility.	0X80	0
IsGood	Boolean	Returns *True* if *DataQuality* is *Good*.	False	True
IsBad	Boolean	Returns *True* if *DataQuality* is *Bad*.	False	False
IsUncertain	Boolean	Returns *True* if *DataQuality* is *Uncertain*.	False	False
QualityDescription	String	A string describing the data quality	""	""

Table 38. Properties of PropertyRef (Continued)

Name	Data Type	Description	Value if PropertyRef is Null	Value when Target is Expression Variable
WriteAccess-Granted	Enum	Specifies if write is granted and possible reasons when it is not granted. *ObjectNotLocked* signifies that the object, which is hosting the property requires to be locked to permit writing, but is not locked. The values can be Granted, PropertyNotFound, WriteNotPermitted, NotWritable, ObjectNotLocked, or NotInitialized.	NotInitialized	Granted
IsWritable	Boolean	Returns *True* if *WriteAccess-Granted* is *Granted*.	True	True
WriteAccessGrant-edDescription	String	A string describing the write access.	""	"True"
Value	Variant	Returns the value of the referenced property.	The Empty Variant value	Depends on the value of the expression variable
IsNull	Boolean	Returns *True* for null references.	True	False
BoolVal	Boolean	Returns the value of the referenced property as Boolean.	No Value	The value of the expression variable or if not convertible to a Boolean, no value and a diagnostic message.

Table 38. Properties of PropertyRef (Continued)

Name	Data Type	Description	Value if PropertyRef is Null	Value when Target is Expression Variable
DateTimeVal	DateTime	Returns the value of the referenced property as DateTime.	No Value	The value of the expression variable as DateTime.
IntVal	Integer	Returns the value of the referenced property as Integer.	False	The value of the expression variable as Integer.
Object	ObjectRef	Returns the reference to the aspect object which has the property.	Null	Null
PresentationName	String	Returns a presentation string of the property reference.	""	""
RealVal	Real	Returns the value of the referenced property as Real.	No Value	The value of the expression variable as Real.
ReferenceSet	Boolean	Returns *False* for null references.	False	True
StringVal	String	Returns the value of the referenced property as String.	No Value	The value of the expression variable as String.
TimeStamp	DateTime	Returns the timestamp of the value which is generally the time at which the value was read depending on the data subscription facility that is used. This may be the device time or server time.	1/1/0001 12:00 AM	1/1/0001 12:00 AM

The value urned by the properties *IntVal*, *BoolVal*, *RealVal*, *StringVal*, *DateTimeVal* depends on the relation between the source data type and the requested data type. The value of the source data type is converted to the target data type type if possible. Otherwise, an error message is displayed in Diagnostics Window and the urned value is set to *No Value*, that is, the corresponding actual value is set to null value for the target type. For example, 0.0 is set for Real data type or "" is set for a String data type.

The following is the method for the PropertyRef data type:

*   ReferencePresentation (PropertyRefReferencePresentationFormatEnum format)

    This method presents a property reference based on the selected format. Available parameters are shown in Table 39.

Table 39. Available presentation formats in PropertyRef#ReferencePresentation method

Available presentation formats in PropertyRef#ReferencePresentation method	Description
Object	The object name.
Group	The aspect name.
Property	The property name.
GroupProperty	The aspect and property name.
ObjectGroup	The object and aspect name.
ObjectProperty	The object and property name.
ObjectDescription	The object description.
ObjectPath	The object path.
Short	The object, and property name.
Long	The object, aspect, and property name.

ViewReference

The value of a ViewReference data type is a reference to an aspect view. For more information on the syntax, refer to Table 68.

Table 40. Properties of ViewReference

Name	Data Type	Description
IsNull	Boolean	Returns *True* for null references.
Object	ObjectRef	Returns a reference to the object being the owner of the reference aspect view.
PresentationName	String	Returns a presentation string of the view reference.

The following is a method for the ViewReference data type:

ReferencePresentation (ViewReferencePresentationFormatEnum format)

This method presents the view reference based on the selected format. Available presentation formats are shown in Table 41.

Table 41. Available presentation formats in ViewReference#ReferencePresentation method

Selectable formats of ViewReference#ReferencePresentation method	Description
Object	The object name.
Group	The aspect name.
View	The view name.
GroupView	The aspect and view name.
ObjectGroup	The object and aspect name.
ObjectView	The object and view name.
ObjectDescription	The object description.
ObjectPath	The object path.

Table 41. Available presentation formats in ViewReference#ReferencePresentation method (Continued)

Selectable formats of ViewReference#ReferencePresentation method	Description
Short	The object and view name.
Long	The object, aspect, and view name.

VerbReference

The value of a VerbReference data type is a reference to an aspect verb or an object verb. For more information on the syntax, refer to Table 68.

Table 42. Properties of VerbReference

Name	Data Type	Description
IsNull	Boolean	Returns *True* for null references.
Object	ObjectRef	Returns a reference to the object owning the aspect which defines the referenced verb.
PresentationName	String	Returns a presentation string of the verb.

The following is a method for the VerbReference data type:

ReferencePresentation (VerbReferencePresentationFormatEnum format)

This method presents the verb reference based on the selected format. The available presentation formats are shown in Table 43.

Table 43. Available presentation formats in VerbReference#ReferencePresentation method

Name	Description for Aspect Verb	Description for Object Verb
Object	The object name.	The object name.
Group	The aspect name.	
Verb	The verb name.	The verb name.
GroupVerb	The aspect and verb name.	The verb name.
ObjectGroup	The object and aspect name.	The object name.
ObjectVerb	The object and verb name.	The object and verb name.
ObjectDescription	The object description.	The object description.
ObjectPath	The object path.	The object path.
Short	The object, aspect, and verb name.	The object and verb name.
Long	The object, aspect, and verb name.	The object and verb name.

WriteSpecification

The WriteSpecification data type is used to assign multiple values to multiple aspect object properties sequentially or as a batch transaction.

The **WriteSpecification** data type is supported by the following functions:

- SingleWrite

- BatchWrite

- SequentialWrite

SingleWrite

The syntax of this function is:

```
SingleWrite (PropertyRef Target, Variant Value)
```

This function is used to write a value to a single target.

Target specifies a property reference to which the value should be written, and *Value* is the value to be written to *Target*.

SequentialWrite

The syntax of this function is:

```
SequentialWrite (Boolean ContinueOnError,
Integer ExtraDelayBetweenWrites, WriteEntry WriteEntry1,
WriteEntry WriteEntry2, ...)
```

This function is used to write multiple values to multiple targets sequentially. Each *WriteEntry* parameter specifies a value and the target for one write operation.

Set *ContinueOnError* as **True** to continue writing even after an error during previous write operation.

ExtraDelayBetweenWrites specifies the time interval (in milliseconds) between two write entries. If this value is set to 0, the next write is started immediately after a write is reported to be ready. Set this to any other value if another time interval is required.

WriteEntry has the following syntax:

```
WriteEntry (PropertyRef Target, Variant Value)
```

Target is a reference to an aspect object property or an expression variable to which the value specified by the *Value* parameter is to be written.

BatchWrite

The syntax of this function is:

```
BatchWrite (WriteEntry WriteEntry1,
WriteEntry WriteEntry2, ...)
```

This function is used to write multiple values to multiple targets at the same time as one transaction. Each *WriteEntry* parameter specifies a value and the target for one write operation.

WriteEntry has the following syntax:

```
WriteEntry (PropertyRef Target, Variant Value)
```

Target is a reference to an aspect object property or an expression variable to which the value specified by the *Value* parameter is to be written.

Enum

The Enum data type is a collective designation for large number of data types each being defined as a collection of names. The Enum types are predefined by graphics or user defined.

For information on defining user enumerations, refer to User Enumerations on page 84.

Graphics defined enumerations comprise:

•   Enumerations that apply to parameters of expression functions.

•   Enumerations that apply to the properties of graphic items.

The following are the two types of enumerations:

•   Set enumerations

•   Non set enumerations

Set Enumerations

A value of a set enumeration is a single name or a combination of names from the enumeration. The first value of a set enumeration represents the empty set value. The empty set value is the "no option selected" value.

For example, *FontStyle* values are Regular, Italic, Strikeover, and Underline. This enumeration can accept a combination of these values. For example, the value Italic | Underline provides a font style which is both italic and underlined.

The value Regular represents the empty set value of the *FontStyle* enumeration. It cannot be combined with other set values. For example, the value, Regular | Italic evaluates to Italic.

Non set Enumerations

Value of a non set enumeration is a single name out of the enumeration.

Other Data Types

Table 44 gives the list of properties and Table 45 gives the list of methods for data types such as BooleanArray, IntegerArray, StringArray, RealArray, DateTime, DateTimeArray, and Rectangle.

Table 44. Data Type Properties

Data Type	Property	Return Type	Description
BooleanArray	Length	Integer	Returns the number of items in the array.
IntegerArray	Length	Integer	Returns the number of items in the array.
RealArray	Length	Integer	Returns the number of items in the array.
StringArray	Length	Integer	Returns the number of items in the array.

Table 44. Data Type Properties (Continued)

Data Type	Property	Return Type	Description
DateTime	AMPM	String	Returns the A.M./P.M. designator for the specified DateTime.
	Day	Integer	Returns the day of the month of specified DateTime (1-31).
	DayName	String	Returns the name of day in the specified DateTime.
	DayOfWeek	Integer	Returns the day of the week of specified DateTime (0-6).
	DayOfYear	Integer	Returns the day of the year of specified DateTime (1-366).
	Hour	Integer	Returns the hour component (0-23).
	Millisecond	Integer	Returns the millisecond component (0-999).
	Minute	Integer	Returns the minute component (0-59).
	Month	Integer	Returns the month component (1-12).
	MonthName	String	Returns the name of the month in the specified DateTime.
	Second	Integer	Returns the second component (0-59).
	ToLocal	DateTime	Returns the specified UTC time converted to local time.
	ToUtc	DateTime	Returns the specified local time converted to UTC time.
	Week	Integer	Returns the week of the year that includes the specified DateTime.
	Year	Integer	Returns the year component (1-9999).
DateTimeArray	Length	Integer	Returns the number of items in the array.

Table 44. Data Type Properties (Continued)

Data Type	Property	Return Type	Description
Rectangle	X	Real	Returns the left most position of the rectangle.
	Y	Real	Return the top most position of the rectangle.
	Width	Real	Returns the width of the rectangle.
	Height	Real	Returns the height of the rectangle.
	Left	Real	Returns the left most position of the rectangle.
	Top	Real	Return the top most position of the rectangle.
	Bottom	Real	Return the bottom most position of the rectangle.
	Right	Real	Returns the right most position of the rectangle.
	TopLeft	Point	Returns the top left position of the rectangle.
	TopRight	Point	Returns the top right position of the rectangle.
	BottomRight	Point	Returns the bottom right position of the rectangle.
	BottomLeft	Point	Returns the bottom left position of the rectangle.
IntegerTuple*	Row	Integer	Returns the Row property of the IntegerTuple.
	Column	Integer	Returns the Column property of the IntegerTuple.

Table 44. Data Type Properties (Continued)

Data Type	Property	Return Type	Description
Point*	X	Real	Returns the X property of the Point.
	Y	Real	Returns the Y property of the Point.

*   Feature Pack Functionality

Table 45. Data Type Methods

Data Type	Method	Return Type	Parameters	Description
BooleanArray	GetRange	BooleanArray	Integer index, Integer count	Returns the sub array starting from the specified position (index) upto the specified length (count),
	GetValue	Boolean	Integer index	Returns the value at the specified position (index) in the array.
	ReplaceMember	BooleanArray	Integer index, Boolean value	Returns the array with the entry at the specified position (index) replaced with *value*.
	ReplaceRange	BooleanArray	Integer index, BooleanArray value	Returns the array with the range starting at the specified position (index) replaced with the values (in value).

Table 45. Data Type Methods (Continued)

Data Type	Method	Return Type	Parameters	Description
IntegerArray	GetRange	IntegerArray	Integer index, Integer count	Returns the sub array starting from the specified position (index) upto the specified length (count),
	GetValue	Integer	Integer index	Returns the value at the specified position (index) in the array.
	ReplaceMember	IntegerArray	Integer index, Integer value	Returns the array with the entry at the specified position (index) replaced with *value*.
	ReplaceRange	IntegerArray	Integer index, IntegerArray value	Returns the array with the range starting at the specified position (index) replaced with the values (in value).
RealArray	GetRange	RealArray	Integer index, Integer count	Returns the sub array starting from the specified position (index) upto the specified length (count),
	GetValue	Real	Integer index	Returns the value at the specified position (index) in the array.
	ReplaceMember	RealArray	Integer index, Real value	Returns the array with the entry at the specified position (index) replaced with *value*.
	ReplaceRange	RealArray	Integer index, RealArray value	Returns the array with the range starting at the specified position (index) replaced with the values (in value).

Table 45. Data Type Methods (Continued)

Data Type	Method	Return Type	Parameters	Description
StringArray	GetRange	StringArray	Integer index, Integer count	Returns the sub array starting from the specified position (index) upto the specified length (count),
	GetValue	String	Integer index	Returns the value at the specified position (index) in the array.
	ReplaceMember	StringArray	Integer index, String value	Returns the array with the entry at the specified position (index) replaced with *value*.
	ReplaceRange	StringArray	Integer index, StringArray value	Returns the array with the range starting at the specified position (index) replaced with the values (in value).

Table 45. Data Type Methods (Continued)

Data Type	Method	Return Type	Parameters	Description
DateTimeArray	GetRange	DateTimeArray	Integer index, Integer count	Returns the sub array starting at the specified (zero based) position with the specified length.
	GetValue	DateTime	Integer index	Returns the value at specified (zero based) position in the array.
	ReplaceMember	DateTimeArray	Integer index, DateTime value	Returns the array with the entry at the specified position (index) replaced with *value*.
	ReplaceRange	DateTimeArray	Integer index, DateTimeArray value	Returns the array with the range starting at the specified position (index) replaced with the values (in value).
Rectangle	IntersectsWith	Bool	Rectangle rectangle	Returns *True* if the given rectangle intersects with the rectangle.

Data Type Conversions

Data Type Conversion is the process of converting the data type of an operand to another data type. Data type conversion is used for the following:

- While performing binary operations for the operands that are not of the same data type.

- While executing functions which require operands of a specific data type.

- To adjust the Return type from an expression to be consistent with the property to which the expression is assigned.

Data type conversions are implicit or explicit.

Implicit conversions are applied automatically without having to be explicitly requested. Table 46 gives the list of implicit type conversions.

Table 46. Implicit Type Conversions

Target Data Type	Source Data Type
Real	DateTime
Real	Integer
Real	Float
RealArray	DateTimeArray
DateTimeArray	RealArray
DateTime	Real
Rotation	Real
Rotation	Integer
PointList	String
IntegerArray	String
RealArray	String
StringArray	String
BooleanArray	String
Brush	Color
Pen	Color
Pen	Brush
Variant	Integer
Variant	Real
Variant	String
Variant	Boolean
Variant	DateTime

Table 46. Implicit Type Conversions (Continued)

Target Data Type	Source Data Type
Variant	IntegerArray
Variant	RealArray
Variant	StringArray
Variant	BooleanArray
Variant	DateTimeArray
Variant	LocalizedText
String	LocalizedText
Float	Integer
Float	Real
LocalizedText	String
HistoryRef	PropertyRef
Path	String

Explicit conversions includes the functions, which should be called for performing the type conversion. For example, Integer (Real operand). Table 47 gives the list of explicit type conversions.

Table 47. Explicit Type Conversions

Target Data Type	Source Data Type
Integer	Real
Integer	String
Integer	Variant
Integer	Float
Real	String

Table 47. Explicit Type Conversions (Continued)

Target Data Type	Source Data Type
Real	Integer
Real	Variant
Boolean	Variant
String	Variant
String	Real
String	Integer
String	Boolean
String	Float
String	DateTime
String	StringArray
String	IntegerArray
String	BooleanArray
String	RealArray
String	DateTimeArray
RealArray	Variant
RealArray	String
IntegerArray	Variant
IntegerArray	String
BooleanArray	Variant
BooleanArray	String
StringArray	Variant
StringArray	String
PointList	String

Table 47. Explicit Type Conversions (Continued)

Target Data Type	Source Data Type
DateTime	Variant
DateTimeArray	Variant

Relation between Aspect Property types and Graphics Data Types

Properties of aspect objects can be of a host of data types as described in Table 48. Graphics supports all possible property data types, except Object[] which is supported for smaller set of data types.

Property data types are presented by their .Net names in the Graphics Builder, in user interfaces such as the Property browser and Test Data dialog.

Table 48 describes the graphics data type that is used for each property data type. This table also provides an alternate name for the property data types because the .Net names or characterizations might be used in different contexts.

Table 48. Aspect Object Property data types and corresponding Graphics Data types

Data Types of Aspect Object Properties			Graphics Data types
.Net Name	Alternate Name	Description	
Boolean	BOOL	Boolean value type	Boolean
String	BSTR	String type	String
DateTime	DateTime	DateTime type	DateTime

Table 48. Aspect Object Property data types and corresponding Graphics Data types (Continued)

Data Types of Aspect Object Properties			Graphics Data types
.Net Name	**Alternate Name**	**Description**	
SByte	I1	8 bit signed integer	Integer
Int16	I2	16 bit signed integer	
Int32	I4	32 bit signed integer	
Byte	UI1	8 bit unsigned integer	
UInt16	UI2	16 bit unsigned integer	
UInt32	UI4	32 bit unsigned integer	
Single	R4	32 bit floating point number	Real
Double	R8	64 bit floating point number	
Object	Variant	Flexible type	Variant
Boolean[]	Array of BOOL values	Array of boolean value	BooleanArray
String[]	Array of BSTR values	Array of strings	StringArray
DateTime[]	Array of DateTime values	Array of DateTime values	DateTimeArray

Table 48. Aspect Object Property data types and corresponding Graphics Data types (Continued)

Data Types of Aspect Object Properties			Graphics Data types
.Net Name	**Alternate Name**	**Description**	
SByte[]	Array of I1 values	Array of 8 bit signed integers	IntegerArray
Int16[]	Array of I2 values	Array of 16 bit signed integers	
Int32[]	Array of I4 values	Array of 32 bit signed integers	
Byte[]	Array of UI1 values	Array of 8 bit unsigned integers	
UInt16[]	Array of UI2 values	Array of 16 bit unsigned integers	
UInt32[]	Array of UI4 values	Array of 32 bit unsigned integers	
Single[]	Array of R4 values	Array of 32 bit float-ing point numbers	Real
Double[]	Array of R8 values	Array of 64 bit float-ing point numbers	
Object[]	Array of Variant val-ues	Array of flexible type values	Not supported

Expression Functions

The following tables describe the functions that can be called from expressions.

Table 49. Math Functions

Function	Return Type	Parameters	Description
Abs	Integer	Integer value	Evaluates to the absolute of *value*.
Abs	Real	Real value	Evaluates to the absolute of *value*.
Log	Real	Real value	Evaluates to the natural (base e) logarithm of *value*.
Log	Real	Real value, Real base	Evaluates to the logarithm of *value* in the base *base*.
Log10	Real	Real value	Evaluates to the base 10 logarithm of *value*.
Truncate	Real	Real value	Returns the integer part of *value*.
Ceiling	Real	Real value	Evaluates to the smallest integer greater than or equal to *value*.
Floor	Real	Real value	Returns the largest integer less than or equal to *value*.
Sqrt	Real	Real value	Evaluates to the square root of *value*.
Max	Real	Real value1, Real value2	Returns the value largest among *value1* and *value2*.
Max	Integer	Integer value1, Integer value2	Returns the value largest among *value1* and *value2*.
Min	Real	Real value1, Real value2	Returns the value smallest among *value1* and *value2*.

Table 49. Math Functions (Continued)

Function	Return Type	Parameters	Description
Min	Integer	Integer value1, Integer value2	Returns the value smallest among *value1* and *value2*.
Exp	Real	Real power	Evaluates *e* raised to *power*.
Pow	Real	Real value, Real power	Evaluates to the *value* raised to *power*.
Remainder	Real	Real dividend, Real divisor	Returns the remainder resulting from the division of *dividend* by *divisor*.
Round	Real	Real value, Real digits, Mid-pointrounding mode	Rounds the specified number to the specified precision. *mode* specifies the way to round the value if it is between two other numbers. *AwayFromZero* rounds the specified number towards a nearest number that is away from zero. *ToEven* rounds the specified number towards the nearest even number.
Sign	Integer	Integer value	Returns *-1* if *value* is less than zero. Returns *0* if *value* is equal to 0. Returns *1* if *value* is greater than 0.

Table 49. Math Functions (Continued)

Function	Return Type	Parameters	Description
Sign	Integer	Real value	Returns *-1* if *value* is less than zero. Returns *0* if *value* is equal to 0. Returns *1* if *value* is greater than 0.
ClipToInteger	Integer	Real value	Returns the clipped (saturated) integer from Real value. If the real value fits into the integer, that value is urned. Otherwise maximum or minimum value of integer is urned.

Table 50. String Functions

Function	Return Type	Parameters	Description
AsciiArrayToString	String	IntegerArray array, Integer noOfChars, Integer noOfChars-PerInt	Converts an integer array of ascii codes into a string. It generates an error if *noOf-CharsPerInt* is less than the length of the *array* parameter times the *noOfCharsPerInt*.
CreateFormatString	String	Integer noOfDeci-mals	Creates a format string to for-mat real values. For more infor-mation, refer to Format on page 253.
TrimEnd	String	String stringValue	Returns *stringValue* after removing the trailing white-space characters.
TrimStart	String	String stringValue	Returns *stringValue* after removing the leading white-space characters.

Table 50. String Functions (Continued)

Function	Return Type	Parameters	Description
Trim	String	String stringValue	Returns *stringValue* after removing all the leading and trailing white-space characters.
Substring	String	String stringValue, Integer index, Integer count	Returns the substring from *stringValue*, based on the start position (*index*) and length (*count*).
LowerCase	String	String stringValue	Converts *stringValue* to lower case.
UpperCase	String	String stringValue	Converts *stringValue* to upper case.
Length	Integer	String stringValue	Returns the number of characters in *stringValue*.
LastIndexOf	Integer	String stringValue, String subString	Returns the index of the last occurrence of *subString* in *stringValue*. If the *subString* is not found in *stringValue*, -1 is urned.
IndexOf	Integer	String stringValue, String subString	Returns the index of first occurrence of the *subString* in *stringValue*. If substring does not exist in the string, -1 is urned.
IndexOf	Integer	Integer startIndex, String stringValue, String subString	Returns the index of first occurrence of the *subString* in *stringValue*. The search starts at a specified character position. If *subString* is not found in *stringValue* or if *startIndex* is out of range, -1 is urned.

Table 50. String Functions (Continued)

Function	Return Type	Parameters	Description
Replace	String	String stringValue, String oldValue, String newValue	Replaces all occurrences of the *oldValue* in *stringValue* with *newValue*.
IsMatchRE	String	String stringValue, String regularEx-pression	Indicates whether *regularExpression* finds a match in *stringValue*. For more information on regular expression, refer to Table 60.
SplitRE	String	String stringValue, String regularEx-pression	Splits *stringValue* into an array of substrings at positions defined by *regularExpression* match. For more information on regular expression, refer to Table 60.
ReplaceRE	String	String stringValue, String regularEx-pression, String newValue	Within a specified *stringValue*, replaces strings that match a *regularExpression* pattern with the *newValue* replacement string. For more information on regular expression, refer to Table 60.
Format	String	String format, Vari-ant arg0,.., Variant argX)	For more information, refer to Format on page 253.

Table 51. Trigonometric Functions

Function	Return Type	Parameters	Description
Acos	Real	Real cosine	Evaluates to the angle (in degrees), for which the cosine is *cosine*.
Cos	Real	Real degrees	Evaluates to the cosine of the angle *degrees*.
Cosh	Real	Real degrees	Evaluates to the hyperbolic cosine of the angle *degrees*.
Asin	Real	Real sine	Evaluates to the angle (in degrees), for the sine *sine*.
Sin	Real	Real degrees	Evaluates to the sine of the angle *degrees.*
Sinh	Real	Real degrees	Evaluates to the hyperbolic sine of the angle *degrees*.
Atan	Real	Real tangent	Evaluates to the angle (in degrees), for the tangent *tangent*.
Atan2	Real	Real x, Real y	Evaluates to the angle (in degrees), for which the tangent is the quotient of the specified values.
Tan	Real	Real degrees	Evaluates to the tangent of the angle *degrees*.
Tanh	Real	Real degrees	Evaluates to the hyperbolic tangent of the angle *degrees*.

Table 52. Array Functions

Function	Return Type	Parameters	Description
MakeBooleanArray	BooleanArray	Boolean bool1, Boolean bool2,...., Boolean boolN	Creates a boolean array from the specified boolean values.
MapBooleanArrays	BooleanArray	BooleanArray selectionArray, IntegerArray startIndices, IntegerArray endIndices	This function Returns a boolean array based the indices set. If the first bit is set in the *selectionArray*, all bits starting at the first startIndex and ending at the first endIndex is set in the resulting array. If the fourth bit is set in the *selectionArray*, all bits starting at the fourth startIndex and ending at the fourth endIndex is set in the resulting array.
UnMapBooleanArrays	BooleanArray	BooleanArray selectionArray, IntegerArray startIndices, IntegerArray endIndices	This function Returns a boolean array based on the indices set. If the second bit is set in the *selectionArray*, the first startIndex is 1, and the first endIndex is 2, the first bit is set in the resulting array. If the seventh bit is set in the selectionArray, the third startIndex is 5, and the third endIndex is 8, the third bit is set in the resulting array.
MakeIntegerArray	IntegerArray	Integer int1, Integer int2,..., Integer intN	Creates an integer array from the specified integer values.

Table 52. Array Functions (Continued)

Function	Return Type	Parameters	Description
StringToAsciiArray	IntegerArray	String string, Integer noOfCharsPerInt	Converts the *string* into an integer array of ascii codes.
MakeStringArray	StringArray	String str1, String str2,..., String strN	Creates a string array from the specified string values.
MakeRealArray	RealArray	Real real1, Real real2,..., Real realN	Creates a real array from the specified real values.
MakeDateTimeArray	DateTimeArray	DateTime dateTime1,..., DateTime dateTimeN	Creates a DateTime array from an arbitrary number of DateTimes.
LimitArray*	LimitArray	LimitValueType limit0, LimitValueType limit1,..., LimitValueType limitN	Creates a Limit array from the specified number of limits. Refer to Limit* in Table 59 for the syntax on specifying the limits.

* Feature Pack Functionality

Table 53. Late Binding Functions

Function	Return Type	Parameters	Description
LateBoundVerbRef	VerbReference	String objectPath, String aspectSpec, String verbName, Boolean unique	Returns a reference to a verb found based on call parameters, or null. *aspectSpec* may be left empty for an object verb but not for an aspect verb.
LateBoundObjectRef	ObjectRef	String objectPath, Boolean unique	Returns a reference to an object based on object-Path, or null. Returns null if no object is found, or if several objects are found based on *object-Path* when *unique* is *True*. One object is urned arbitrarily if unique is *False*.
LateBoundViewRefArray	ViewRefArray	String objectPath, String aspectSpec, String viewName	Returns an array of reference that exists, based on the call parameters, or an empty array. Setting *viewName* to an empty string Returns the default view for the existing aspect.
LateBoundPropertyRef	PropertyRef	String objectPath, String aspectSpec, String propName, Boolean unique, Integer updateRate	Returns reference to a property found based on the specified parameters, or null. The function also activates a subscription to property by the rate of *updateRate*. *aspectSpec* may be left empty on all aspects of the object.

Table 53. Late Binding Functions (Continued)

Function	Return Type	Parameters	Description
LateBoundViewReference	ViewReference	String objectPath, String aspectSpec, String viewName, Boolean unique	Returns a view reference that exists, based on the call parameters, or null. Setting *viewName* to an empty string Returns the default view for the existing aspect.
LateBoundPropertyRefArray	PropertyRefArray	String objectPath, String aspectSpec, String propName, Integer updateRate	Returns an array of references to properties found based on the call parameters, or an empty array.
NLSTextFromIdent	String	String ident, String groupName, String fallBackStr	Returns the NLS text that exist, based on the parameters. *fallbackStr* is urned when an NLS text cannot be uniquely identified based on the parameters and before resolving is completed. *ident* specifies the resource ID and *groupName* specifies the object name.

For more information on the late binding functions, refer to

Table 54. Color Functions

Function	Return Type	Parameters	Description
HighestContrast	Color	Color backColor, Color currentFore-Color, Color replaceForeColor1, Color replaceForeColor2	Returns the foreground color with the highest contrast to the background color. If the foreground color gives the sufficient contrast, it will be urned. Else replaceforecolor1 or replaceforecolor1, whichever has the highest contract will be urned.
HighestContrast	Color	Color backColor	Returns White or black, whichever has the highest contrast to *backColor*.
HighestContrast	Color	Color backColor, Color currentFore-Color	Returns the foreground color with the highest contrast to the background color. If the foreground color gives the sufficient contrast, it will be urned. Else, White or Black, which-ever has the highest contrast will be urned.
LogicalColorFromName	Color	String colorName, String groupName, Color fallbackColor	Returns color of the logical color that exists, based on *colorName* and *groupName*. The color group name may be replaced with the empty string if *colorName* is unique. *Fall-backColor* is urned if a logical color is not uniquely found before the resolving process is completed.

Table 54. Color Functions (Continued)

Function	Return Type	Parameters	Description
Brighten	Color	Color origColor, Real percent-Brighter	Changes the color by making it brighter to the extent possible.
Darken	Color	Color OrigColor, Real percentDarker	Changes the color by making it darker to the extent possible.
RGB	Color	Integer Red, Integer Green, Integer Blue	Evaluates to a color.
ARGB	Color	Integer alpha, Integer Red, Integer Green, Integer Blue	Evaluates to an alpha transparent color.
Color	Color	String color	Evaluates to a color from the specified string of hexadecimal values, for example, "#RRGGBB", "0xAARRGGBB".

For more information on the color data type, refer to Color on page 200.

Table 55. Brush Functions

Function	Return Type	Parameters	Description
RadialBrush	Brush	Real centerX, Real centerY, Real focusScale, Color innerColor, Color outerColor	Creates a brush suitable for coloring an elliptical graphic item.
HatchBrush	Brush	Pattern p, Color color1, Color color2	Evaluates to a brush with pattern.

Table 55. Brush Functions (Continued)

Function	Return Type	Parameters	Description
ImageBrush	Brush	Image imageRef, ImageBrushStyle fillStyle	Evaluates to an image brush. *fillStyle* specifies the style of the image brush.
LinearGradientBrush	Brush	Real angle, GradientStop s1,..., GradientStop sN	Generates a brush required for shading a graphic item, which is similar to a rectangle or a cone.
RadialGradientBrush	Brush	Real centerX, Real centerY, Real originX, Real originY, Real radiusX, Real radiusY, GradientStop s1,..., GradientStop sN	Creates a brush for coloring an elliptical item.
Transparent	Brush		Evaluates to a brush with transparent color.

For more information on the brush data type, refer to Brush on page 196.

Table 56. Pen Functions

Function	Return Type	Parameters	Description
Pen	Pen	Brush brush, Real width	Evaluates to a pen where *brush* specifies the color or texture of the pen and *width* is its width.
DashDotPen	Pen	Brush brush, Real width, dashStyle, dashCap, lineJoin	Evaluates to a pen with the color from the brush, and with the specified width, cap, style, and line joining style. For more information, refer to Table 32, Table 33, and Table 34.

For more information on the pen data type, refer to Pen on page 198.

Table 57. Transform Functions

Function	Return Type	Parameters	Description
Move	Transform	Real x, Real y	Evaluates to a transform being a translation in the x and/or y dimensions.
Scale	Transform	Real xFactor, Real yFactor, Real xPos, Real yPos	Evaluates to a transformation that describes a scale operation around the point xPos and yPos.
RotateAt	Transform	Real angle, Real xPos, Real yPos	Evaluates to a transform that describes a clockwise rotation of *angle* degrees around point *xPos*, *yPos*.
Transform	Transform	Real d1, Real d2, Real d3, Real d4, Real d5, Real d6	Evaluates to a transform.

Table 58. WriteSpecification Functions

Function	Return Type	Parameters	Description
SingleWrite	WriteSpecification	PropertyRef Target, Variant Value	For more information, refer to WriteSpecification on page 216.
SequentialWrite	WriteSpecification	Boolean Continue-ueOnError, Integer ExtraDelayBe-tweenWrites, WriteEntry WriteEntry1, WriteEntry WriteEntry2, ...	For more information, refer to WriteSpecification on page 216.
BatchWrite	WriteSpecification	Integer InitialDelay, WriteEntry WriteEntry1, WriteEntry WriteEntry2, ...	For more information, refer to WriteSpecification on page 216.

Table 59. Miscellaneous Functions

Function	Return Type	Parameters	Description
Shear	Rotation	Real xAngle, Real yAngle	Evaluates to a value to be applied to the Rotation property of the graphic item. For more information, refer to Rotation on page 196.
Point	Point	Real x, Real y	Evaluates to a point.
PointList	PointList	Point point1, Point point2,......Point pointN	Evaluates to a point list
ScaledPointList	PointList	PointList pointList, Real scaleX, Real scaleY	Scales all points in the *pointList* according to *scaleX* and *scaleY*.
Font	Font	String FontFamily, Real size, FontStyle style, FontWeight weight	Defines a font.
Intersection	Rectangle	Rectangle rectangle1, Rectange rectangle2	Returns the intersection between rectangle 1 and rectangle 2. Empty rectangle is urned if intersection is not possible.
NamedValue	NamedValue	String name, String value	Defines a value used by the aspect view invoker to transfer named values to the invoked aspect.
ItemEntry	ItemEntry	String inputItem-Name, String pre-sentationString	Defines a value that is used to set entry values of popup menu and similar items.

Table 59. Miscellaneous Functions (Continued)

Function	Return Type	Parameters	Description
CurrentDisplayVie-wReference	ViewReference	-	Returns the "view reference" for the graphic display in which this function is called. Is only available in object aware aspects.
CurrentViewReference	ViewReference	-	Returns the "view reference" for the graphic element in which this function is called. Is only available in object aware aspects.
IsInputEnabled	Boolean	-	Returns *True* if input is enabled, that is, this element and all its parents have the **EnableInput** property set to *True*. It is only available in object aware aspects.
FocusMovement	KeyboardNavigation	FocusDirectionType direction, Boolean nextOnApply	Allows the configuration of direction and focus handling after an apply operation.
SelectedView	Integer	-	Used in a multiple view element to find the selected view.
BuilderLevel	Integer	-	Returns a negative value at runtime. In the Graphics Builder, BuilderLevel Returns a value 0 if called in the aspect being edited. Returns 1 in the first level element instances. Returns 2 in the second level element instances and so on.
MultiSelection	MultiSelection	MultiSelection (Item-Content ItemContent(),...)	For more information, refer to MultiSelection.

Table 59. Miscellaneous Functions (Continued)

Function	Return Type	Parameters	Description
ItemContent	ItemContent	ItemContent(PropertyRef Target, Variant Value, String PresentationName, ItemStatus Status)	For more information, refer to ItemContent.
Limit*	LimitValueType	Real value, LimitType Type, Pen pen, Brush brush	Defines the upper limit or lower limit value for a trend with the color specified in *pen* and *brush*. Refer to Table 56 for Pen functions and Table 55 for Brush functions.
NavigationHistoryPosition*	Integer	ViewReference viewreference, String scope	Returns the index of *viewReference* in the history list of the aspect views that have been shown in the area specified by the reference target *scope*. The *scope* set to an empty string selects the area where the executing aspect is shown.
NumberOfLogicalScreens*	Integer		Returns the number of logical screens, that is, the number of screens configured in the **WorkplaceLayout** aspect for the workplace in which the graphic aspect is invoked.
GetScreenArraySize*	IntegerTuple		Returns the number of rows and columns of the physical screens of the workstation.
UserFullName*	String		Returns the full name of the Workplace user. The Workplace user can be the user logged in to the system or a logover user if a logover has been performed.

Table 59. Miscellaneous Functions (Continued)

Function	Return Type	Parameters	Description
UserAccountName*	String		Returns the account name of the Workplace user. The Workplace user can be the user logged in to the system or a logover user if a logover has been performed.
UserRoles*	StringArray		Returns an array of user roles that the Workplace user has. The Workplace user can be the user logged in to the system or a logover user if a logover has been performed.
IsMarked*	Boolean		This should be used by elements that implement marking by them-selves, rather than using the _**MouseOver** variable. **IsMarked** Returns *True* also when the element is not the mouse event receiver, but the mouse event receiver operates in ain Object Awareness mode. For more information on ain object awareness, refer to Retain Object Awareness on page 162.

*   Feature Pack Functionality

Regular Expressions

A regular expression is a sequence of characters for describing a search pattern. All characters match exactly the same character except for the special characters | * ? + ( ) { } [ ] ^ $ \. These characters must be preceded by **** to refer to the characters themselves. Table 60 describes a few special characters.

Table 60. Special Characters for regular expressions

Character	Description
.	Matches any single character except a line break.
*	Matches zero or more occurrences of the preceding clause, and forms the possible matches.
?	Matches zero or one occurrence of the preceding clause.
+	Matches zero or more occurrences of the preceding clause.
[]	Matches any one of the characters in the []. To specify a range of characters, list the starting and ending character separated by a dash (-). For example, [a-z].
[^]	Matches any character that is not in the set of characters that follows the ^.
^	Anchors the match string to the beginning of a line.
$	Anchors the match string to the end of a line.
|	Matches the clause before the OR (|) symbol or after this symbol.
()	Surrounds a construct that is interped as one clause.

The regular expression syntax used is Microsoft .Net Framework regular expression. For more information on .Net Framework regular expression, refer to *http://msdn.microsoft.com.*

Consider the following examples.

Example 1:

^[0-9]+$

This matches a string that consists solely of one or more digits, but no other characters.

Example 2:

((([0-9]+\.[0-9]*)|([0-9]*\.[0-9]+)|([0-9]+))

This matches any real number, such as 92, 9.4, and .3 in a string.

Example 3:

[0-9a-fA-F]+

This matches any hexadecimal number, such as 9e and AF in a string.

Format

The Format function is used for formatting the text. The syntax for the function is,

```
string Format (String format, Variant arg0,.., Variant argX)
```

where *format* is the format of the text.

For example:

```
Format("Values are {0} and {1}", exprVar1, exprVar2)
```

where, *exprVar1* and *exprVar2* are expression variables and should be variant compatible.

For more information on composite formatting, refer to *http://msdn.microsoft.com*.

Optional parameters

Optional parameters, *2 – N* are of data type *Variant*. The Format function supports optional parameters of data types for which implicit conversions to *Variant* (such as Real, Integer, Boolean, String) is possible.

Overview of format parameter

The format string may contain one or more "format items" where each format item has the following syntax:

```
{index [,alignment][:formatString]}
```

The matching braces "{" and "}" are required. The mandatory *index* component which is also called parameter specifier, is a number starting from 0 that identifies a corresponding *arg0*, *arg1*, … in the list of arguments.

The optional *alignment* component is a signed integer indicating the preferred formatted field width. If this value is less than the length of the formatted string, alignment is ignored and the length of the formatted string is used as the field width. The formatted data in the field is right-aligned if *alignment* is positive and left-aligned if *alignment* is negative. If padding is necessary, white space is used.

The optional *formatString* component is a format string that is appropriate for the type of parameter being formatted. *formatString* may support both standard and custom numeric formats, for numeric values.

Standard numeric formats

The *format* string takes the form of *Axx* for standard numeric formats.

A is the alphabetic character called the Format Specifier, and *xx* is the optional integer called Precision Specifier. Table 61 describes the format strings.

Custom numeric formats

The following is an example of a custom numeric format.

```
Format("{0:#00.000", myRealSymbol)
```

myRealSymbol is formatted such that atleast two digits are presented to the left of the decimal point and three decimals are always presented. Padding zeros are used if necessary, to the left of the decimal point. '.' represents the decimal point which is presented based on the regional settings of the executing machine.

Table 61. Format Strings

Format String	Description
C or c	Currency
D or d	Decimal, only for the integer data type. The precision specifier indicates the minimum number of digits required in the resulting string. If required, the number is padded with zeros to the left to produce the number of digits given by the precision specifier.
E or e	The number is converted to a string of the form "-d.ddd…E+ddd" or "-d.ddd…e+ddd", where each 'd' indicates a digit (0-9). The precision specifier indicates the number of digits required after the decimal point.
F or f	The number is converted to a string of the form "-ddd.ddd…". The precision specifier indicates the desired number of decimal places.
N or n	The number is converted to a string of the form "-d,ddd,ddd.ddd…", where "-" indicates a negative number symbol if required, "d" indicates a digit (0-9), "," indicates a thousand separator between number groups, and "." indicates a decimal point symbol. The precision specifier indicates the desired number of decimal places.
X or x	This is supported only for Integer. The number is converted to a string of hexadecimal digits. Use "X" for capital A – F. The precision specifier indicates the minimum number of digits required in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier.

FormatReal

The syntax of FormatReal is,

```
string FormatReal (Real value, Integer noOfDecimals)
```

This function is used for formatting a real value, when the number of decimals is determined by a variable value. The number of decimals appearing to the right of the decimal point is determined by the *noOfDecimals* parameter.

noOfDecimals can also be a negative number. This reduces the precession of the number by replacing the digits with 0. This applies for large numbers.

Consider the following examples:

FormatReal (0.1234,2) returns *0.12*.

FormatReal(123456.3,-2) returns *123500*.

FormatDateTime

The syntax of FormatDateTime is,

```
string FormatDateTime (String formatString, String langcult,
DateTime datetime)
```

where *formatString* determines the formatting of the text, and *langcult* is the language culture (for example, en-US refers to US English). Refer Table 62 for examples of language and culture strings.

For example, *FormatDateTime("F", "en-US", DateTime(2008, 11, 18)).* which will evaluate to "Tuesday, November 18, 2008 12:00:00 AM".

Table 62 describes some examples of language and culture strings.

Table 62. Examples of Language and Culture Strings

Language and Culture String	Description
en-US	English, USA
en-AU	English, Australia
Fr-FR	French, France
de-DE	German, Germany

Table 62. Examples of Language and Culture Strings (Continued)

Language and Culture String	Description
ru-RU	Russian, Russia
sv-SE	Swedish, Sweden

Table 63 describes the format of presentation.

Table 63. Format Strings

Format String	Description
"d"	Short date
"D"	Long date
"f"	Full date / time (short time)
"F"	Full date / time (long time)
"g"	General date / time (short time)
"G"	General date / time (long time)
"t"	Short time
"T"	Long time

FormatTime

The syntax of FormatTime is,

`string FormatTime (String formatString, Integer milliSeconds)`

where *formatString* specifies the time unit used in the presentation of the value.

For example, consider a time value of 11122233344 milliseconds.

  – *FormatTime (minute, 111222333444)*

    returns 128Days 17Hours 30Minutes

– *FormatTime (ms, 111222333444)* returns *128d 17h 30m 33s 344ms*

Table 64 describes the different format strings.

Table 64. Format Strings

Format String	Short Identifiers	Long Identifiers
Year	y	Year
Month	mo	Month
Day	d	Day
Hour	h	Hour
Minute	m	Minute
Second	s	Second
Millisecond	ms	Millisecond

Functions for Late Binding

System entities which are not possible to bind to, during configuration time using Graphics Builder can be accessed through late binding expression functions. These functions locate entities based on names.

Late binding functions are single reference or array reference functions.

Single reference functions include:

- LateBoundVerbRef

- LateBoundObjectRef

- LateBoundPropertyRef

- LateBoundViewRef

Array reference functions include:

- LateBoundPropertyRefArray

- LateBoundViewRefArray

Parameters for these functions are ObjectPath, AspectSpecifier, and DataEntity Name. The functions LateBoundPropertyRef and LateBoundViewRef also have a parameter called Unique.

Following are late binding functions that return resource values.

• LogicalColorFromName

• NLSTextFromIdent

For more information on the late binding functions, refer to Expression Functions on page 233.

ObjectPath parameter is a string that refers to the target object, *AspectSpecifier* is the name of the aspect or "" (empty string), *DataEntityName* is the name of the data entity being referred to. In *AspectSpecifier*, search is done with respect to the name specified or it is searched on all objects if this is specified as "".

Single reference functions return one value. All single reference functions possess the following property.

```
Boolean unique
```

This parameter allows the user to decide whether the lookup must yield a uniquely found entity. The function returns a null value if exactly one entity is not found based on the function parameters and if *Unique* is set to *True*.

The function returns arbitrarily one of several candidates being found, if *Unique* is set to *False*.

Table 65 shows some examples of values for object path parameters. In the *Example Value* column, *A, B, C* designate object names. In the *Description* column, *A, B, C* should be seen as objects with names *A, B, C* respectively.

Table 65. Examples of object path parameters

Example Value	Description
.	Specifies the invocation object of the graphic element performing the late binding.
./A	Find *A* in any structure at a distance below the invocation object.

Table 65. Examples of object path parameters (Continued)

Example Value	Description
./A/B	First find *A* in any structure at any distance below the invocation object and then find *B* in any structure at any distance below *A*.
./[Direct]A	Find *A* in any structure directly below the invocation object.
./[Direct][Functional Structure]A	Find *A* in **Functional Structure** directly below the invocation object.
./ [Functional Structure]A / [Control Structure]B	First find *A* in **Functional Structure** below the invocation object and then find *B* in **Control Structure** below *A*. *A* must exist in **Functional** and **Control Structure**.
./[Functional Structure]A/[]B	First find *A* in **Functional Structure** below the invocation object and then find *B* in any structure below *A*.
./[Direct][UP]/*	Find a parent object in any structure.
./[Direct][UP]/*/*	Find the parent object of the current parent object.
./[UP]A/B[DOWN]C	Search the structures appearing above the invocation object till *A* is found, continue the search upwards until *B* is found, and continue the search downwards until *C* is found.
A/B	Find *A* anywhere in the conce reference scope and then find *B* in any structures appearing below *A*.
{object id}/B	Find the object containing the specified object id and then find *B* in the structures appearing after the searched object.
[Functional Structure]A/*	Find all objects in **Functional Structure** appearing below *A*.

Expression Symbols

Expression symbols can be Data type symbols or Entity symbols. Data type symbols are Enum values, named color values (such as Red), values such as NaN and Infinity (for Real), and null.

A common trait is that both categories may need quoting (Enum values may contain space). Prefixes are never used for data type symbols.

The remaining sections describe only entity symbols. The entity symbols are referred to as symbols.

Expressions contain references to the following.

- Graphic entities such as input properties and expression variables

- Data entities

- Resources

The nature of data and resource references are described in Reference Status on page 143.

These references are represented in the form of symbols in the presentation of an expression. Symbol is a text string having a specific syntax. Following are examples of symbols.

- *ip::myProp* refers to an input property.

- *ev::myExpVar* which refers to an expression variable.

- *lc::GeneralColors:static* which refers to a logical color.

- *pr::MixerMotor:ControlConnection:Speed* which refers to an aspect object property.

Symbol Variations

The syntax for a symbol consists of optional and compulsory parts. For example, the following is the syntax of a resource reference.

```
[<symbol prefix>::][<Name of the resource group>:]<Name of the
resource>
```

In this example, the parts that is mentioned within [and] are optional.

For a reference to a logical color, the following variations exist.

- ColorName

- lc::ColorName

- ColorGroupName:ColorName

- lc::ColorGroupName:ColorName

where, *lc* is the prefix for logical color.

Symbol Syntax

This section describes the syntax to be used for symbols in expressions.

- Table 66 defines the prefixes used in expressions.

- Table 67 describes the syntax for references to graphic entities.

- Table 68 describes the syntax for data entity references.

- Table 69 describes the syntax for resource references.

Table 66. Prefixes used in the expressions

Prefix	Description
pr::	Reference to a property
lc::	Logical color
nt::	Reference to an NLS text
im::	Reference to an image
ev::	Reference to an expression variable
ip::	Reference to an input property
is::	Reference to out terminal of a graphic or input item
lv::	Local variable such as _MouseOver or _Now
ft::	Reference to a logical font

Table 66. Prefixes used in the expressions (Continued)

Prefix	Description
vw::	Reference to an aspect view
vr::	Reference to an object or aspect verb
br::	Reference to a logical brush
hl::	Reference to a history reference

Table 67. Syntax for references to graphic entities

Symbol Type	Syntax
Input property reference	[ip::]<Input property name>
Expression variable reference	[ev::]<Expression variable name>
Reference to out terminal of a graphic item or input item	[is::]<Input item name>.<Out terminal name>
Local variable	[lv::]<Name of the local variable>
Mouse variable	[lv::]<Name of the mouse variable>

A sub property *IsConnected* exists for input properties. The *IsConnected* property is used to check if an expression, which is not a constant value, is connected to the input property for the current element instance. The following is the syntax:

```
<input property name>.IsConnected
```

For example:

```
'Direction.IsConnected'
```

where *Direction* is an input property.

IsConnected is typically used in a situation where an element accesses a value through a **PropertyRef** property but also has an **OverrideValue** property. The element may fetch the value to present through the **PropertyRef** property when an expression is not connected to the **OverrideValue** property and otherwise from **OverrideValue** property. *IsConnected* can be used to determine the source of the value.

Table 68. Syntax for data entity references

Symbol Type	Syntax
Reference to aspect object property	[pr::][.:<Aspect Name>:]<Property specification>
	[pr::]<Object Symbol>:[<Aspect Name>:]<Property specification>
Reference to aspect view	[vw::][.:<Aspect Name>:]<View Name>
	[vw::]<Object Symbol>:<Aspect Name>[:<View Name>]
Reference to object verb	[vr::][.:<Aspect Name>:]<Verb Name>,
	[vr::]<Object Symbol>:[<Aspect Name>:]<Verb Name>
Reference to aspect verb	[vr::].:<Aspect Name>:<Verb name>
	[vr::]<Object Symbol>:<Aspect Name>:<Verb Name>
Reference to history log	[hl::][.:<Aspect Name>:]<Property specification>[,History Log specification]
	[hl::]<Object Symbol>:[<Aspect Name>:]<Property specification>[,History Log specification]

Dual formats are presented in Table 68 for each data reference. The format without **<Object Symbol>** is used when the referenced data entity exists on the invocation object for the graphic aspect. **<Object symbol>** should be specified if the data entity exists on any other object.

Object symbol is the name of the target object if this name uniquely identifies the object. Otherwise object symbol represents the path. For example, AncestorName/..../ObjectName.

<Property specification> in aspect object property references, is the name of the property.

The syntax for a verb reference is different for object and aspect verb references. Aspect name is mandatory for aspect verb references.

Table 69. Syntax for resource references

Symbol Type	Syntax
Reference to Logical color	[lc::][<Color group name>:]<Name of the logical color
Reference to NLS text	[nt::][<Text group name>:]<Text identity>
Image reference	[im::][<Image group name>:]<Name of the image>
Reference to a logical font	[ft::][<Font group name>:]<Name of the logical font>
Reference to a logical brush	[br::][<Brush group name>:]<Name of the logical brush>

Symbol Quoting

Quoting of a symbol is required when the symbol interferes with the expression parsing. Such symbols are enclosed in *$'....'*.

Symbol quoting is required for:

- Symbol containing characters that are relevant for expression parsing such as space, +, -, and #.

- Symbol beginning with a numeric character.

- Symbols identical to expression keywords such as if, then, else, true, or false.

For example, consider a symbol *Pump-Lower*. Quoting is required for this symbol, because it would be assumed as two symbols (*Pump* and *Lower*) and a subtraction function if parsed without quoting.

A **'** character is not permitted in an expression symbol.

Character Escaping

A symbol might consist of several names (object, aspect, property ...) being separated by the **:** (colon) character. Therefore, if a name contains one or several colons, these need to be escaped as **\:** to prevent the colons from being interped as name separators. A property name *My:Property* has to be written as *My\:Property* when being part of an expression symbol.

Table 70 describes the characters that need character escaping.

Table 70. Character Escaping

Character	Description	Rendered as
:	Colon used as delimiter in symbols	\:
\	Escaping character	\\

Symbol Ambiguity

A symbol is considered as ambiguous if more than one target entity is found based on the symbol.

For example, consider an expression variable *myValue* and also an aspect object property *myValue* on the invocation object. The ambiguous symbol *myValue* can be made unique:

• If the intended target is the expression variable, use the prefix *ev::myValue*.

• If the intended target is the aspect object property, use any of the following:

    – *pr::myValue*

    – *$'.:General Property:myValue'*

Adding a prefix does not disambiguate an ambiguous symbol if it can be resolved to more than one target of the same type.

For example, there are two different *General Property* aspects on the same object, both having a property of the same name. In this situation, it is not possible to createa unique symbol. It is still possible to create a unique reference using the reference browser. For more information on references, refer to Reference Status on page 143.

Consider an object *myObject* with an aspect *myAspect* with a property *Value*. There also exists another object named *myObject*.

To create a unique symbol, the object part can not only be the name of the object. An attempt is done to create a path *parentobject\myObject* which uniquely identifies the object. The *parentobject* is first searched in the **Functional Structure**. If not found, the search is done in the **Control Structure**. If this attempt fails, the symbol will not be unique.

Coping with Non Unique Symbols

In some situations, it is not possible to create a unique symbol because of improper naming of target entities. Such symbols if typed using keyboard, cannot be resolved.

The following are the prerequisites to handle a non unique symbol within an expression:

* The symbol must be unique within the scope of the expression being edited.

* The referenced entity exists as a prebound symbol.

> A prebound symbol is a symbol representing a reference which was resolved in the expression before beginning to edit it or a reference which was entered using a browse tool.

* The non unique symbol is exactly as it was generated by the expression editor.

If these prerequisites are not accomplished, the user should adjust the naming of target entity to eliminate the non unique symbol.

Local Variables

Graphic aspects contain local variables (for example, *_MouseOve*r and *_Now*). These variables are accessed as symbols in expressions. The syntax for accessing a local variable is,

```
<Name of local variable> or lv::<Name of local variable>
```

Table 71 describes the mouse variables, that is variables that reflect the state of the mouse. These variables allow the mouse interaction to be displayed by an instance of a graphic element.

Table 72 describes the *_Now* variable, which represents present time.

Table 73 describes the variables, which are used for the implementation of custom layout schemes. These variables are normally used only when the *Presentation Mode* of a graphic aspect is set to *Anchored*.

Table 71. Mouse Variables

Variable	Data Type	Description
_MouseOver	Boolean	During mouse captured event: _MouseOver is *True* in the captured item when the mouse hovers over it. When the mouse is not captured, _MouseOver is *True* in the mouse event receiver (item over which the mouse hovers) and *False* in any other mouse event consumers.
_MouseCaptured	Set Enum: None, Left, Right, XButton1, XButton2	_MouseCaptured takes a value other than *None* in the capture item. The value is *None* in a other mouse event consumer. The value of _MouseCaptured reflects the mouse buttons that were initially pressed.
_MouseButtonState	Set Enum: None, Left, Right, XButton1, XButton2	This is reflected in the capture item and provides the current state of the mouse buttons.The value is *None* in graphic aspect instances not having captured the mouse.

Table 72. Time Variables

Variable	Data Type	Description
_Now	DateTime	Specifies the current date and time in UTC format. This value can be converted to a string representation using FormatDateTime on page 256.

Table 73. Custom Layout variables

Variable	Data Type	Description
_Width	Real	Specifies width of the current instance of the element.
_Height	Real	Specifies height of the current instance of element.
_OriginalHeight	Real	Specifies the configured height of the element.
_OriginalWidth	Real	Specifies the configured width of the element.

Out Terminals

Out Terminals are defined for all graphic items (for example, Push Button and Input Bar) and most input items (for example, Property Writer and Bool Dew) in a graphic aspect.

Out Terminals are used to access information from an inner element. The syntax for accessing Out Terminals is,

```
[is::]<Name of item>.<Name of Out Terminal>
```

For example, consider a graphic display with a *Text* primitive. Add a *Property Writer* input item to this primitive.

The following are examples of out terminals that are accessible:

- *Text.Extent*, which returns a rectangle.

- *PropertyWriter.WriteInProgress*, which returns a boolean value. It specifies whether a write operation is in progress.

The user can also create out terminals while configuring generic and graphic elements. To create an out terminal, define an expression variable with the attribute *Exposed as Out Terminal*.

No Value Handling

An expression may yield or not yield a value during the execution. The following are examples of expressions that do not yield a value:

- An expression which is a reference to an aspect object property yields no value until the value of the property is provided by the data subscription facility.

- An expression yields no value if an error occurs while executing this expression.

For example, consider an expression **target#Value** where target is an input property of type **PropertyRef**. This expression yields no value if the value of target is null.

Every value handled by Process Graphics can be considered to include two parts:

- The "no value" flag.

- The actual value.

If "no value" flag is set to *True*, the value part is set to a well defined value; the null value for the values data type.

Each data type has a null value. The following are examples of null values:

  – *0* for **Integer** data type

  – *0.0* for **Real** data type.

  – "" for **String** data type.

  – **False** for **Boolean** data type.

General rule for handling no value in expressions

The general behavior of an expression is to unconditionally yield no value whenever it encounters a "no value" value.

Consider the following expression which calculates the mean value of two properties:

```
(prop1 + prop2)/2
```

If **prop1** or **prop2** has no value, this expression evaluates to no value. The expression will not yield a value of **(prop1)/2** or **(prop2)/2** if the value of only one property is retrieved.

The common behavior of an expression statement is to propagate no value when encountering no value, but some expressions statements are exceptions to this rule.

The following expression statements do not always propagate no value encountering in response to no value:

• if-then-else statements.

• Logical operators (&& and ||).

• Late binding functions.

• Sub properties for property references.

> Subsequent sections describe when each of the above expression statements do not propagate a "no value" value.

Handling of no value in if-then-else expressions

The **if-then-else** statement yields the value of the **else** branch if a "no value" value is encountered in its condition statement.

Consider the following expression:

```
if (prop1 + prop2)/2 > 50 then
  Green
else
  Red
```

This expression yields *Red* that is not a no value, when the statement *(prop1 + prop2)/2 > 5* yields a no value. This statement encounters a no value when either *prop1* or *prop2* has a "no value" value.

Encountering a condition value of no value and a value of *False* yield the same result.

i   "No value" values encountered in the True or False branches of the **if-then-else** statement propagate according to the general rule as described in General rule for handling no value in expressions on page 271.

Consider the following examples.

These examples may appear identical but they are not, because no value is not handled in the same manner.

Example 1

```
if prop > 50 then
   True
else
   False
```

This expression yields *False* when **prop** has no value.

Example 2

```
prop > 50
```

This expression yields no value when **prop** has no value.

i   The second expression can replace the first if handling no value in **prop** can be ignored.

It may be an unwanted feature that the outcome at condition = no value does not differ from the outcome when condition = false. In such cases, the following nested **if-then-else** statement should be used which tests for "no value"

```
if !condition#HasValue then
    <return value at condition = no value>
  else if condition then
```

```
      <return value at condition = true>
  else
    <return value at condition = false>
```

Adding this test for "HasValue" may be important for two reasons:

1. To avoid the risk of the operator interping the absence of a value as a real value.

2. To gain performance.

A common situation is that, value of condition starts with NoValue and then changes to true or false. Performance can therefore be gained by not drawing or performing any other time consuming operation at no value which changes to the presentation for condition = true or false.

Handling no value in logical operations

The following are logical operations:

–   A && B

–   A || B

Logical operations can be used for controlling the execution of an expression. For example, in the expression **statementA && statementB**, **statementB** is not executed when **statementA** yields *False*. The mechanism can be used to improve performance.

Consider the values of the expressions **A && B** and **A || B** as described in Table 74.

Table 74. Values of A && B and A || B

Value of A	Value of B	Result of A && B	Result of A || B
False	False	False	False
True	False	False	True
No value	False	False	No value
False	True	False	True
True	True	True	True

Table 74. Values of A && B and A || B (Continued)

Value of A	Value of B	Result of A && B	Result of A || B
No value	True	No value	True
False	No value	False	No value
True	No value	No value	True
No value	No value	No value	No value

Handling no value in late binding functions

Late binding functions never yield a "no value" value.

A late binding function does not start to resolve a value until all the parameters have a value. Table 75 describes the urned value of late bound functions.

Table 75. Late binding functions - Return Values

Late bound function category	urned value before resolve complete	urned value after resolve complete
Functions urning a single reference (for example, LateBoundProperty-Ref)	null	A resolved reference
Functions urning an array of refer-ences (for example, LateBound-PropertyRefArray)	An array with length = 0	An array of resolved refer-ences
Functions urning resource values (for example, LogicalColorFrom-Name)	The fallback value	The resolved resource value

Quality sub properties

The quality sub properties, that is, the properties of PropertyRef values do not yield no value even if the data subscription has not retrieved the value for the referenced

property. The sub properties, **IsGood** and **IsBad** yield *False* while waiting for the data subscription to return a value.

Handling no value in Graphic Items

Several graphic items do not specifically test for "no value" on their properties. These just accept the null value that is provided by the value part. The presentation of some graphic items changes when encountering "no value" values.

The **Text**, **Bar**, and **RangeBar** graphic primitives check for no value. The **Text** primitive handles a no value by not drawing the text. The **Bar** and **RangeBar** primitives handle no value by not drawing the bar. This behavior of **Text**, **Bar**, and **RangeBar** primitives avoids the misconceptions by operators, that is, mistaking a no value and null value for a correct value.

Testing for no value values

The **#HasValue** property is used to verify the existence of a value. This property is applicable for most data types but not Reference arrays or Reference data types. Reference types may have a null value and Reference arrays may have a length of zero. Both Reference arrays and Reference data types never evaluate to a no value.

The **#HasValue** property can be applied to input properties, expression variables, and out terminals.

Input Properties, Expression Variables, and Out Terminals may have "no value" values.

Input Properties: If a "no value" value is applied to a property of the graphic item which is an instance of a graphic element or generic element, then no value is encountered while accessing the corresponding input property inside the element.

Expression Variables: An expression variable has no value when the expression yields no value.

Out Terminals: An out terminal can be implemented using an expression variable. If an expression variable has a "no value" value, the corresponding out terminal also yields a no value.

Consider the following examples:

Example 1

```
if (prop1 + prop2)#HasValue then
   prop1 + prop2
else
   3.14
```

This expression returns 3.14 if *prop1* or *prop2* has no value.

Example 2

```
if ip::MyInputProp#HasValue then
   ip::MyInputProp
else
   3.14
```

This expression returns 3.14 if *MyInputProp* has no value.

Example 3

```
if ev::MyExpVar#HasValue then
   ev::MyExpVar
else
   3.14
```

This expression returns 3.14 if *MyExpVar* has no value.

Example 4

```
if is::IntegerDew1.ValueToWrite#HasValue then
   is::IntegerDew1.ValueToWrite
else
   3.14
```

This expression returns 3.14 if *IntegerDew1.ValueToWrite* has no value.

Example 5

```
if !pr#IsNull then
   pr#Value#HasValue
```

```
else
   False
```

where, *pr* is a reference to an aspect object property.

This expression returns *False* if:

– *pr* is Null.

– *pr* has a value but *#Value* returns no value.

This expression also avoids generating diagnostic messages by not executing *pr#Value* when *pr* is Null.

The **.HasNoValue** sub property that is limited to input properties and expression variables, is operational in the 800xA system version 5.1 and later versions, in spite of being superseded by the **#HasValue** property

Section 5 Resource Management

Resource management allows the user to create image, font, and brush resources that can be accessed by the graphic aspects in Process Graphics. The user can also create resource strings associated to a specific language through Resource management. Each resource contains a unique ID.

Resource management is supported by the **NLS Resource Data / NLS Resource Manager** aspect.

Creating the aspect for Resource Management

Execute the following steps to create the **NLS Resource Data / NLS Resource Manager** aspect.

- Select the structure and object to create the aspect.

- Right-click on the object and select **New Aspect** from the context menu. The **New Aspect** dialog box appears.
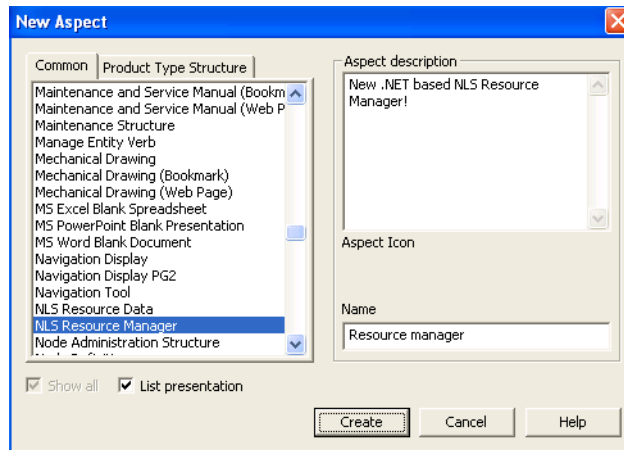
Figure 88. New Aspect

- Select **List Presentation** and a list of aspect categories appear.

- Select **NLS Resource Manager / NLS Resource Data**.

- Type a name for the aspect and click **Create**.

Config View

The *Config View* of the **NLS Resource Manager** or **NLS Resource Data** aspect allows the user to add resource strings corresponding to a specific language.

NLS Resource Manager aspect is preferred to be used while handling string resources.

Execute the following steps to open the Config View of the aspect.

- Right-click on **NLS Resource Manager / NLS Resource Data.**

- Select *Config View* from the context menu. The following window appears.
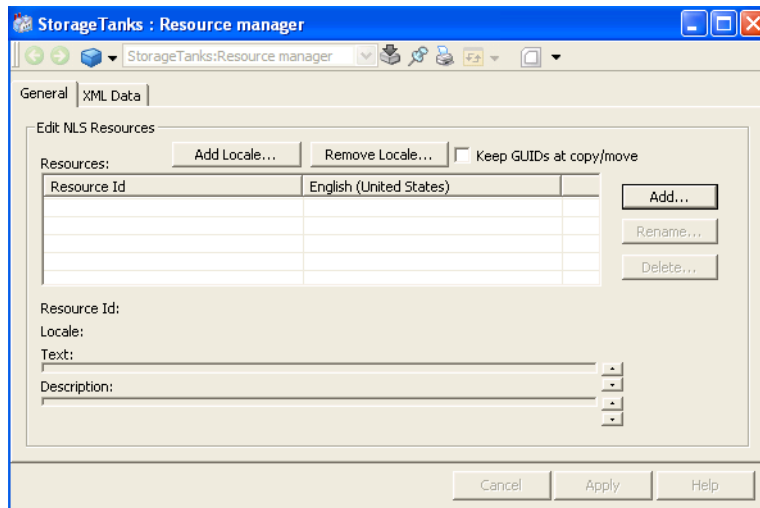
Figure 89. Config View

General tab

The General tab in the *Config View* is used to add resource strings and locales.
Execute the following steps.

* Click **Add** to add an NLS string. The **New Resource** dialog box appears.
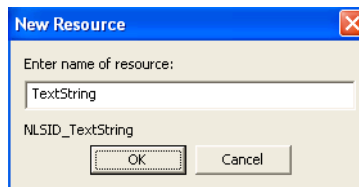
Figure 90. New Resource

* Type a name for the resource and click **OK**.

- Click the language column corresponding to the resource to add the NLS string. Refer Add locale to add a language or Remove locale to remove a language.

- Type the text for the language in **Text**.

- Type a description for the text if required in **Description**.

> ℹ️ Select the resource and click **Rename** to give a new name for the selected resource. The **Rename Resource Id** dialog box appears. Enter a new name for the resource and click **OK**.
>
> Select the resource and click **Delete** to delete the selected resource. A message box appears for confirming the deletion. Click **OK** to delete the resource. Otherwise click **Cancel**.

- Click **Apply** to save the changes.

Add locale

Click **Add Locale** to add a new language. The following dialog box appears.
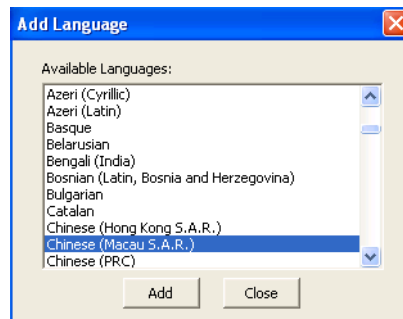
Figure 91. Add Language

Select a language from the list of available languages and click **Add**. Click **Close** and the selected language is added in **Resources**.

Remove locale

Click **Remove Locale** to remove any language from **Resources**. The following dialog box appears.
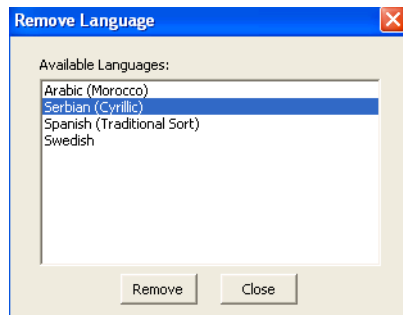
Figure 92. Remove Language

Select the language from the available list and click **Remove**. Click **Close** and the selected language is removed from **Resources**.

English (United States) is a default language and cannot be removed.

XML Data tab

The XML Data tab in the *Config View* helps in bulk data management of resources. It is used to import string resources from an XML file to the aspect or export the string resources from the aspect to an XML file.

The XML file contains string resources. The user can edit the string resources using an XML tool.
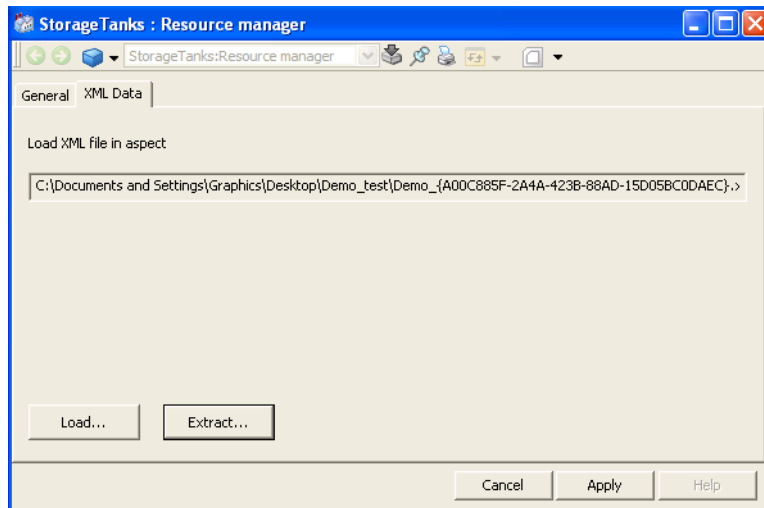
Figure 93. XML Data

Click **Load** and select the XML file that is to be loaded into the aspect. The path of the selected file is displayed in **Load XML file in aspect**.

Click **Extract** and select the XML file to which the resources from the aspect are to be stored. The path of the selected file is displayed in **Extract aspect to XML file**.

Click **Apply** to save the changes.

Main Config View

The *Main Config View* of the **NLS Resource Data** aspect allows the user to add image, font, and brush resources. These resources can be used by the graphic aspects in Process Graphics. This view appears only for **NLS Resource Data** aspect.

Execute the following steps to open the Main Config View of the aspect.

*   Right-click on **NLS Resource Data.**

*   Select *Main Config View* from the context menu. The following window appears.
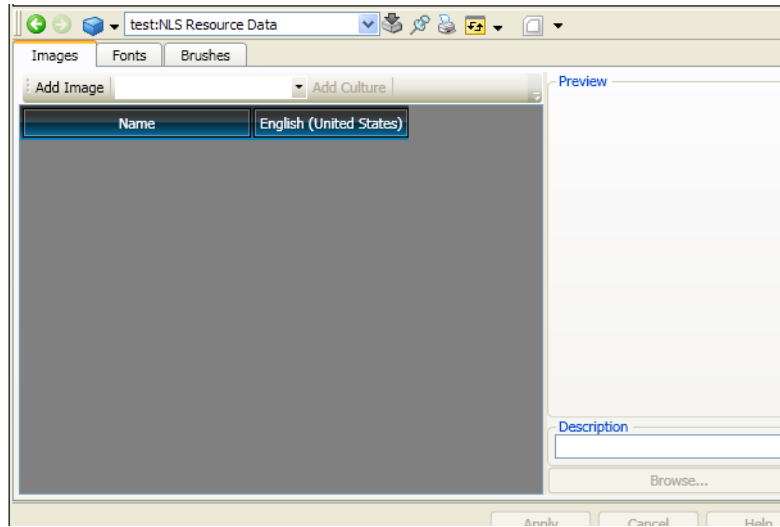
Figure 94. Main Configuration View

*   To select a culture for an image or font resource, select the required culture from the drop-down list and click **Add Culture**.

Images tab

The Images tab allows the user to add image resources.
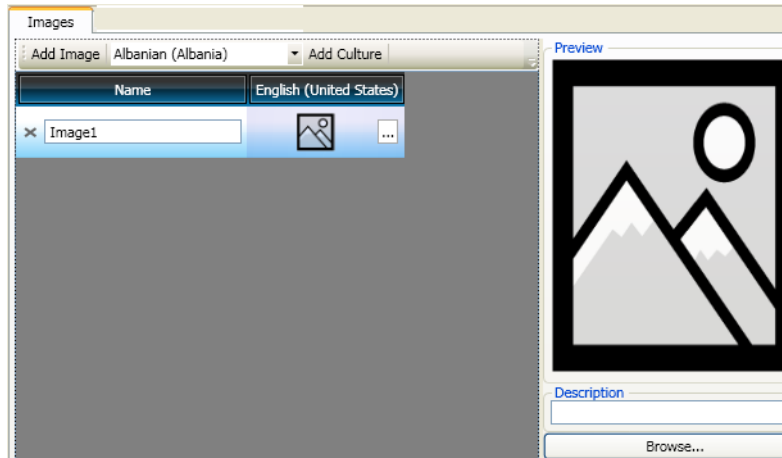
Figure 95. Image

Execute the following steps to add an image.

- Click **Add Image** to add a new image.

- Type a **Name** for the image.

- Select the desired culture of the image, and type a **Description**.

- Click **...** and select the image to be added. The selected image appears in **Preview**.

- Click **Apply** to save the changes.

To delete an image, select the image from the resource area and click ✕.

Fonts tab

The Fonts tab allows the user to add font resources.
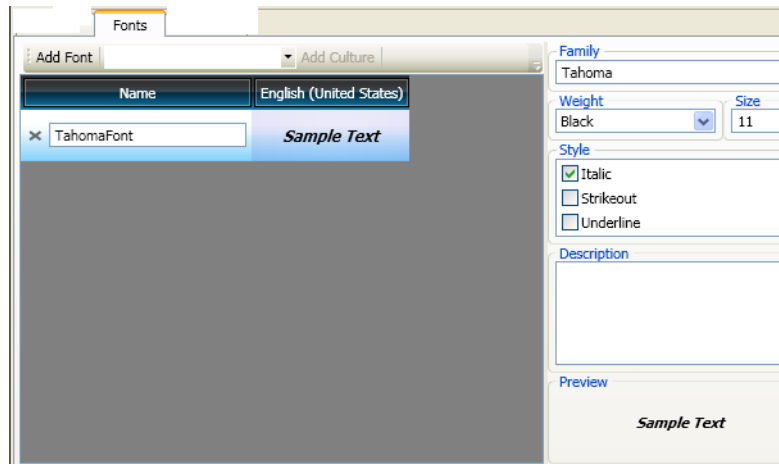
Figure 96. Font

Execute the following steps to add a font.

- Click **Add Font** to add a new font.

- Type a **Name** for the font.

- Select the desired culture of the font.

- Type the **Description** and select the **Family**, **Size**, **Style**, and **Weight** for the
  font. **Preview** displays a sample text in the new font.

- Click **Apply** to save the changes.

To delete a font, select the font and click ✖.

Brushes tab

The Brushes tab allows the user to add brush resources.
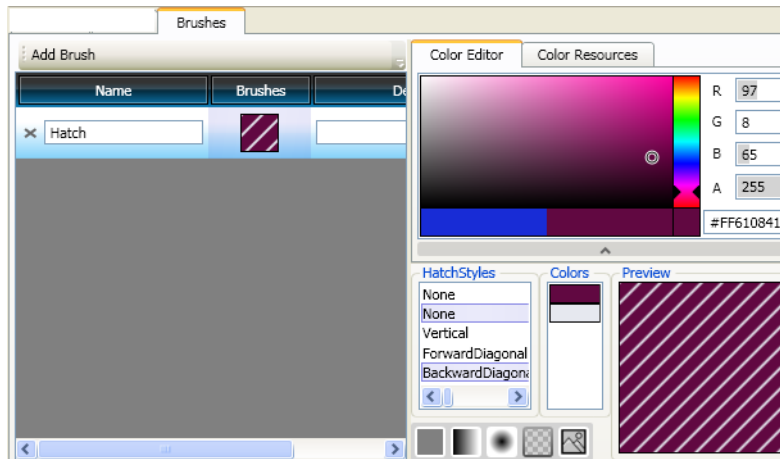
Figure 97. Brush

Execute the following steps to add a brush.

• Click **Add Brush** to add a new brush.

• Type the **Name** and **Description** for the brush.

• Click **Brushes** and select any one of the following for the brush.

– Solid color

– Linear gradient

– Radial gradient

– Hatch

– Image

Refer to Solid Color, Linear gradient, Radial gradient, Hatch, Image for setting the brushes.

• Select the color using **Color Editor** or **Color Resources**.

In **Color Editor**, type different values for **R, G, B,** and **A** to get the corresponding color. Or, drag the "o" appearing within the editor to get the required color.

R is the value for Red, **G** is the value for Green, and **B** is the value for Blue. The values for all factors can vary between 0 and 255.

When the value for **A** is 0, the item is transparent, and when it is 255, the item is opaque.

In **Color Resources**, select a required color from a group. This displays the system colors.

The color editor cannot be used for *Image* brush.

- The color selected using the editor appears in **Colors**.

  Click **Add** to add another color. Use **Editor** to select a different color.

  Preview gives a view of the selected color.

Select a color and click **Rem** to remove the selected color.

Use **Up** and **Down** to change the order of the selected colors.

- Click **Apply** to save the changes.

To delete a brush, select the brush and click ✖.

Solid Color

This is used for brushes with a solid color.

Linear gradient

This is used for a 3D presentation of the selected color, based on the gradient stops and angle.
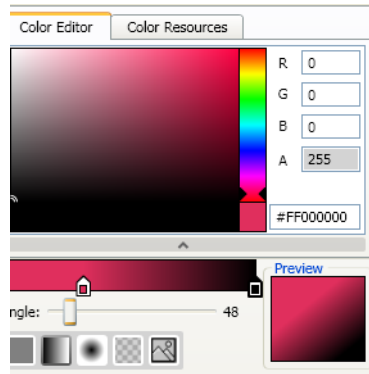
The user can add gradient stops by clicking the gradient bar. Right-click 🔒 to remove the gradient stop.

Select the angle to display the gradient stops by dragging 🔲 .

Radial gradient

This is used for a 3D presentation of the selected color, based on the gradient stops and center, origin, and radius of X and Y positions.
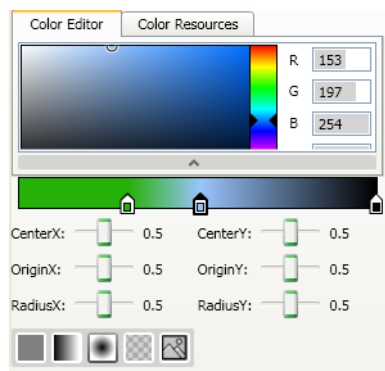
The user can add gradient stops by clicking the gradient bar. Right-click 🔒 to remove the gradient stop.

Specify the values for **CenterX, CenterY, OriginX, OriginY, RadiusX**, and **RadiusY** by dragging ⬚.

Hatch

This is used to give a shading for the color based on the selected pattern.
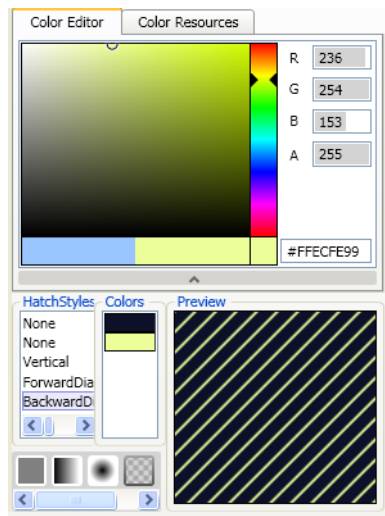
Select a pattern in **HatchStyles**.

In **Colors**, select the color for outer color and inner color.

Image

This is used to select an image resource or an image from the system. The user can set the tile styles to display the image.
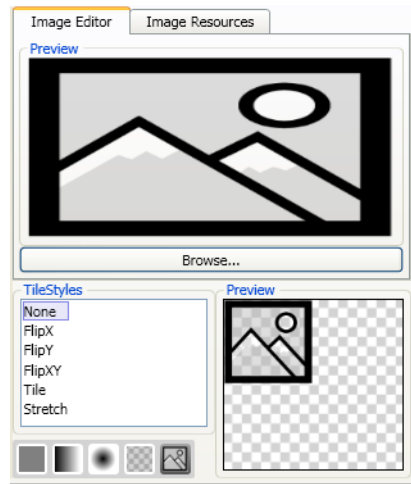
–   The *Image Resources* tab is used to select an image resource from the available list. This list contains all the images that are defined in the 800xA system (in the **NLS Resource Data** aspects). Click a resource and select the image from the list.

–   The *Image Editor* tab is used to select an image from the computer. Click **Browse** and select the image.

Select the style for displaying the image from **TileStyles**.

Section 6 Diagnostics Window

The Diagnostics Window is used to get an overview of the graphic aspect. This window presents the diagnostics information about an invocation of a graphic aspect. For example, this window is used if there are any performance issues or runtime errors for the graphic aspect.

The following information are presented in the Diagnostics Window:

– Duration for navigation, aspect creation, and subscriptions.

– The properties subscribed by the graphic aspect.

– Runtime error information for the graphic aspect.

– Late bound references of the graphic aspect.

– Miscellaneous information (for example, number of graphic items and number of elements) of the graphic aspect.

The Diagnostics Window is available for a runtime invocation and for the Graphics Builder. There are two ways to invoke the Diagnostics window.

1. In Graphics Builder, select **File > Diagnostics**.

2. In the workplace, right-click a graphic display or faceplate and select **Diagnostics** from the context menu.

The Diagnostics window in Graphics Builder is a subset of the Diagnostic window in runtime of a graphic aspect.

The Diagnostic window in runtime shows a detailed diagnostic information about the graphic aspect. In Graphics Builder, this window displays only the errors, warnings and miscellaneous information.

The Diagnostics Window contains six tabs:

* Summary.

* Timing.

* Subscriptions.

* Errors and Warnings.

* Late Binding.

* Misc.
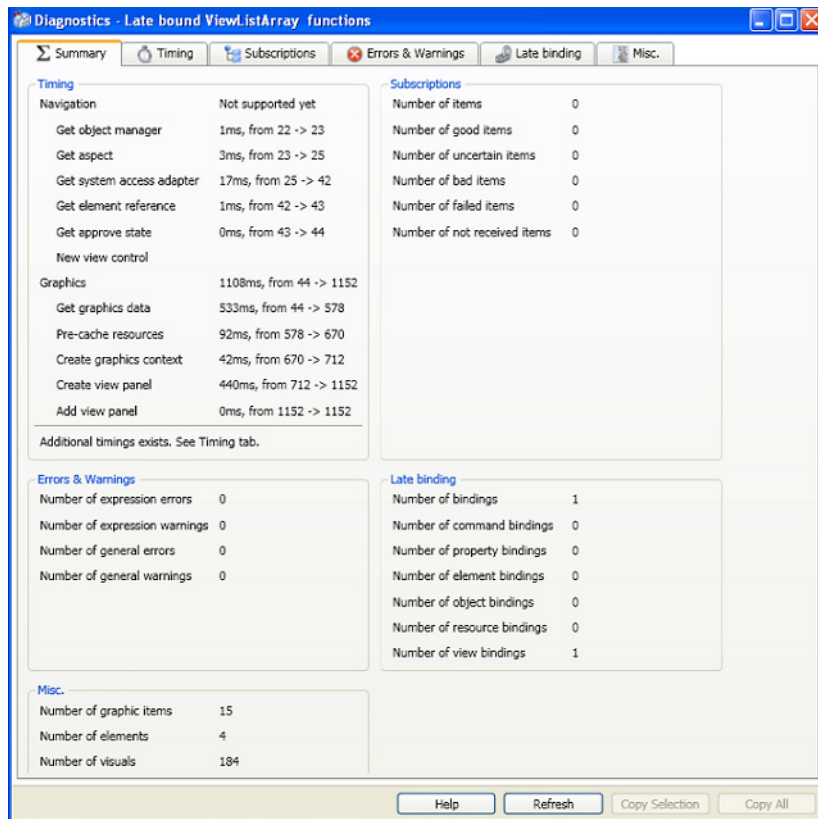
Figure 98. Diagnostics

Summary

Information from the other tabs in **Diagnostics** window are displayed in a summarized form in the **Summary** tab.
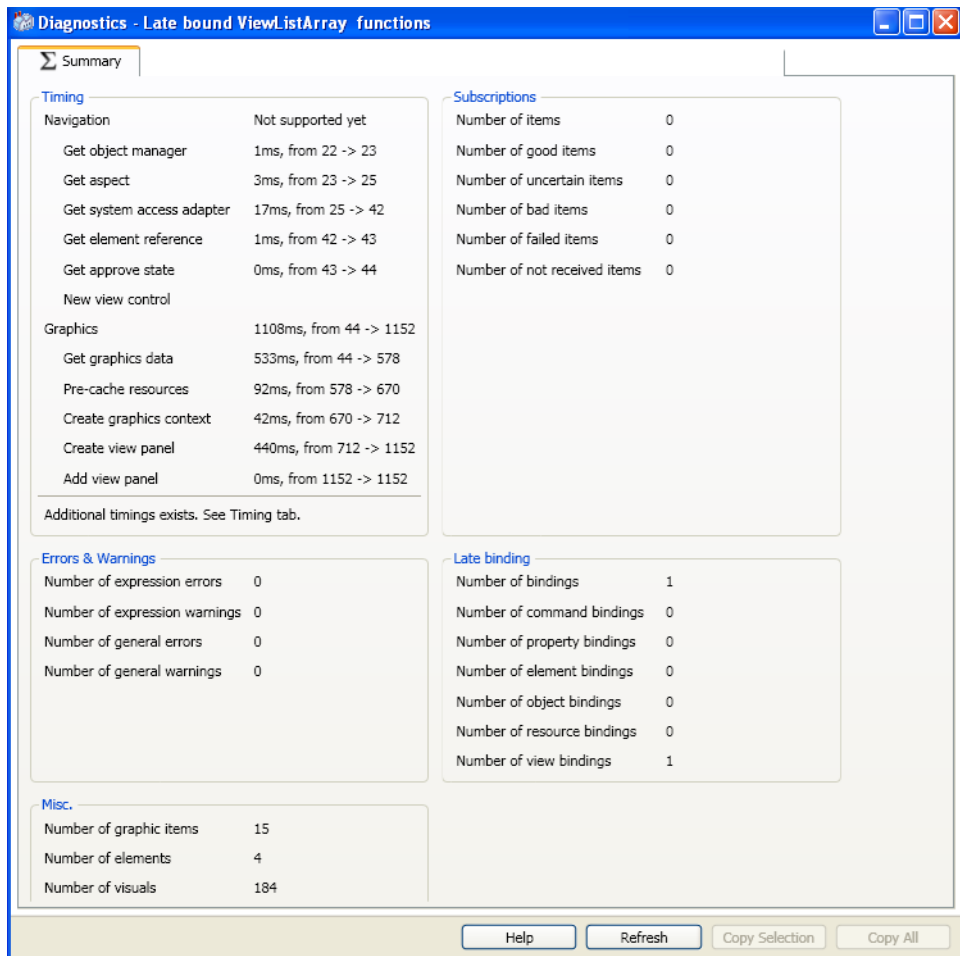
Figure 99. Summary tab

Timing

The **Timing** tab contains an expandable list of timed events for navigation, creation of the graphics and creation of subscriptions.
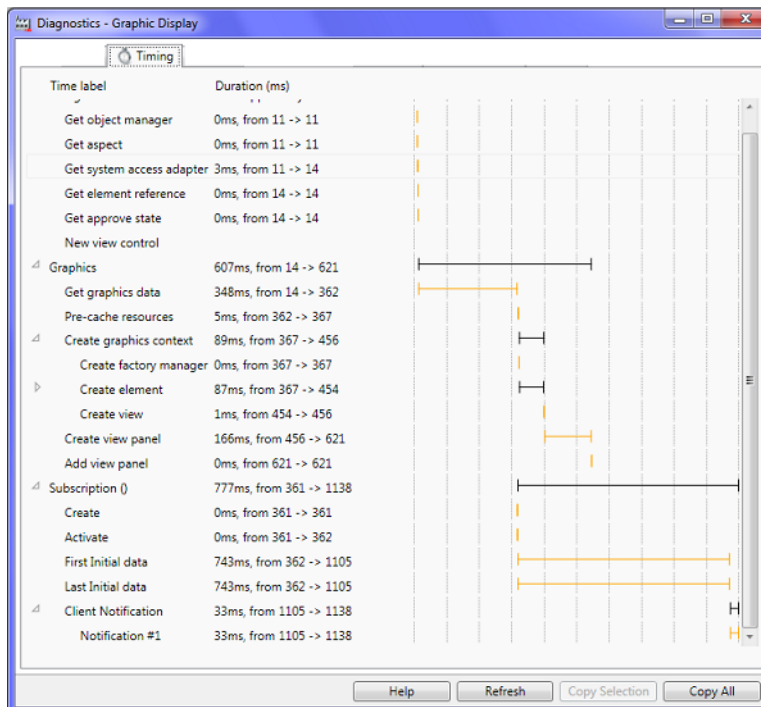
Figure 100. Timing tab

- **Time Label** displays **Navigation, Graphics, and Subscription**.

  – **Navigation** contains the navigation duration. It is the time starting from the selection of aspect to the start of display generation.

  – **Graphics** displays information on the graphics timing.

  – **Subscription** displays the subscription timing entry for each data subscription created by the aspect. Subscription category is displayed within parenthesis. *Empty* category contains subscription items that

originate from the property referenced in the aspect. *Late bound* category contains subscription items that originate from late bound property references. After resolving the late bound references, the subscription is activated. The first late bound subscription is displayed as Batch number 1 and the subsequent batches are incremented. Each subscription timing contain the following information:

Create is the time to create the subscription.

Activate is the time to activate the subscription.

First Initial Data displays the time starting from activation until the first initial data is received.

Last Initial Data displays the time starting from activation until the last initial data is received.

Client Notification displays the detailed subscription timing information.

- **Duration** displays the time in milliseconds or one of the following:

    – **Not supported yet** when the timing is not yet supported. This will be supported in future versions.

    – **Pending** when the operation has not been completed.

Subscriptions

The **Subscriptions** tab is visible if the aspect contains subscriptions. This tab displays a list of properties subscribed by the aspect. The user can view details of all subscription objects. The user can also select a subscription object to display the details.
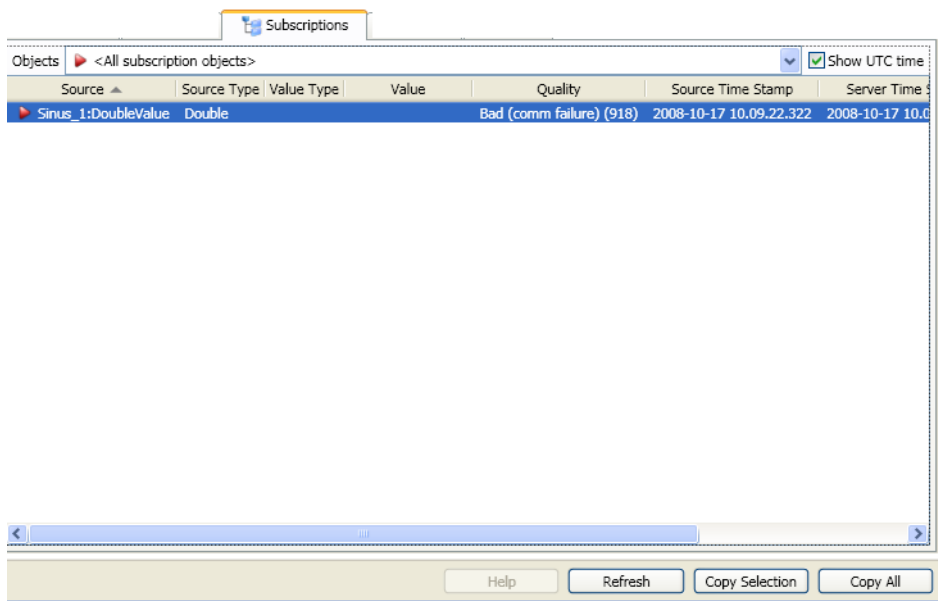
Figure 101. Subscriptions tab

Following are displayed for each subscription item.

- Select **Show UTC time** to show time stamp as UTC time. Otherwise it is shown in local time.

- **Source** specifies the source name.

- **Source Type** specifies the data type of the source.

- **Value Type** displays the data type of the received value.

- **Value** displays the current value.

- **Quality** displays the data quality of the current value.

- **Source Time Stamp** displays the time stamp defined by the source.

- **Server Time Stamp** displays the time stamp defined by the server.

- **Access** displays *R* if it is readable and *W* if it is writable.

- **First data (ms)** displays the time starting from activation until the first initial data for the subscription item is received.

- **Category** displays one of the following.

  - **Empty** for subscription items originating from direct property references in the aspect.

  - **Late bound (<Identifier>)** for subscription items originating from late bound property references. The first late bound subscription is displayed as Batch number 1 and the subsequent batches are incremented.

Each subscription item has a graphic indication that represents the current status.

- **Green** specifies Good quality.

- **Yellow** specifies Uncertain quality.

- **Red** specifies Bad quality.

- **Gray** specifies Failed or not received.

Each selectable object in **Objects** drop-down has a graphic indication that represents the status of subscription object.

- **Green** specifies that all subscription items for the object has good quality.

- **Yellow** specifies that one or more subscription items for the object has uncertain quality. Subscription item is not of bad quality, failed or not received.

- **Red** specifies that one or more subscription items for the object has bad quality. Subscription item is not failed or not received.

- **Gray** specifies that one or more subscription items for the object is failed or not received.

Click **Refresh** to update the list of items and the values.

Errors and Warnings

The **Errors and Warnings** tab contains runtime error information for the aspect. It displays the details of expression errors and errors not originating from execution of expressions. Select an instance from the **Instance** drop-down to display the error details if any.

Top portion of this tab displays the expression errors and the bottom portion of this tab displays the general errors.
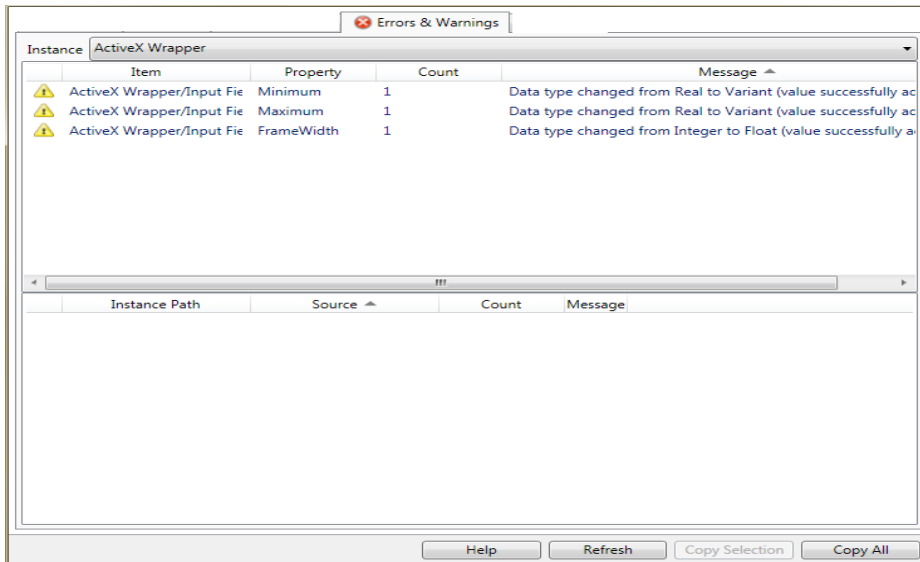
Figure 102. Errors and Warnings tab

Following details are displayed for each expression error.

• **Item** displays the graphic item or input item name having errors in expressions or empty expressions for expression variable expressions.

• **Property** specifies the property name or expression variable name where the expression is defined.

• **Count** displays the number of times the error has occurred.

• **Message** is a text representation of the error.

Following details are displayed for general errors.

• **Source** displays the error source.

• **Count** displays the number of times the error has occurred.

• **Message** is a text representation of the error.

InstancePath displays the path of the element instances in the following format.

```
<invoked graphic aspect>/<graphic element instance
name>/.../<graphic item name> or <input item name>
```

For example, consider a graphic display *ReactorDisplay* with a graphic element instance *Reactor1*. *Reactor1* contains another graphic element instance *Motor1*. *Motor1* contains a text primitive *Text1*. The **InstancePath** is displayed as *ReactorDisplay/ Reactor1/ Motor1/ Text1*.

Click **Refresh** to update the list of errors.

Late Binding

The **Late Binding** tab is visible if the aspect contains late bound references. This tab displays the late bound references defined on the aspect. Select an instance from the **Instance** drop-down to display the late bound reference details. The instances containing late bound references are available in the list.
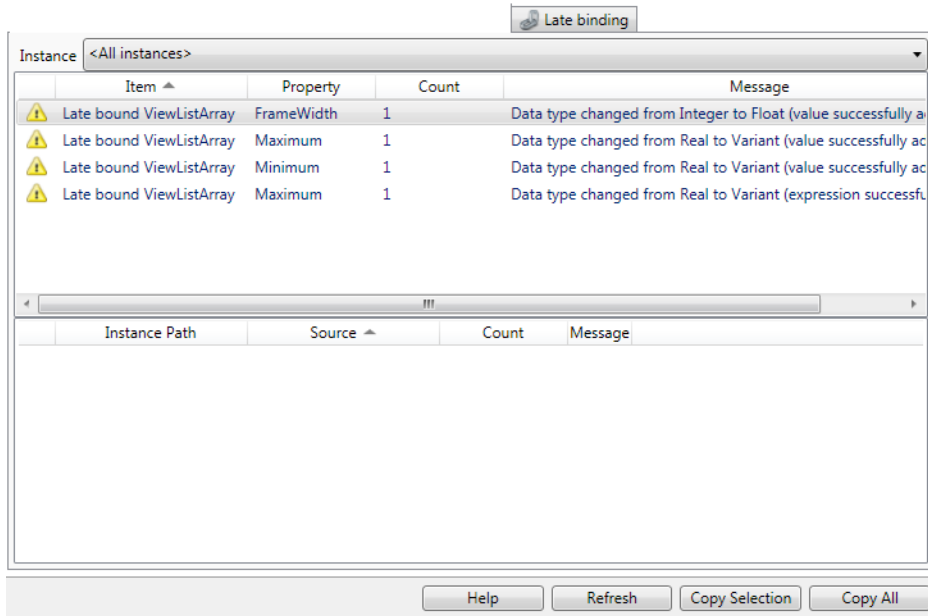
Figure 103. Late Binding tab

Following details are included in this tab.

• Select **Show UTC** to show the time stamp as UTC time. Otherwise it is shown in local time.

• **Type** displays one of the following:

  – **Resource** specifies a late bound resource reference.

  – **Command** specifies a late bound command reference.

  – **Object** specifies a late bound object reference.

  – **Property** specifies a late bound property reference.

  – **Property array** specifies a late bound property reference array.

  – **View** specifies a late bound view reference.

- – **View array** specifies a late bound view reference array.

- – **Element** specifies a late bound element reference.

- • **Parameters** displays the list of parameters of the late bound function.

- • **Unique** specifies the unique parameter of the late bound function.

- • **Execution time** displays the absolute time of initiation of execution of the late bound function. The time format is specified by **Show UTC**.

- • **Delay** specifies the time starting from the initiation of graphic aspect until the initiation of late bound function execution.

- • **Duration** specifies the duration of late bound function execution, that is, the time used by the system to resolve the late bound function.

- • **Bound item** is a text representation of late bound reference.

Misc.

The **Misc.** tab contains miscellaneous information about the aspect.

Number of graphic items	73
Number of elements	14
Number of graphic elements	0
Number of generic elements	14
Number of element types	10
Number of visuals	245
Render Tier	0.0
	No graphics hardware acceleration. The DirectX version level is less than version 7.0.
Main blob size	8039

[Help]   [Refresh]   [Copy Selection]   [Copy All]

Figure 104. Misc. tab

Following details are displayed in this tab.

- **Number of graphic items** displays the total number of graphic items in the aspect and the referenced aspects. Graphic items referenced by aspect view primitives or wrapped VBPG aspects are not included.

- **Number of wrapped Windows Forms items** displays the number of wrapped Windows Forms items. This is displayed only if the aspect contains wrapped Windows Forms.

- **Number of Wrapped ActiveX items** displays the number of wrapped ActiveX items. This is displayed only if the aspect contains ActiveX items.

- **Number of Wrapped WPF items** displays the number of wrapped WPF items. This is displayed only if the aspect contains wrapped WPF items.

- **Number of Wrapped VBPG items** displays the number of wrapped VBPG items. This is displayed only if the aspect contains wrapped VBPG items.

- **Number of Wrapped Aspect View items** displays the number of wrapped Aspect View items. This is displayed only if the aspect contains wrapped Aspect View items.

- **Number of elements** displays the total number of elements in the aspect including elements in the referenced aspects. Elements items in aspects referenced by Aspect View primitives or wrapped VBPG aspects are not included. Elements include the total of number of graphic elements and generic elements.

- **Number of graphic elements** displays the total number of graphic elements in the aspect including graphic elements in the referenced aspects. Element items in aspects referenced by Aspect View primitives or wrapped VBPG aspects are not included.

- **Number of generic elements** displays the total number of generic elements in the aspect including generic elements in the referenced aspects. Element items in aspects referenced by Aspect View primitives or wrapped VBPG aspects are not included.

- **Number of element types** displays the total number of element types referenced by the aspect including elements in the referenced aspects. Element types in aspects referenced by Aspect View primitives or wrapped VBPG aspects are not included.

- **Number of visuals** displays the total number of WPF visuals in the aspect.

- **Render Tier** displays a rendering tier. This defines the level of graphics hardware capability and performance for a device that runs a WPF application. For more information, refer to Microsoft Corporation.

- **Main Blob size** displays size of the main blob.

Section 7 Faceplate Framework

Faceplates are used for monitoring and controlling the process. Faceplates contain the following runtime views:

- Reduced Faceplate View (optional)

  The contents in this view are optimized to cover the most frequently used process operator actions. The faceplates elements in this view are small in size and can have faceplates for different objects visible at the same time.

- Faceplate View (mandatory)

  This view covers the normal process operator actions. It is the default view. The faceplates in this view are small in size and can have faceplates for different objects visible at the same time.

- Extended Faceplate View (optional)

  This view contains functions and information intended for the process engineer, or the advanced operator (for example, tuning display).

The **Config View** of a faceplate is used for configuring runtime views. For more information, refer to Configuring the Faceplate on page 322.

Figure 105 shows a faceplate available for an object in the **Functional Structure**. For more information on creating a faceplate, refer to Create a New Faceplate on page 320.
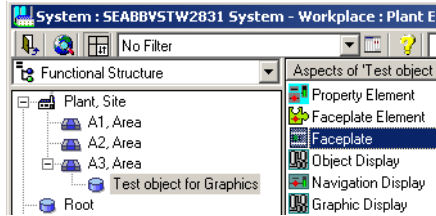
Figure 105. Faceplate aspect available for an object

The active faceplate view is indicated at the bottom of the faceplate.

Faceplate view is the default size of the faceplates. To adjust the size to reduced or extended, refer to Layout Tab on page 324.

Figure 106 shows examples of the three faceplate views.
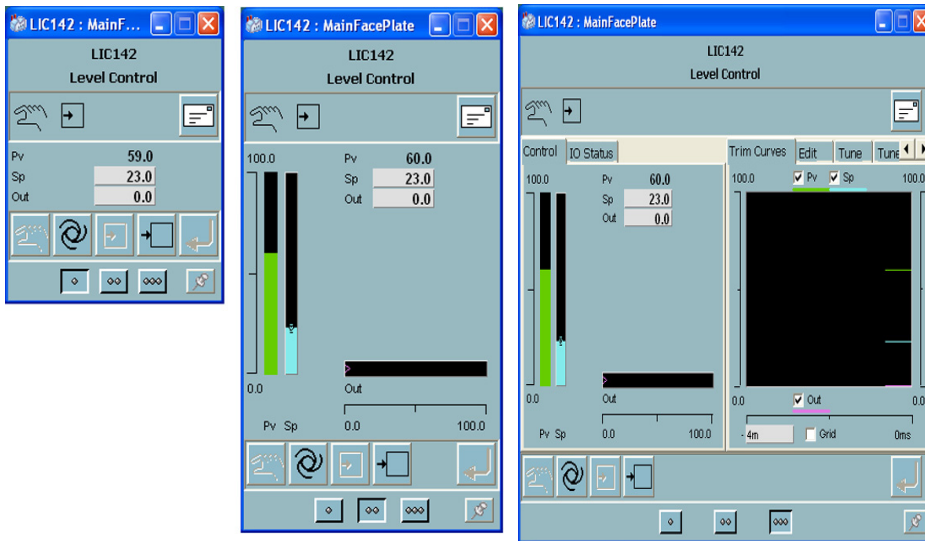
Figure 106. Reduced Faceplate, Faceplate and Extended Faceplate View Example

In Figure 107, the faceplate is launched by a left-click on a graphic element placed on a graphic display. Another way of launching a faceplate is to select an aspect in the workplace, right-click and select one of the three runtime views from the context menu.
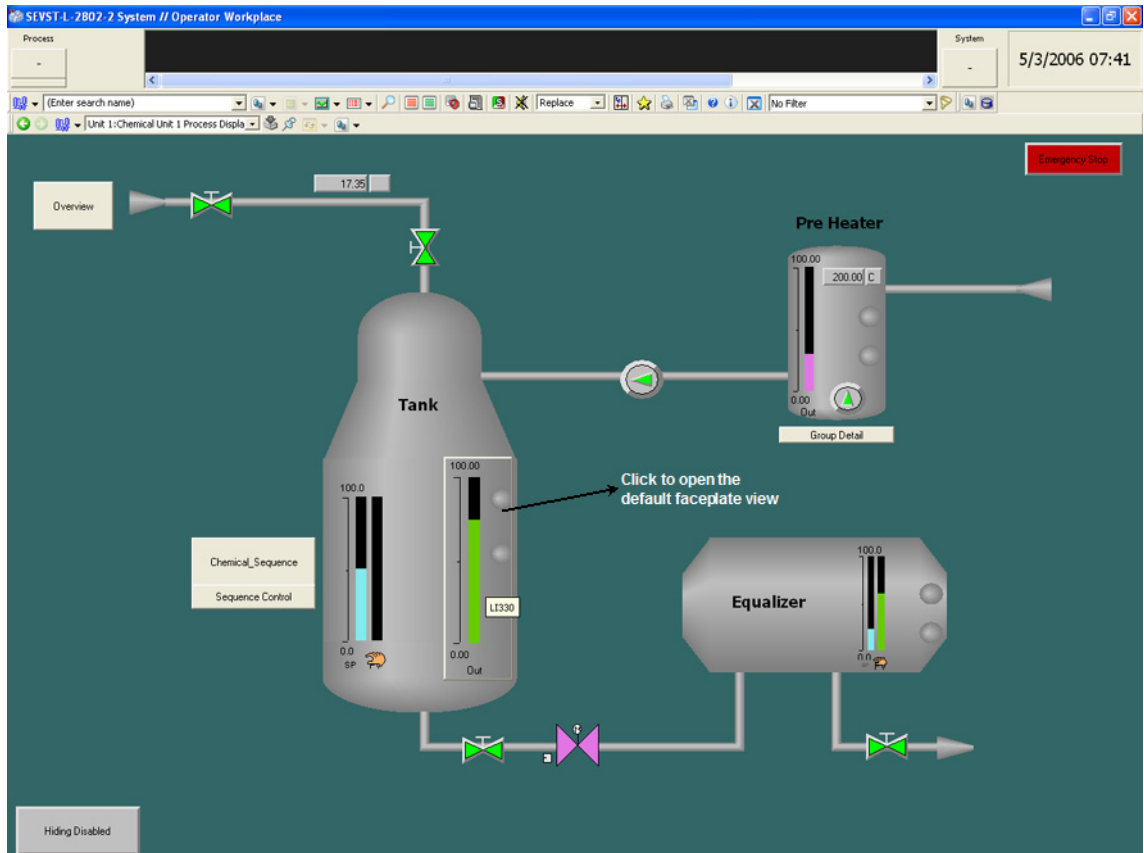
Figure 107. Graphic Display

Faceplate Overview

Faceplate is divided into different areas and each area is configured.
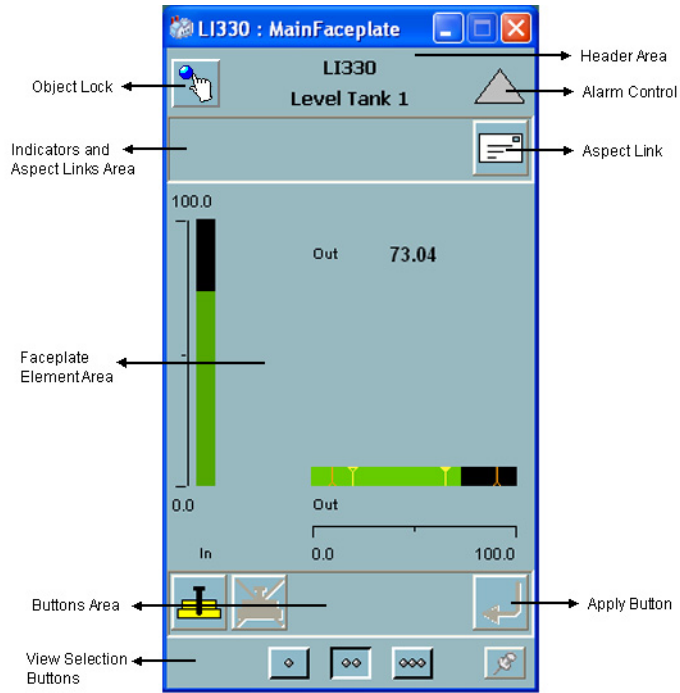
Figure 108. Faceplate Terminology

Header Area

Each faceplate contains a **Header Area**, which consists of the following:

• Object name

• Object description

• Alarm control

i   Object name and object description is optional because of user profile setting.

Object lock control and alarm control are configured in the **Header** tab of the faceplate configuration view.
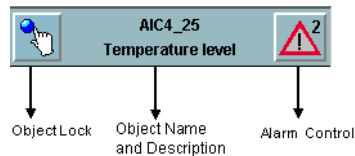
Figure 109. Example of a Header area

Table 76 describes the user profile used to control the visibility of the faceplate header information.

Table 76. User Profiles for faceplate header

Name	Description	Values
HideFaceplateHeaderString	Hides the object name and object description displayed in the header area of the faceplate.	To hide the header information in the faceplate: In **Graphic Profile Values**, select **Local** in **Datasource** frame and then select the **Value** check box.
HideFaceplateHeader*	Hides the header area of the faceplate.	To hide the header area of the faceplate: In **Graphic Profile Values**, select **Local** in **Datasource** frame and then select the **Value** check box.

*   Feature Pack Functionality

Lock Control

Object Locking is used in environments where several operator workplaces control the same objects. The Object Lock function gives the privilege only to one operator to operate the objects. The Lock Control button is optional. The faceplate framework just reserves space for it in the header area.

To configure the Object Lock function, refer to Header Tab on page 325. This aspect interacts with the Lock Server. For configuration and use of the Lock Server, refer to *System, 800xA, Administration and Security (3BSE037410*).*

If Autolock is enabled, the object will be locked after the user opens the faceplate. When the faceplate is closed, the Lock Status will revert to unlocked.

If the Object Lock function is controlled manually, the user locks an object by clicking the **Object Lock** push button (refer Figure 109). If the lock required option is enabled, all buttons and commands will be disabled till the user clicks the Object Lock button.

To switch between Autolock and manual lock, change the *FaceplateAutoLockObject* profile value in the *Graphics Profile Values user profile*.

ℹ️ If legacy locking (that is, a property named LOCK) is used, faceplate buttons and other controls in the faceplate element do not appear dimmed.
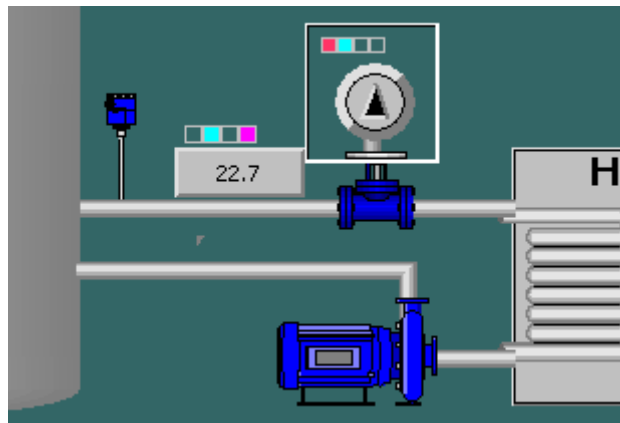
Figure 110. Example of a Locked Graphical Element

The white frame is displayed automatically on the object aware graphic element if the lock server is used. In Process Graphics 2, support for the white lock frame is included.

For more information on how to configure white object lock frame on graphic elements, refer to Object Locking on page 173.

The Object Lock push button has different appearances depending on the lock state. Table 77 and Table 78 displays the different icons.

Table 77. Lock Status Using the Lock Control Aspect.

Lock status	Button	Icon	Background Color
Unlocked	Raised		Bluegrey
Locked by me	Sunken		White
Locked by *user:machinename*	Flat		Yellow

Table 78. Lock Status Using the Legacy Locking

Lock status	Button	Icon	Background Color	Lock Property Value
Unlocked	Raised		Bluegrey	00
Locked by me	Sunken		White	11
Locked by other	Flat		Yellow	01

If the object is locked by another operator, operation is not possible until the lock is released. To release an object lock, the operator owning the lock can manually unlock it, or the operator can close the faceplate. The object lock may time out after a default time of four minutes. A privileged user can also break the object lock through the context menu of Lock Control.

Alarm Control

The Alarm Control button indicates the alarm state of the object, and allows acknowledgement of alarms. The alarm control is a graphic element aspect. Alarm Control details are displayed in the header area of the faceplate. Alarm button configuration is done using the AlarmControl aspect.

Table 79 shows the most common alarm indicators.

Table 79. Alarm Indicators

Icon	Description
▲1	Alarm is active and acknowledged.
⚠1	Alarm is active, but not acknowledged.
?	Indicates illegal alarm configuration.
▲1	Alarm is automatically disabled by the system.
⚠X1	Alarm disabled by a user.
▲	Alarm is idle, that is there are no alarms.

Table 79. Alarm Indicators (Continued)

Icon	Description
	Alarm is neither active, nor acknowledged.
	Hidden Alarm. A white rectangle covering an alarm symbol, shows that there are hidden alarms for the object.
	Shelved Alarm. A white circle appearing to the right of the alarm symbol shows that there are shelved alarms for the object.

Indicators and Aspect Links Area

Indicators and Aspect Links are displayed in this area. Refer to Indicators Tab on page 328 for more information on configuring this the indicators and aspect links area.

Indicators show a label or an icon, as the result of a configured expression. This expression includes different subscribed object properties. The standard format for icons are 32x32 (normal Windows icon size).

Aspect links are buttons that display another aspect as an overlap window.

 Status indicators are configured to the left, and aspect links to the right.

Layout settings in the *Config view* of the faceplate controls the maximum number of indicators and aspect links allowed in the status and navigation bar. A configured faceplate view configured as a default view, can contain six indicators and aspect links. The amount of rows to be displayed in the Status and navigation bar area are also configured. For more information on this configuration, refer to Layout Tab on page 324.

Faceplate Element Area

Aspects are displayed in the faceplate element area. It can also be displayed in tab groups. Faceplate element aspects should be included in this area. Other aspects may also be included.

The orientation of aspects or tab groups can be horizontal (the default order) or vertical, but not both in the same faceplate aspect.

Each tab in a tab group contains one aspect view. A scrollbar is visible to show all the tabs in a tabgroup.

The faceplate element area as shown in Figure 111 contains two tabgroups ordered horizontally.
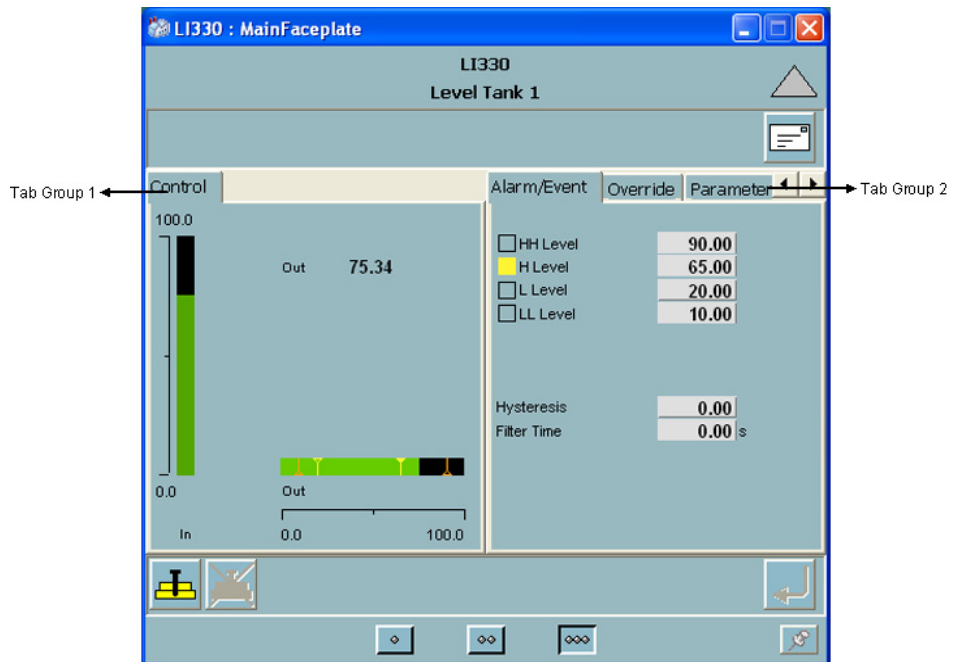
Figure 111. Example of a Faceplate Element Area

Buttons Area

Buttons that control properties of the object are included in the Buttons area. The number of button positions displayed in a row depends on the width of the faceplate view. The default number of button positions per row is six.

If a button is placed at position 7 it is discarded if there is only one button row. It is positioned first on the second row when there are more than one button rows configured.

The user can configure to view the **Apply** button. This button is always positioned to the right (static position) if it is visible. For more information on how to configure the Button area, refer to Buttons Tab on page 333.

View Selection Buttons

Select a faceplate view. If a view does not exist, the button representing that view is disabled.
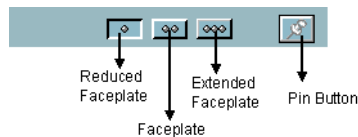
Figure 112. View Selection Buttons

Execute the following steps to hide the footer of a faceplate:

1. Select **Default** in the **Library Structure > Default View Class > Faceplate System View Class**.
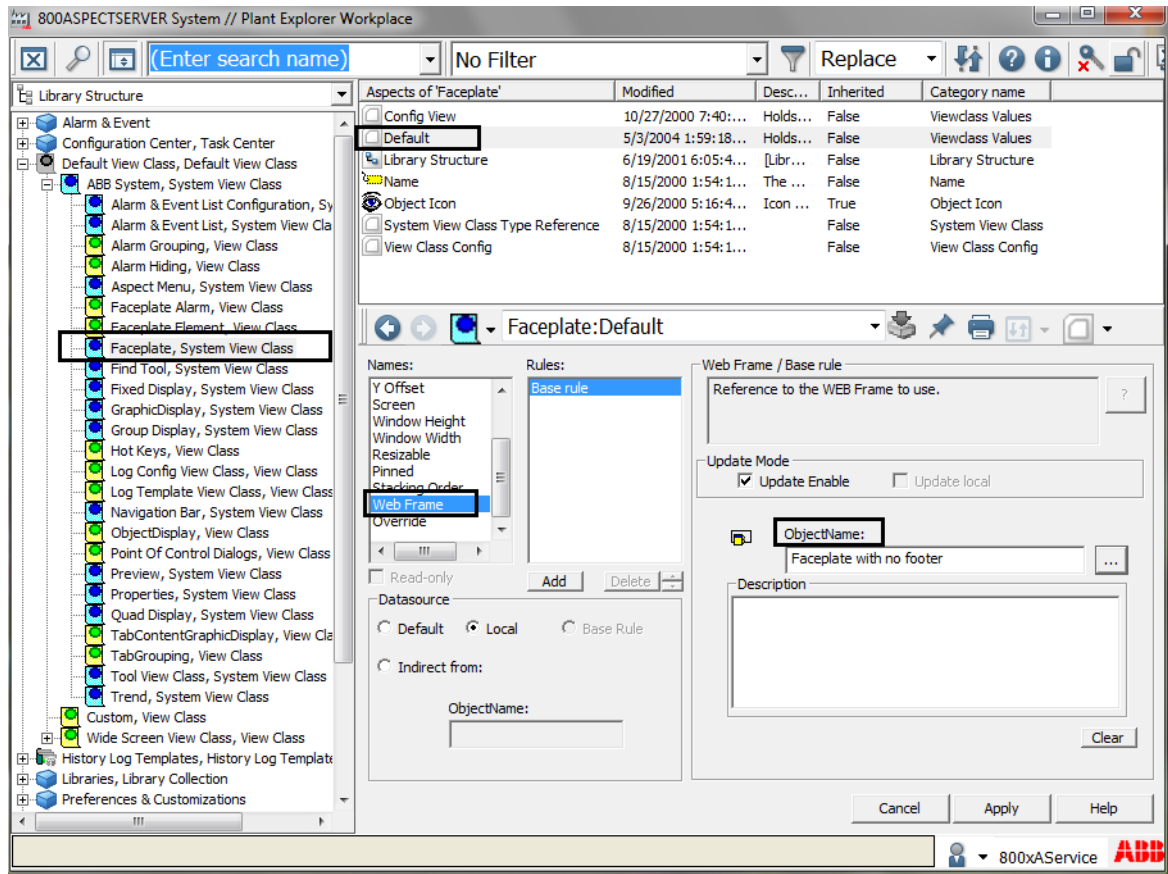
Figure 113. Default aspect of the Faceplate System View Class object

2.   In **Names**, select **Web Frame**.

3.   In **ObjectName**, click **...** to select the respective workplace frame object. The
     Select Object dialog appears.
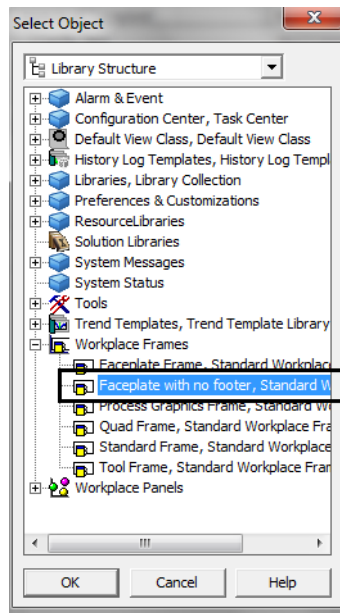
Figure 114. Select object

4.   Select **Faceplate with no footer** in **Library Structure > Workplace Frames** and click **OK**.

5.   Click **Apply** (see Figure 113) to save the changes.

_____

Pin Button

By default the faceplates are unpinned. If the user opens a faceplate and requires to open another faceplate, the first one is replaced with the new faceplate. To make the new faceplate appear in a new window, the user can configure the faceplate to be pinned.

Execute the following to have the faceplates pinned by default:

1.   Expand the **Library Structure**.

2. Select **Default View Class > ABB System, System View Class > Faceplate, System View Class**.

3. Select the default aspect and open the Config view. Select **Pinned** from the list displayed in **Names**.

4. Select **Datasource** as **Local**.

5. Select **Yes** in **Boolean Value**.

It is also possible to set only one faceplate to be pinned. Refer
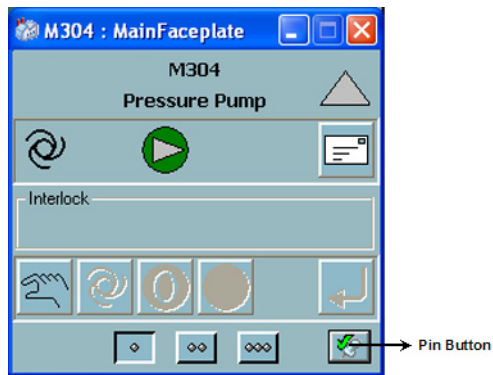
Figure 115. A Faceplate with the Pin Button

The first faceplate on the window is pinned. The next faceplate that is opened is placed in a second window. If a third faceplate should appear on the window, the second faceplate should also be pinned. Otherwise the second faceplate will be exchanged by the third.

Create a New Faceplate

This sections helps the user to create a faceplate. Execute the following steps.

1. Select an aspect object for an object type in the **Object Type Structure**, for example, an aspect object within *Control System Object Type Group*.
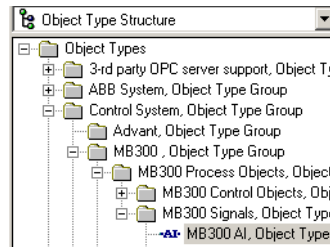
Figure 116. Select Aspect Object

2. Right-click on the aspect and select **New Aspect** from the context menu. The **New Aspect** screen appears.

3. Select **Faceplate / Faceplate PG2** from the aspect list.
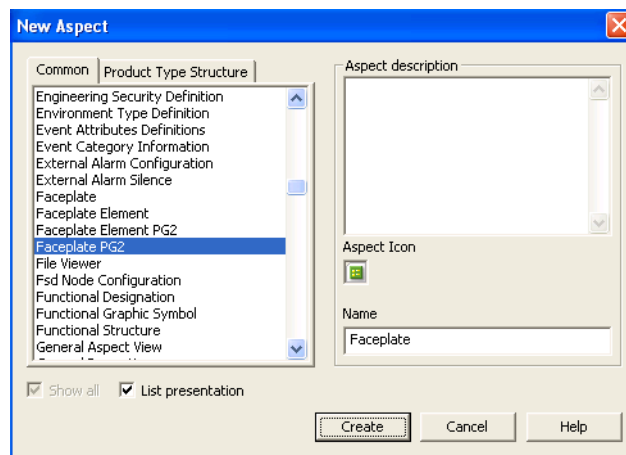
Figure 117. New Aspect Window

> Two faceplate categories **Faceplate** and **Faceplate PG2** exist to support VBPG and PG2 faceplates respectively in the same object type.

4. Click **Create**. The newly created aspect appears in the aspect area.

Configuring the Faceplate

This section describes about configuring the appearance and behavior of three faceplate runtime views. Configuration is done using the *Config view*. Only the user having the privileges of an *Application Engineer* is allowed to configure faceplates.

Config view for faceplates does not have full Unicode support and requires the correct language setting for using Far East languages.

Note that if the faceplate is Inherited, changes can be made only in the object type.

Aspects of 'CP311 Sample'	Modified	Desc...	Inherited	Category name
Alarm and Event List	2000-11-05 10:5...	Alar...	True	Alarm and Event List
Circulation Pump Sample Type Ref...	2000-11-05 11:1...	Circul...	False	Circulation Pump Sa...
Faceplate	2000-11-05 14:3...	Face...	True	Faceplate
Functional Structure	2001-11-28 14:0...	The ...	False	Functional Structure
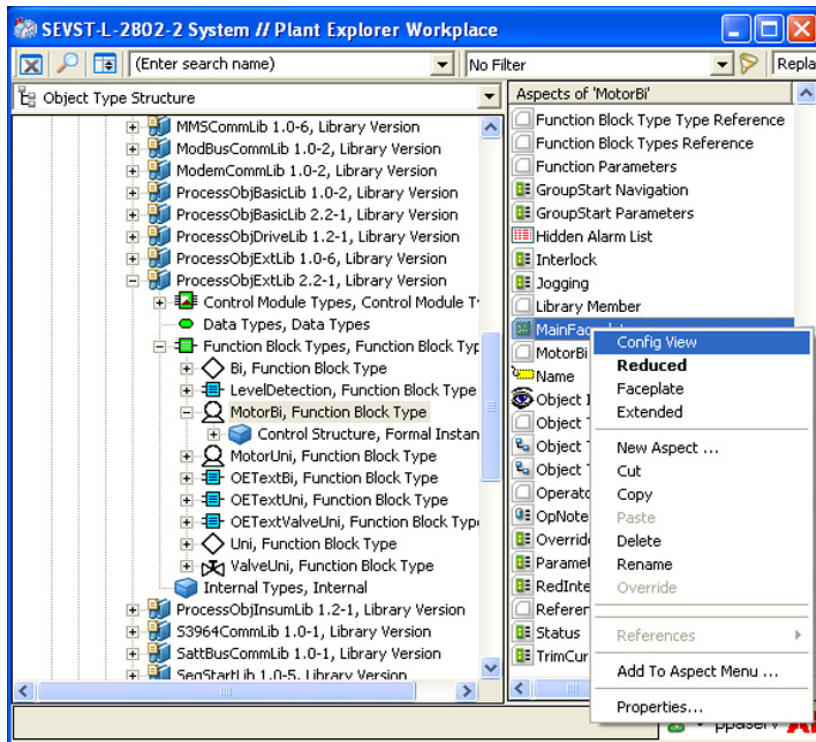
Figure 118. Open Config View

The *Config view* contains the following:

- Layout Configuration of the faceplate geometry.

- Header Configuration of the object lock control and alarm control.

- Indicators Configuration of the Status and Navigation bar.

- Buttons Configuration of the Button area.

- Element Configuration of the faceplate element area.

- Online Help Configuration of the help files.

- Approval or unapproval of faceplates.

Layout Tab

Layout tab contains information on runtime view that should be displayed and the geometrical layout of the views. The faceplate view is mandatory.

Enable/disable of faceplate views requires a restart of the workplace and Operator Workplace.
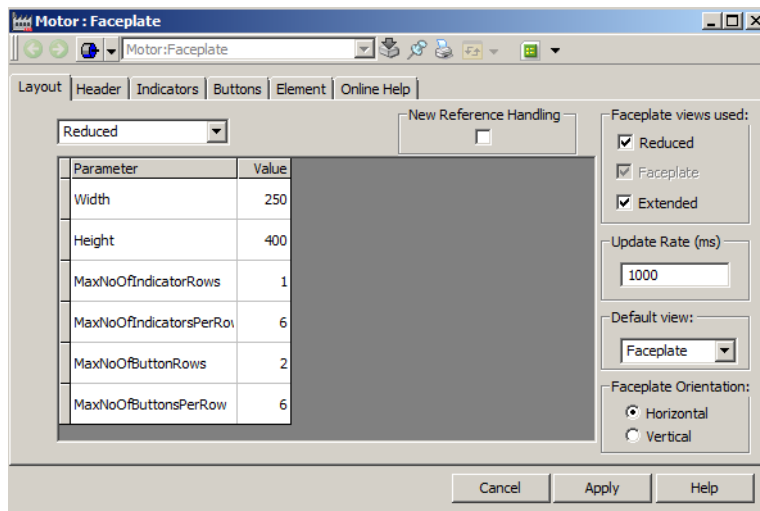
Figure 119. Layout Tab with Default Values

Faceplate view is the default runtime view that is launched when an operator clicks on a graphic element in a graphic display. Once the faceplate is opened this can be changed using the **View selection** buttons.

User can also select the orientation of the faceplate element area.

The default size of the faceplate element aspects are designed using preconfigured values of the faceplate width and height.

Figure 119 shows the configurable parameters in the **Layout** tab.

The update rate for subscribed properties for indicators and buttons are initially set to 1 second. User can configure a new value (in milliseconds) in **Update Rate**.

Header Tab

The override settings for locating the Lock button and the Alarm button is configured in the **Header** tab. These configuration settings are also performance related in case of a search required for finding the Alarm Control or Lock Control aspects. Maintenance work is reduced if the standard aspects are used directly and not using copies of the aspects.
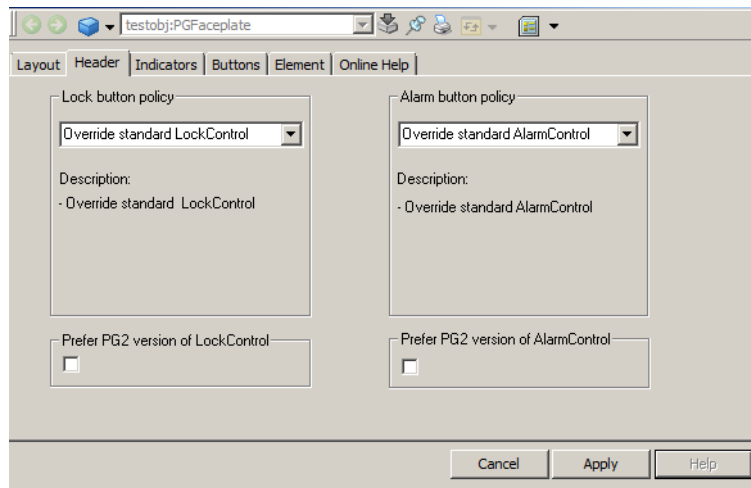
Figure 120. Header Tab

Object Lock Button

The following settings are done in the **Lock button policy** area:

- **Default search order**: Uses the following search order.

    – Override Standard LockControl

    – Legacy lock handling

    – Standard LockControl

    – No Lock button

        This search order is executed sequentially. For example, first the LockControl aspect present on the invocation object is used if it exists. Otherwise, the LOCK property on the invocation object is used if exists, and so on.

- **Override Standard LockControl**: Uses the LockControl aspect present on invocation object, if it exists.

    Select **Prefer PG2 version of LockControl** to use the PG2 lock control aspect if it exists on the object. This is used only if the object includes both VBPG and PG2 faceplates.

- **Legacy lock handling**: Used if the LOCK property exists on the invocation object.

- **Standard LockControl**: Uses the LockControl aspect in the **Graphics Structure**, below **Graphic Tools VB6-> Special**.

- **No Lock button**: No lock button is used.

The user can view a faceplate but cannot operate it for the following scenario:

1) *Lock Service* is enabled.

2) *Default search order* is configured, and search result displays *No Lock button* used.

3) *Operation requires locking* is enabled in Lock Server configuration.

For more information on configuration and usage of the Lock Server, refer to *System 800xA, Administration and Security (3BSE037410*)*.

> ℹ️ If the *Lock Service* is disabled, only a search for legacy lock handling is performed.

Alarm Control Button

The following settings are available in the **Alarm button policy** area:

- **Default search order**: Uses the following search order.

  – Override Standard AlarmControl

  – No Alarm button

  This search order is executed sequentially. For example, first the AlarmControl aspect present on the invocation object is used if it exists. Otherwise, the no alarm button is used.

- **Override standard AlarmControl**: Uses the AlarmControl aspect present on the invocation object, if it exists.

  Select **Prefer PG2 version of AlarmControl** to use the PG2 alarm control aspect if it exists on the object. This is used only if the object includes both VBPG and PG2 faceplates.

- **No alarm button**: No alarm button is used.

- **Standard AlarmControl**: Uses the AlarmControl aspect in the **Graphics Structure > Graphic Tools VB6 > Special**.

Configuration of Alarm Control and Lock Control

To use the standard object lock and alarm control aspects, instantiate the aspect group **AlarmAndLockControl**. These aspects can also be copied manually from the **Aspect System Structure**.

For more information, refer to Lock Control on page 312 and Alarm Control on page 314.

> ℹ️ If the user requires a lock control but does not require an alarm control, then instantiate the aspect group **AlarmControlGroup**.
>
> If the user requires only a lock control, use the aspect group **LockControlGroup**.

To create the **AlarmAndLockControl** aspect, execute the following steps.

1. Add a reference to the aspect group **AlarmAndLockControl** to the object type by creating a new aspect category **AlarmAndLockControl**.
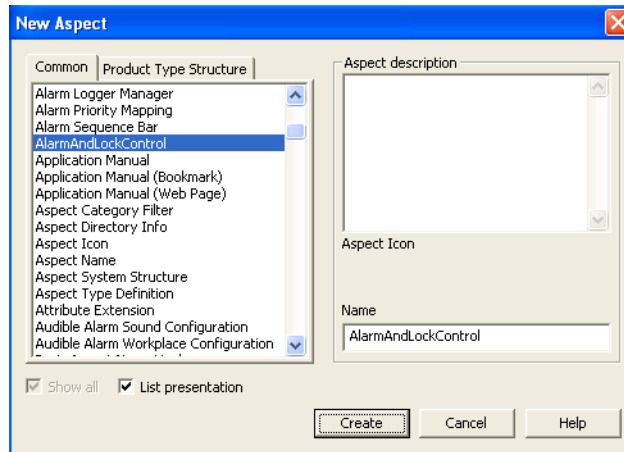
Figure 121. AlarmAndLockControl Aspect

This aspect group contains the **AlarmControl** and the **LockControl** aspects.

> The alarm and lock controls are also available in **Aspect Structure > Aspect Groups > Aspect Group Reference > AlarmAndLockControl**.

Indicators Tab

The **Indicators** tab is used to configure the indicators and aspect links to be displayed in the status and navigation area of the faceplate. The user decides the position to display them, depending on the selected view.
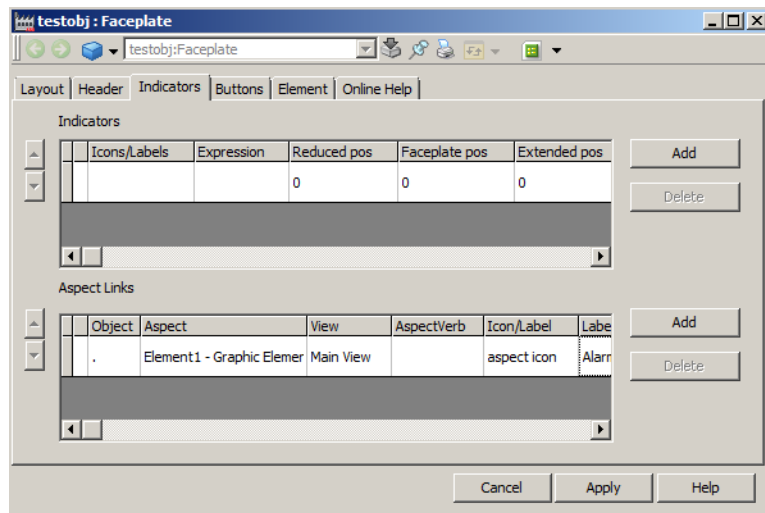
Figure 122. Indicators Tab

Configuring Indicators

Table 80 displays the configurable parameters in the **Indicators** tab that affect the Status and Navigation area in the faceplate.

Table 80. Indicators

Parameter	Description
Icons/Labels	The list of Icons/Labels that represent the result of object properties that are subscribed and specified as expressions.
Expression	A calculation urning an integer value. The result must be in the range from 0 to number of Icon/Labels configured - 1. Refer to Expression Syntax in Faceplates on page 342 for more information.

Table 80. Indicators (Continued)

Parameter	Description
Reduced pos	Specifies the position of the indicator in the Reduced view. Position starts from 1 and is counted from left to right with increasing numbers. 0 = Not used by this view. The exact position of the indicator depends on *No. of indicators per row*, and *No. of indicator rows*
Faceplate pos	The position within the faceplate view.
Extended pos	The position within the extended view.

Click **Icons/Labels** field, and the **Labels** dialog box appears (refer Figure 123). Add the number of icons or labels to be used for each indicator. The values in **Index** is numbered as 0, 1, 2 to N-1.

Click the leftmost column to select an entire indicator row. The user can delete the selected row or order the rows using the spin buttons.
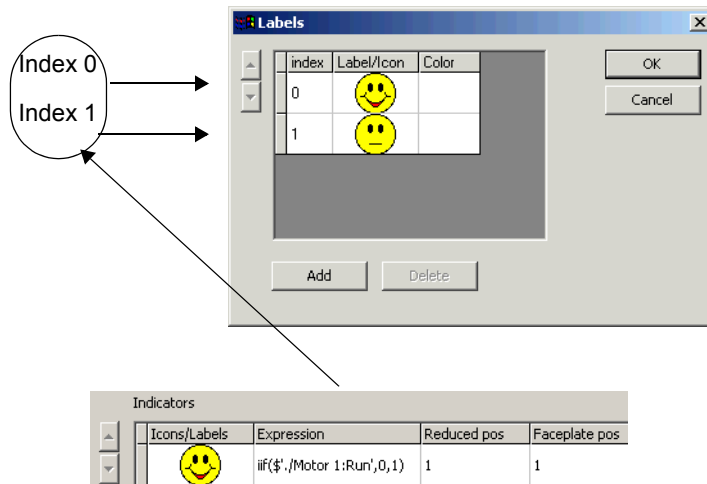
Figure 123. Configuration of Icons/Labels

Indicators are positioned to the left in the **Status** and **Navigation** area.

For example, to configure the first icon/label to be displayed if the value of a property *LIMIT* is below 50, and the second icon/label to be displayed if the value of the same property is above 50, set the expression for the indicator as:

```
iif($'.:Control Connection:LIMIT' < 50,0,1)
```

Figure 124 displays an indicator if the expression for the indicator is wrongly set, or the expression or expression value causes an error condition.

Figure 124. Faceplate with Indicator during an Error Condition

Configuring Aspect Links

Table 81 displays the configurable parameters for the aspect links. Aspect links are positioned to the right in the **Status** and **Navigation** area. An aspect link is a button used as a shortcut to call up any type of aspect as an overlap window.

Table 81. Aspect Links

Parameter	Description
Object	The object of the aspect to be displayed.
Aspect	The aspect to be displayed as an overlap window.
View	View for the selected aspect. If a view is selected, the link cannot have a verb.
AspectVerb	Verb connected to the aspect. If a verb is selected the link can have no view.
Icon/Label	Label, icon or aspect icon to be displayed on the aspect link.
Label color	The text color used for the label.

Table 81. Aspect Links (Continued)

Parameter	Description
Reduced pos	The position within the Reduced view. Refer Table 80 for more information.
Faceplate pos	The position within the faceplate view.
Extended pos	The position within the extended view.
FontSize	The size of the text.

Select Object Dialog window appears on clicking the **Object** column. Select the required object from this screen and click **OK**. If the faceplate is configured in the **Object Type Structure**, user should select only the objects belonging to the same object type.

Click **Icon/Label** column and the **Add Label or Icon** screen appears. Select **Use aspect icon** to get the aspect icon as a graphical representation of the button, or select an icon or specify a label to be displayed.

Refer to *System 800xA, Operations, Operator Workplace Configuration (3BSE030322*)* for more information on Global Verbs.

Buttons Tab

The *Buttons* tab allows the user to configure the buttons to be used in the Buttons area of the faceplate. The behavior of the buttons is specified by the **Action** column and the default value is *SystemDefault*.
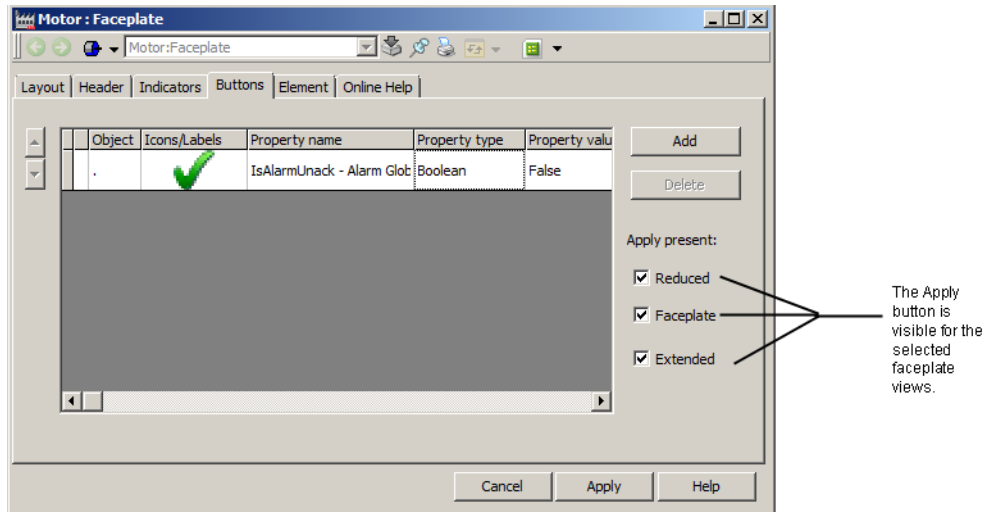
Figure 125. Buttons Tab

Table 82 displays the configurable parameters for the Button area.

Table 82. Buttons

Parameter	Description
Object	The object of the property to be used.
Icon/Labels	Label or icon to be used as caption.
Property name	The object property to which the value should be written when the button is clicked.
Property type	The data type of the selected object property (ReadOnly).
Property value	The value to be written to the property while clicking the button.
Enabled	An expression specifying whether the button is enabled, dimmed or invisible. No expression means that the button is always enabled. [0 = dimmed, 1 = enabled, 2 = invisible].
Tooltip text	Tooltip to be displayed for the button.
Reduced pos	The position within the Reduced view. Refer to Table 80 for more information.
Faceplate pos	The position within the faceplate view.
Extended pos	The position within the extended view.
Action	Sets direct or applied action for the button. Refer to Buttons on page 433 for more information.

More than one button can have the same position. In this case, only one of the overlapping buttons should be visible (that is, only one button can have the expression value of 1 or 0).

Select **Reduced**, **Faceplate**, or **Extended** in **Apply present**, to activate the Apply button for the different views.

The **Apply** button is always located to the right of the button area and cannot be repositioned.

Elements Tab

The elements tab is used to configure the faceplate element area. Aspects can be put directly, or can be organized in tab groups.
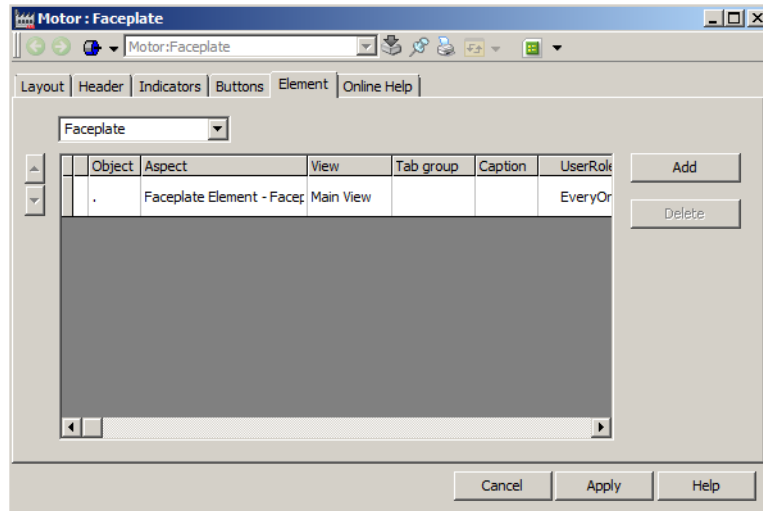
Figure 126. Elements Tab

Faceplate Element aspects are designed to be used here, but any aspect type can be used. A faceplate element aspect has a default size to fit within the faceplate view.

Click the **Aspect** column to select the aspect to be represented as a tab in the faceplate.
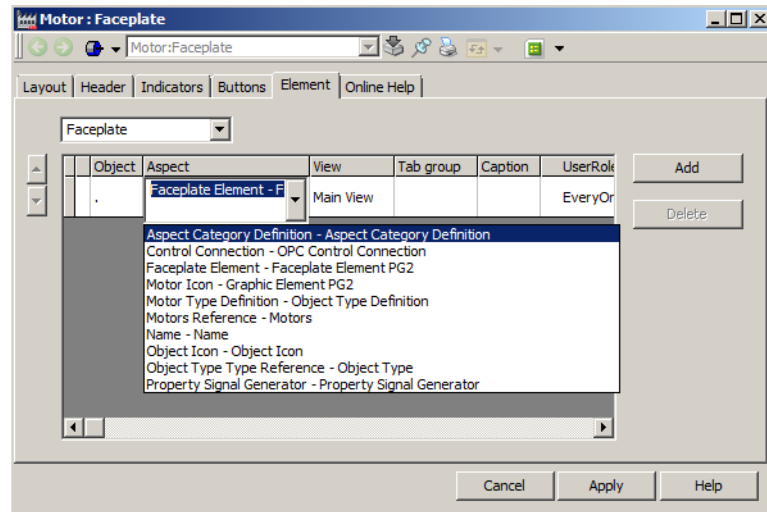
Figure 127. The Aspect Drop Down Menu

Select a view to be displayed, in the **View** column.

Type an identifying string representing the identity of a tab group in the **Tab Group** column. A simple number is suitable because the identification string is not visible to any operator.

For example, enter "1" for the first tab group, enter "2" for the second and so on. The location of the tabs in a group is defined by the order among them and can be ordered using the spin buttons.
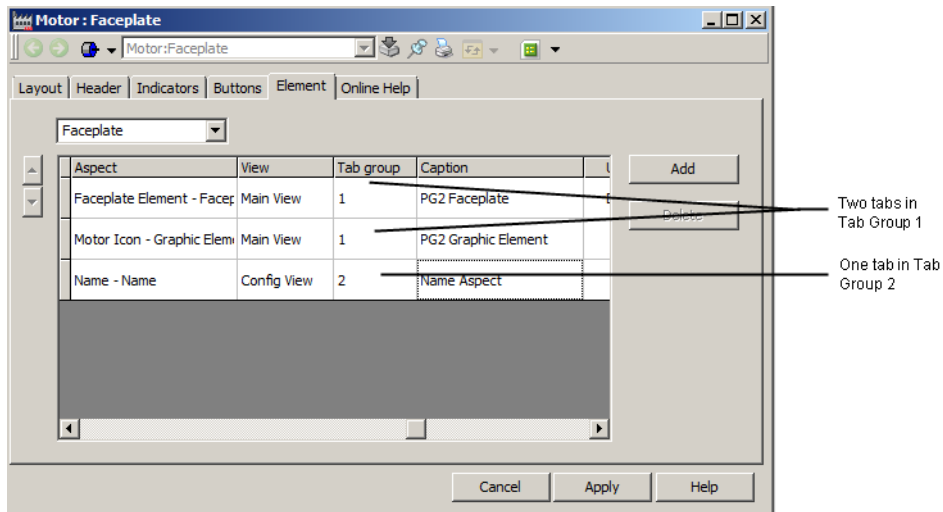
Figure 128. Configuration of Two Tab Groups
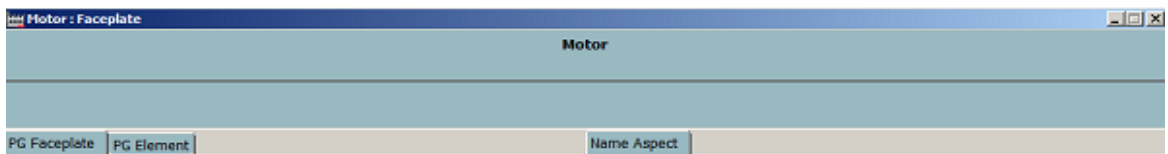
Figure 129. The Two Tabs as Seen in the Faceplate

The following window appears on clicking the **Caption** column. Type name for the tabs and click **OK**.
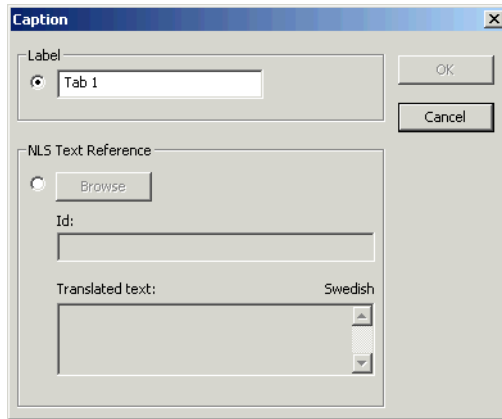
Figure 130. The Caption Dialog Box

User Roles column defines the user group having the privilege to access the tab.
Only the users belonging to the defined group can view the tab.
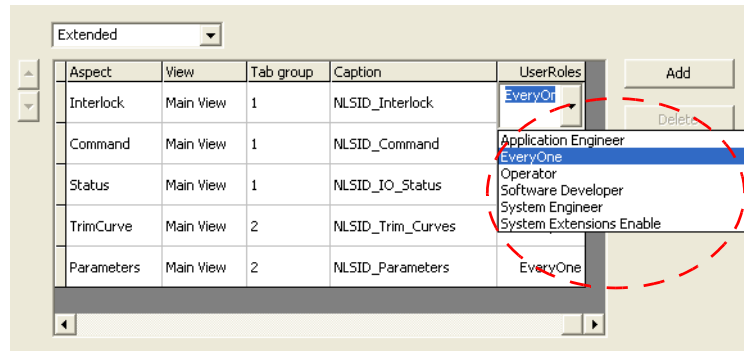
Figure 131. Defining of User Role for Faceplate Tabs

Table 83. The Elements Tab Columns

Parameter	Description
Object	The object of the aspect to be used.
Aspect	The faceplate element aspect (or other aspect) placed in the faceplate element area.
View	Aspect view for the selected aspect.
Tab group	The tab group identity as a string, for example, as a numeric value 0,1, and so on. The group identity is not shown anywhere.
Caption	Name of the tab.
User Roles	User group having access rights for the tab.

Online Help Tab

The Online Help tab defines the help text to be displayed on pressing the **F1** key from the faceplate.
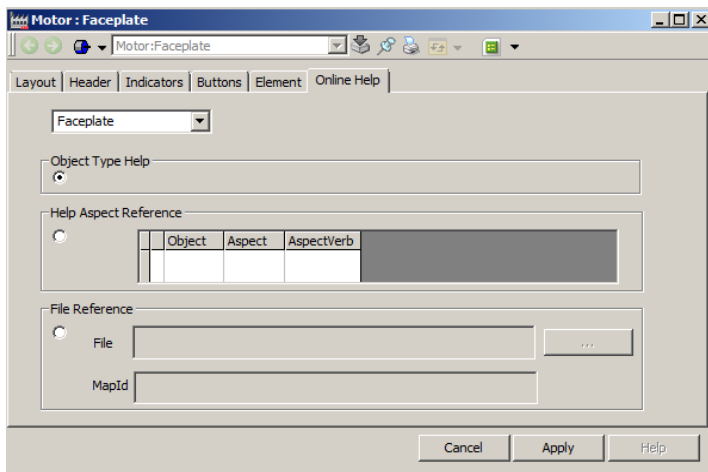
Figure 132. Online Help Tab

This tab contains the following:

- **Object Type Help** - Help for the object type is displayed.

- **Help Aspect References** - Select the object and the help aspect.

- **File Reference** - Specify a helpfile (.chm) and the link to point a specific section in the helpfile.

Refer to *System 800xA, Configuration (3BDS011222*)* for more information on the help aspect types.

Internationalization of Faceplates

Label names, tooltip texts and caption of the tabs in tab groups can be internationalized.

Select **NLS Text Reference** in the window that appears on clicking the **Caption**, **Tooltip text**, or **Icon/Label** columns and click **Browse**.
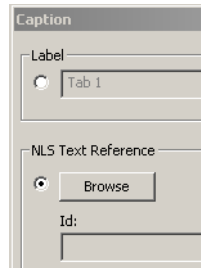
Figure 133. Internationalize Faceplates

The following screen appears on clicking **Browse**.
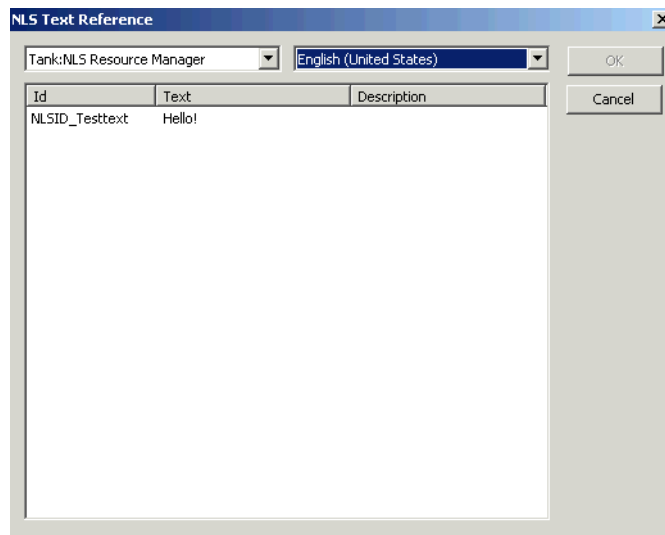
Figure 134. Text Reference

Select the *Resource Manager* aspect to be used. This aspect contains a list of identifiers representing the required text. Select the identifier and click **OK**.
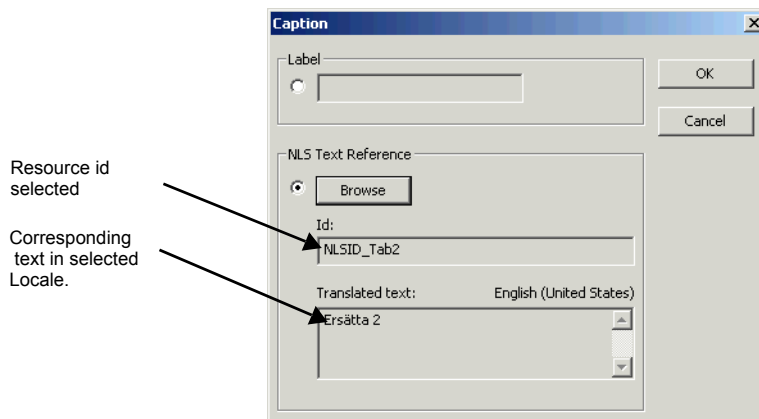
Figure 135. Caption Dialog Box With Domestic Text

Click **OK** and the selected text is changed in the faceplate tab.

Expression Syntax in Faceplates

The following are two types of expressions.

- To select an indicator or to return an integer within the range zero to the umber of configured icon/labels -1.

    Expressions urning other values are invalid and will display a dashed error rectangle.

- To enable or disable buttons.

    The values should be in the interval, 0 = dimmed, 1 = active, or 2 = invisible.

The syntax of an object property reference is:

```
$'ObjectReference:aspect:property'
```

A dot '.' as an object reference indicates the invocation object for a faceplate. For object properties of an object that is a formal instance of a composite object type, the full object path should be given.

```
$'./FormalInstanceName:aspectName:propertyName'
```

The following is an example of expression in faceplates:

```
$'.:Control Connection:VALUE' * 2
```

IIF statement is used for conditional expressions. Following is the syntax.

```
IIF (<logical expression>, <true part>, <false part>)
```

<true part> or *<false part>* can be another IIF statement. *<true part>* evaluates to a value when the *<logical expression>* results into a True value. *<false part>* evaluates to a value when the *<logical expression>* results into a False value.

For example:

```
IIF($'.:Tanklimit:limit'< 100,0,1)
```

Security

Faceplates support security. OPC does the check and confirms that no read or write operations are performed without the user having the required permissions.

Operations

Table 84 describes the two security operations in faceplates. They control the permissions of the user to access different views or configure the aspect.

Table 84. Faceplate Security Operations

Operation	Description
Read	The user has permission to see the main and config views.
Write	The user has permission to modify the aspect

Operation/Permission Mapping on Faceplate a Aspect Category

Figure 136 shows the mapping between operations and permissions in a faceplate aspect category. The user needs read permissions to see the faceplate and configure permissions to modify the faceplate.
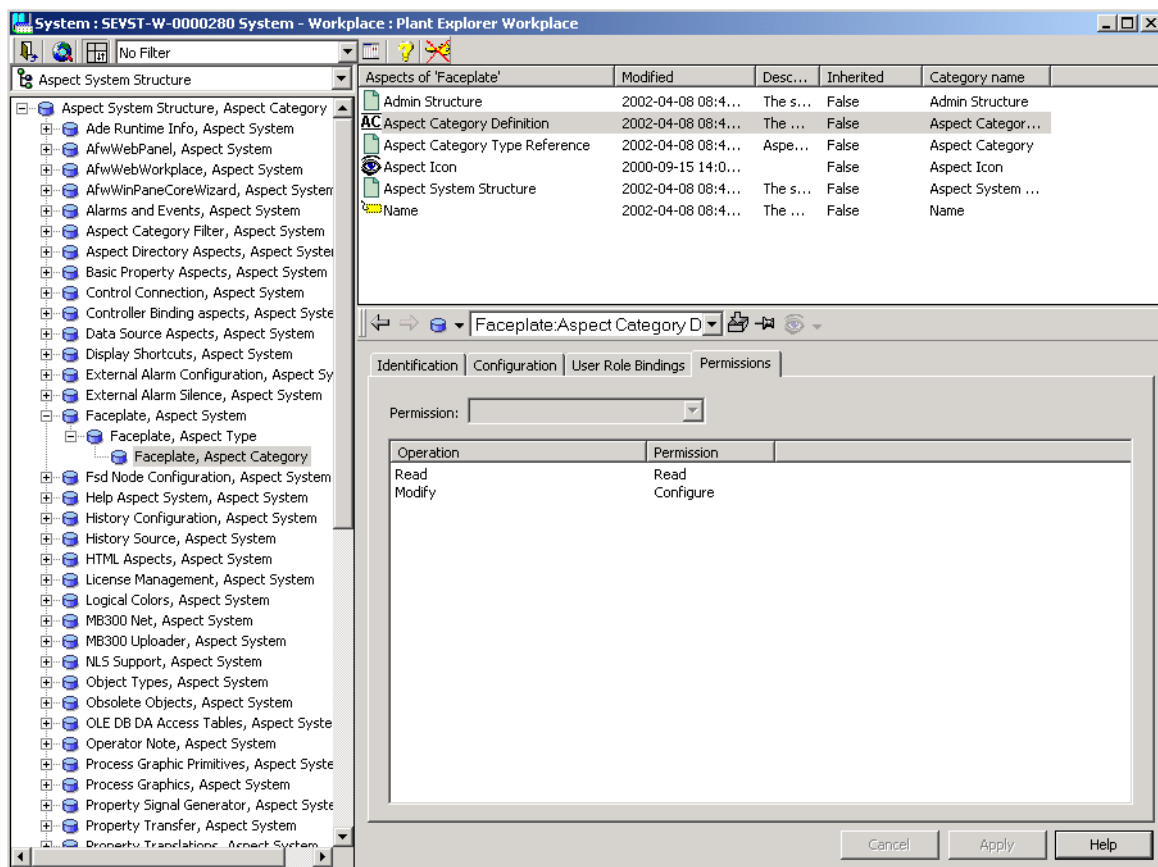
Figure 136. Permission/Operation Mapping on a Faceplate Aspect Category

Section 8 Tutorial

This section provides two tutorials on how to use Process Graphics, using best practices. The following are the tutorials included in this section:

* Basic Tutorial gives an overview of Process Graphics. Process Graphics is used to create graphic aspects.

* Advanced Tutorial describes how to interact with Process Graphics using buttons and input items. This tutorial also describes the procedure to create animations in Process Graphics.

Basic Tutorial

The Basic Tutorial explains the following:

* How to build a Graphic Element.

* How to create and use Input Properties.

* How to create and save a Solution Library.

* How to build a Graphic Display.

* How to create Expressions.

* How to add Graphic Elements to the Graphic Display.

The graphic display built in this tutorial is a tank farm with two tanks as shown in Figure 160. The tanks are connected through a pipe and a valve. The inputs for this tank farm are MV233 (Valve), LI233 (TankLevel), and LI330 (TankLevel). This graphic display includes the flow direction from one tank to the other and also displays a message during the opening of the valve.

The objects and the application behind the Basic Tutorial is not included in the system. Use objects created in the user-defined application.

In this example, the Valve is created in an AC 800M customer library *DemoLib* and the object type is *MyValveUni*. The TankLevel is an object of type *SignalReal*.
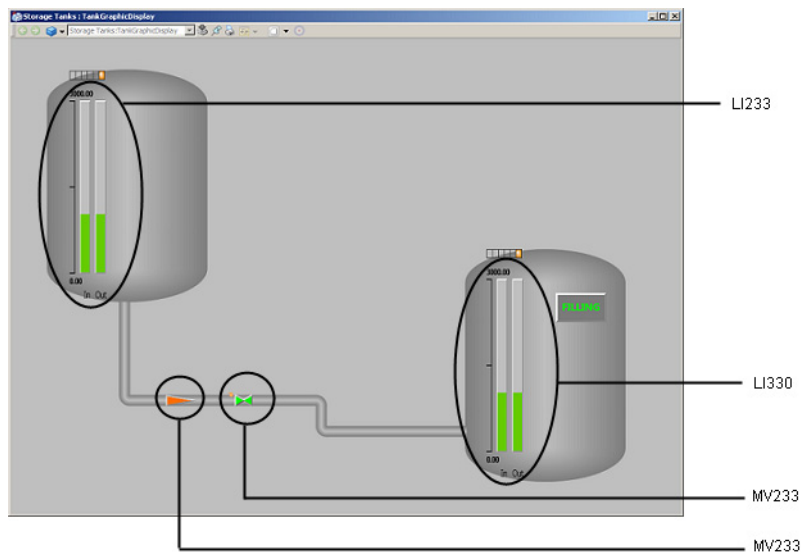
Figure 137. Graphic Display with the Graphic elements

How to build a Graphic Element

This section describes how to create a graphic element and to launch the Graphics Builder for editing.

Execute the following steps to create the **FlowDirection** graphic element.

1. Browse to the AC 800M library *DemoLib* and the object type *MyValveUni* in the **Object Type Structure**.

2.   To create a new graphic element, right-click on the object and select **New Aspect** from the context menu.

3.   Select **Graphic Element PG2** from the aspect list and type **FlowDirection** as the name of the aspect. Click **Create**.
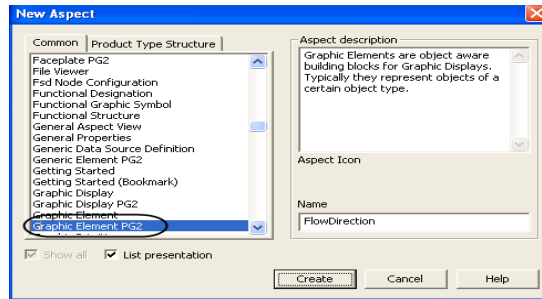
Figure 138. New Graphic Element aspect

4.   To launch the Graphics Builder, right-click on the newly created aspect and select **Edit** from the context menu. The Process Graphics Builder opens.
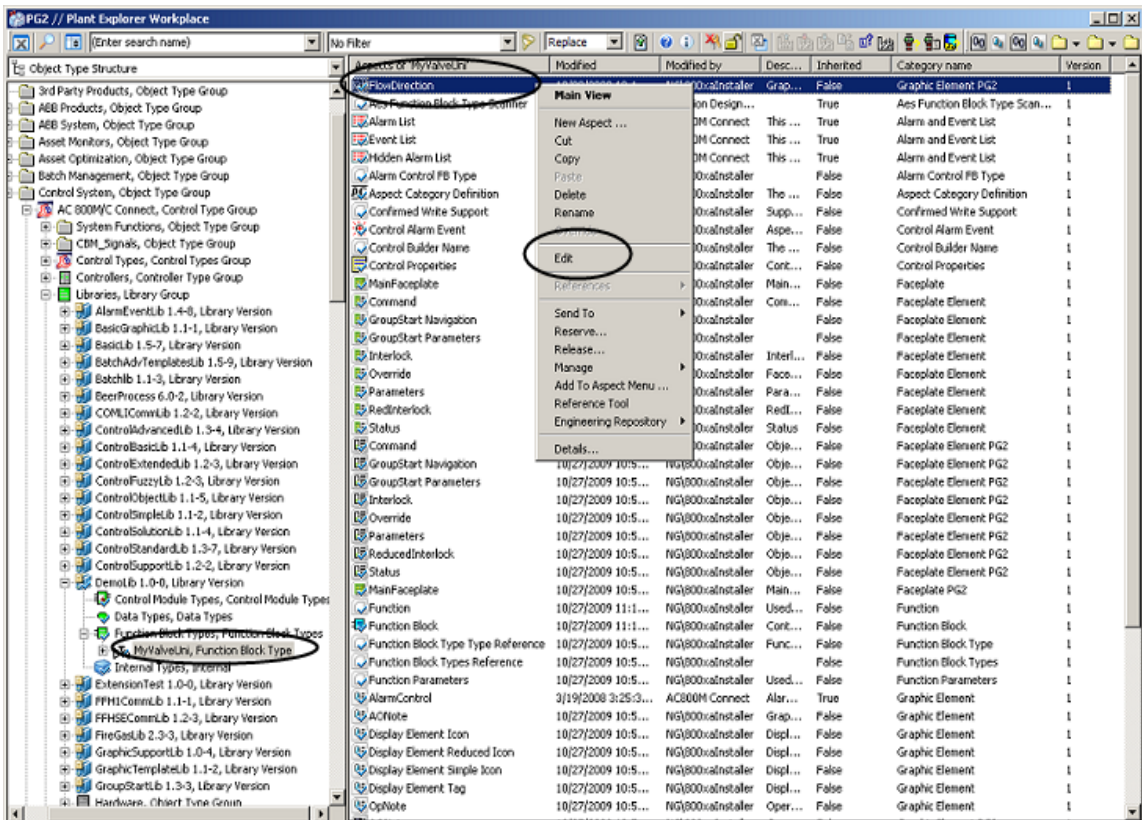
Figure 139. Editing the Graphic Element

5. Select **Triangle** primitive (**Toolboxes > Shapes**). Drag and drop the primitive into the edit panel. Also set the **Direction** to **Right**.

    Change the *Height* and *Width* properties of the element, to a value close to the size that the element will be used with. This will minimize the possible distortion of re-scaling the element.
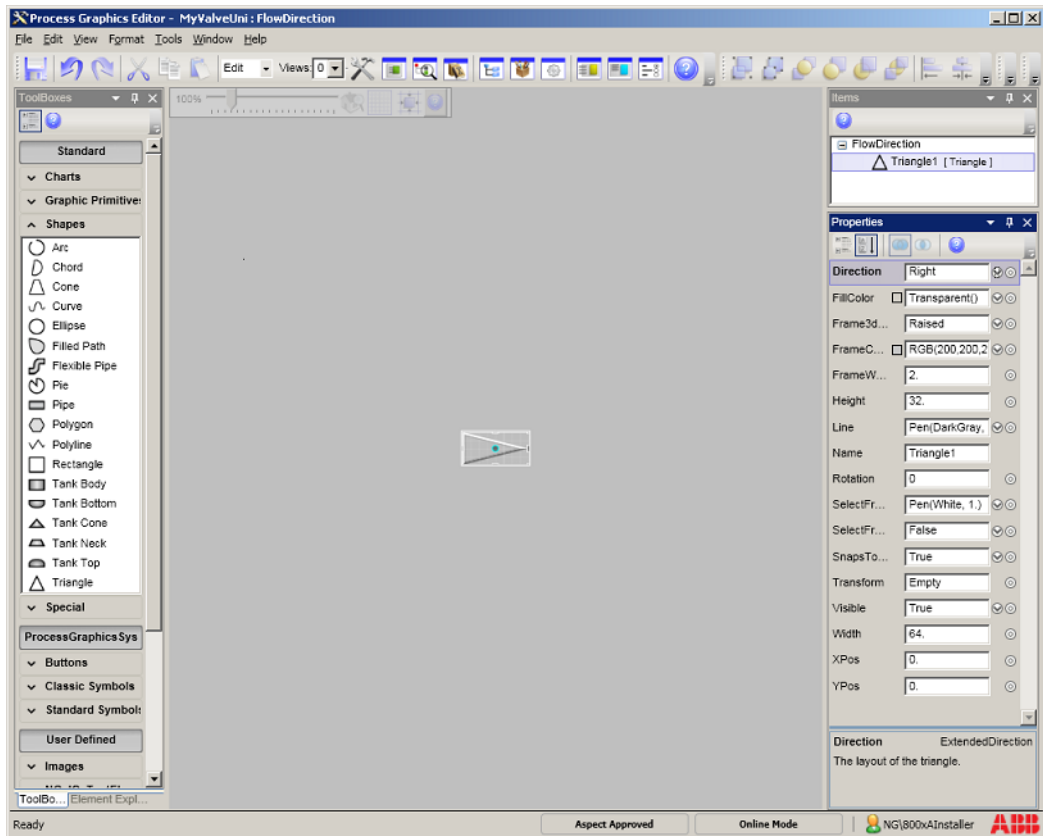
Figure 140. Editing the Graphic Element in Graphics Builder

For more information on adding this graphic element to the
TankGraphicDisplay graphic display, refer to How to add Graphic Elements to
the Graphic Display on page 362.

How to create and use Input Properties

Create an input property for the **FlowDirection** graphic item to change the fill color
of **Triangle** primitive when the valve is open or closed and name it *FillValve*. This

input property will be used only after adding this graphic element to the graphic display. This provides the user a possibility of selecting fill color for the **FlowDirection** graphic element.

To create an input property:

1. Select **View > Input Properties** or click ⊞ in the toolbar.
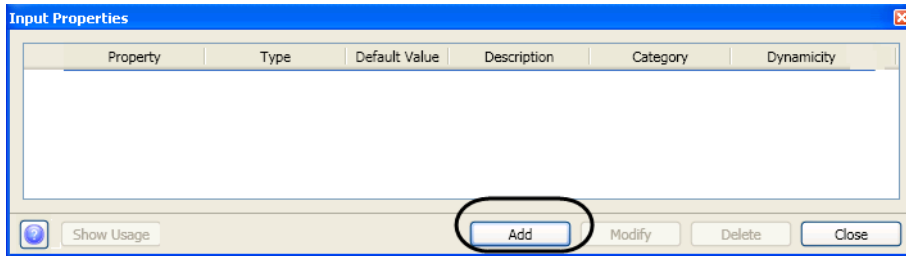
Figure 141. Input Property

2. Click **Add** to display the **Add Input Properties** dialog.

3. Type **FillValve** as the name. Select **Color** as the data type**, Appearance** as category (used as sorting criteria), and **Dynamic** as the dynamicity. Also type a valid **Description** for the input property.

ℹ️   For more information on input properties, refer to Input Properties on page 81.
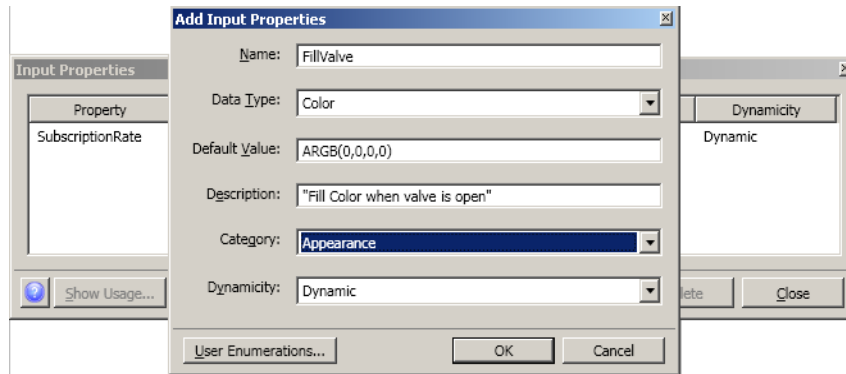
Figure 142. Adding Input Property

4.   Click **OK** to save the input property, and then click **Cancel**.
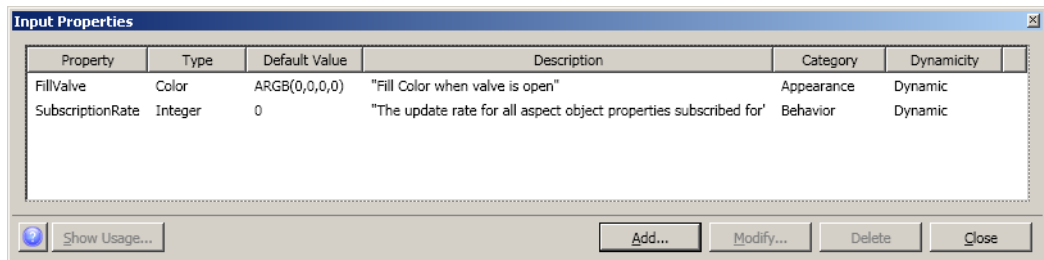
Figure 143. FillValve Input Property

5.   Click **Close** to close the **Input Properties** window.

6.   Select the triangle in the edit panel.

7.   In the **Properties** window, click ⊙ in **FillColor** property to open the
     Expression Editor.

8.   Click in **ProcessData**. Browse to **MyValveUni** in the **Object Type Structure** and double-click **FB1.Value**. This property is *True* when the valve is open.
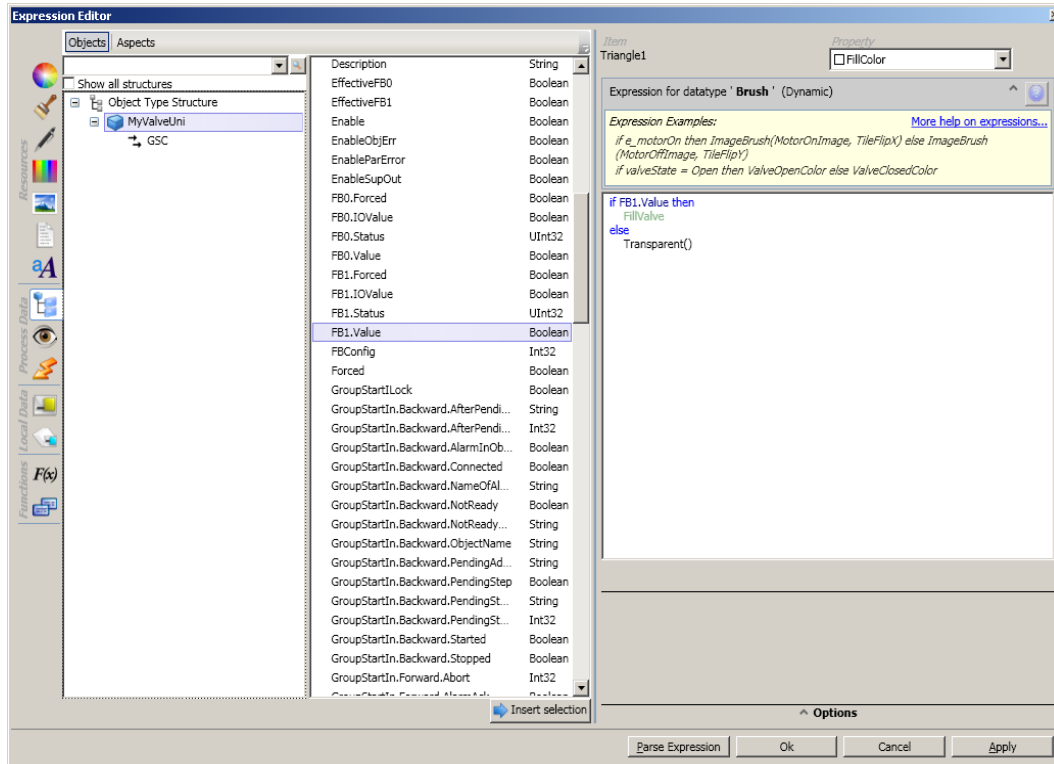
Figure 144. Adding Property Reference to the Expression

9.   Click in **LocalData (Input Properties)** and select the input property required for the expression.
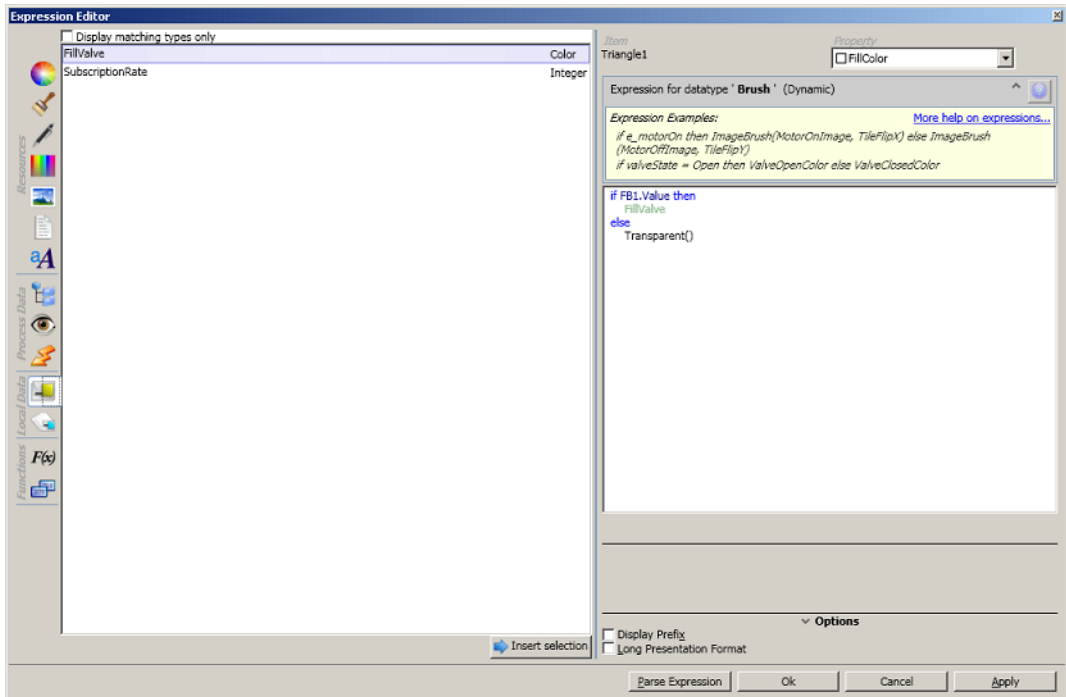
Figure 145. Adding Input Property to Expression

10. Create the following expression.

```
if .:FB1.Value then
     ip::FillValve
else
     Transparent
```

11. Click **Apply** and then click **OK** to save the changes. This expression is displayed in the **FillColor** property.

12. Select **File > Save** to save the changes.

How to create and save a Solution Library

This section helps the user to create a solution library. A solution library **Tank** is created which contains a tank solution that can be reused in any graphic aspects. Execute the following steps to create a solution library.

1. Create a graphic display called **TankDisplay** and launch the Graphics Builder. For more information on creating graphic displays, refer to How to build a Graphic Display on page 355.

2. Drag and drop **Tank Body**, **Tank Top**, and **Tank Bottom** primitives (**Toolboxes > Shapes)** into the edit panel.

3. Move and resize the primitives as shown in Figure 146.

4. Press SHIFT and click **Tank Body**, **Tank Top**, and **Tank Bottom** to select the primitives.

5. Click ▣ to group the primitives.
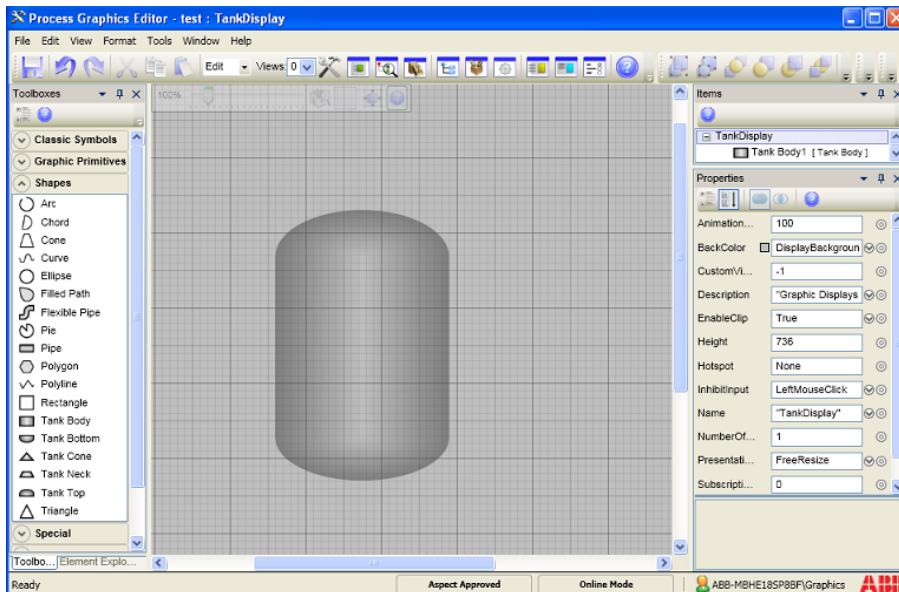
Figure 146. Graphic Display to Create a Solution Library

6.   Select **View > Solution Library**. The **Solution Libraries** window appears.
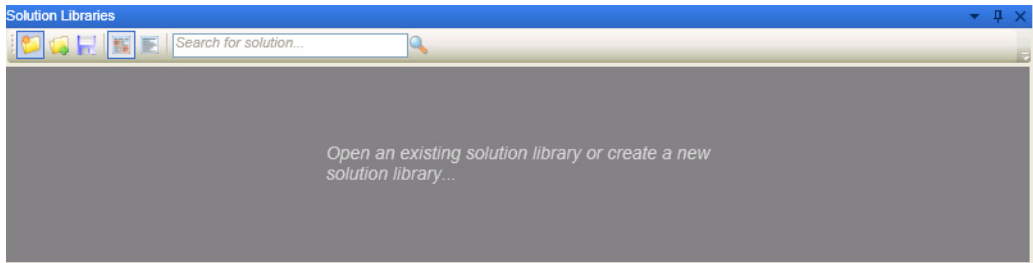
Figure 147. Solution Libraries

7.   Click 📁 to create a new solution library.

8.   Click *New Solution library* and type **Tank** as the name of the solution library.

9.   Right-click on the tank primitive and select **Save as Solution**. This adds the tank solution into the solution library.

10.   Click 💾 to save the **Tank** solution library.

How to build a Graphic Display

This section explains about creating a graphic display named **TankGraphicDisplay**. It also guides the user to launch the Graphics Builder to edit the **TankGraphicDisplay**. Execute the following steps to create this aspect.

1.   In the **Functional Structure**, create an object with the name **StorageTanks**.
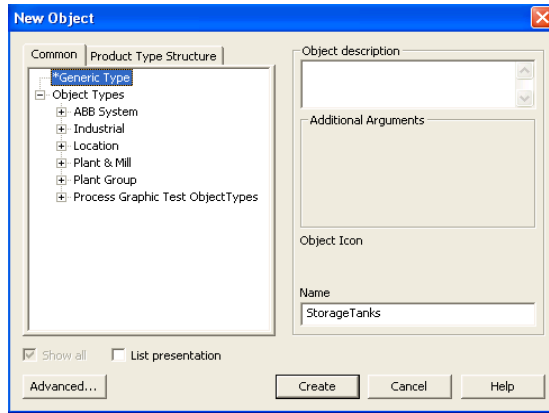
Figure 148. New Object

2.   To create a graphic display, right-click on the object and select **New Aspect** from the context menu. The **New Aspect** dialog appears as shown in Figure 150.
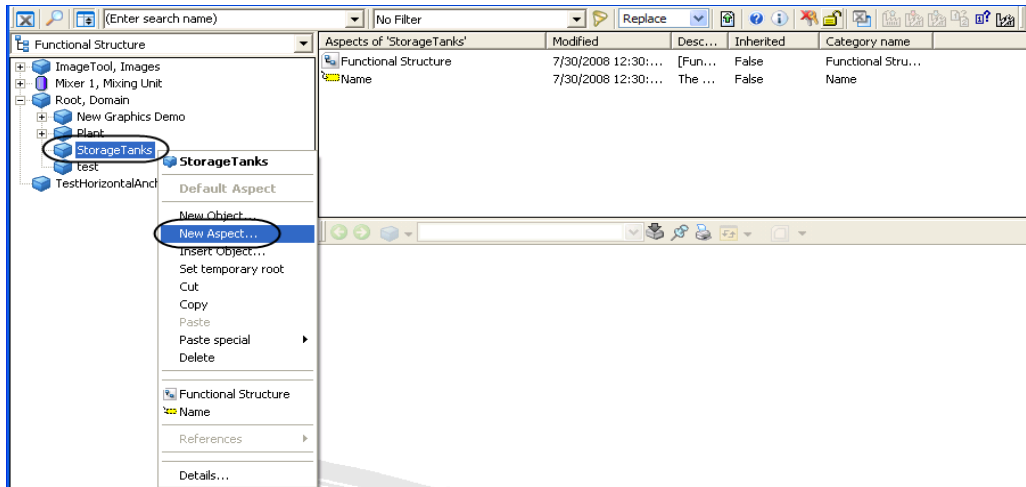
Figure 149. Context menu of StorageTanks object
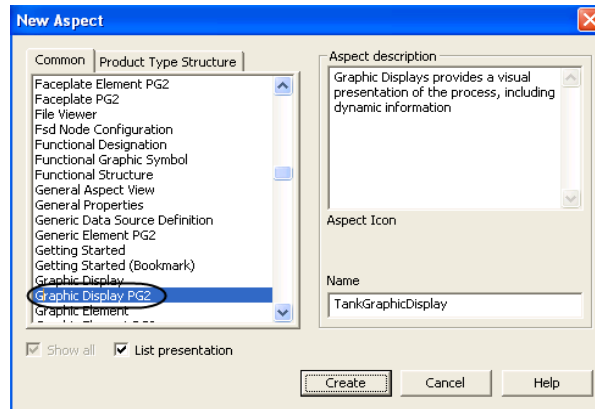
Figure 150. New Graphic Aspect

3. Select **Graphic Display PG2** from the aspect list and type
    TankGraphicDisplay as the name for graphic aspect. Click **Create**.

4. To launch the Graphics Builder, right-click on the newly created aspect and
    select **Edit** from the context menu as shown in Figure 151. The Process
    Graphics Builder opens.
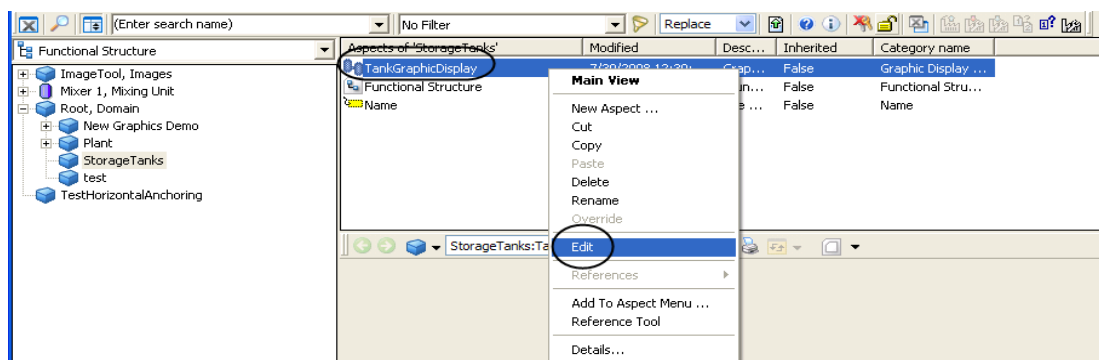
Figure 151. Launching the Graphics Builder for TankGraphicDisplay

5.   Select **View > Solution Library** to open solution libraries window. For more information on creating a solution library, refer to How to create and save a Solution Library on page 354.

6.   Click and select **Tank** to open the solution library.
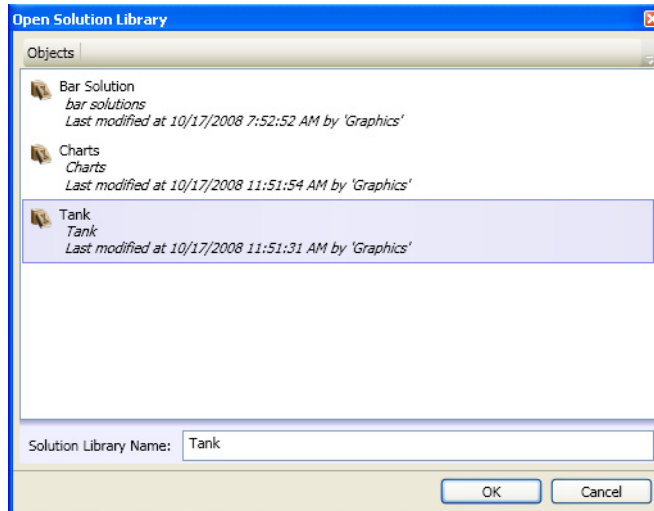
Figure 152. Open a Solution Library

7.   Drag and drop the tank solution two times, from the **Solution Libraries** window into the edit panel. Move the tank solutions to the required positions in the edit panel.

8.   Drag and drop **Flexible Pipe** primitive (**Toolboxes > Shapes)** into the edit panel. Configure the pipe to the required appearance.
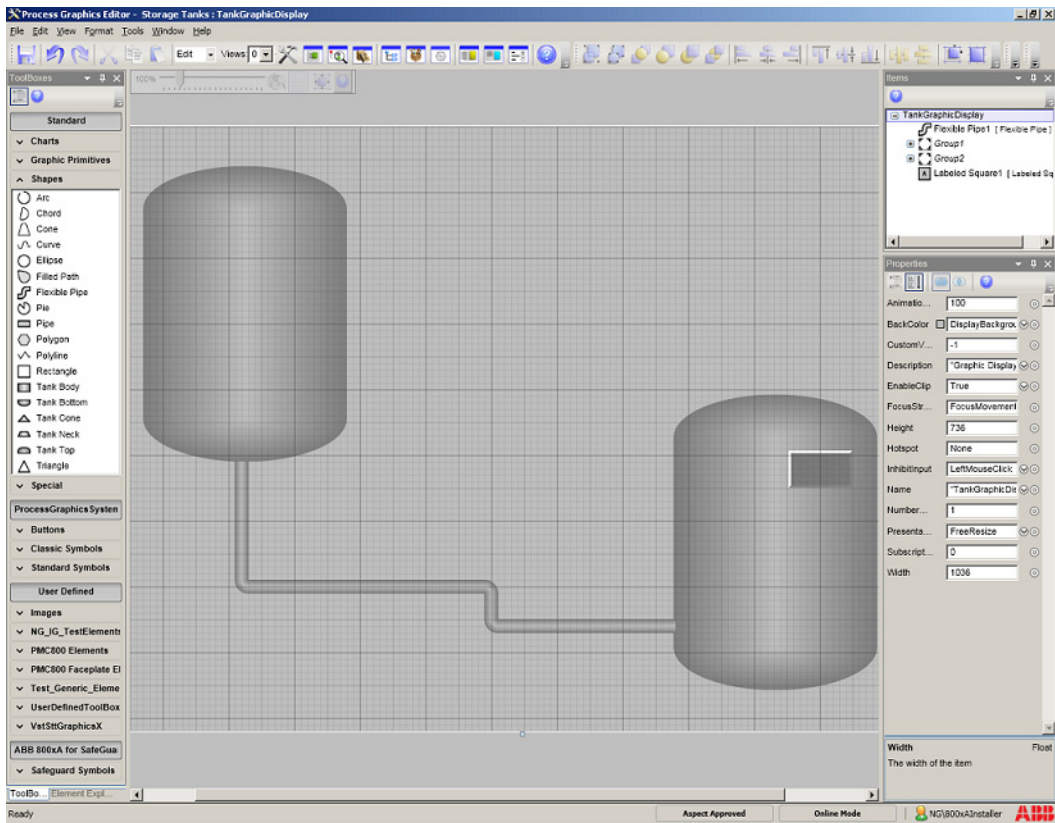
Figure 153. Graphics Builder

9.   Select **Labeled Square** primitive (**Toolboxes > Classic Symbols**). Drag and drop this primitive into the edit panel. Configure this primitive to the required appearance.

If required, resize and group the primitives for the display to appear as shown in Figure 153.

10. Select **File > Save** to save the display.

How to create Expressions

This section describes how to use the **Expression Editor** to assign expressions to the properties of graphic items.

Execute the following steps to create an expression for changing text in the **LabeledSquare** when the valve is open or closed. This expression assigns a text "Filling" to the lower tank when the valve is open.

1. Select **View > Graphic Items**. All the items added to the graphic display appears in **Items** on the right of the edit panel.

2. Select **LabeledSquare** from **Items** to view the properties of this primitive.
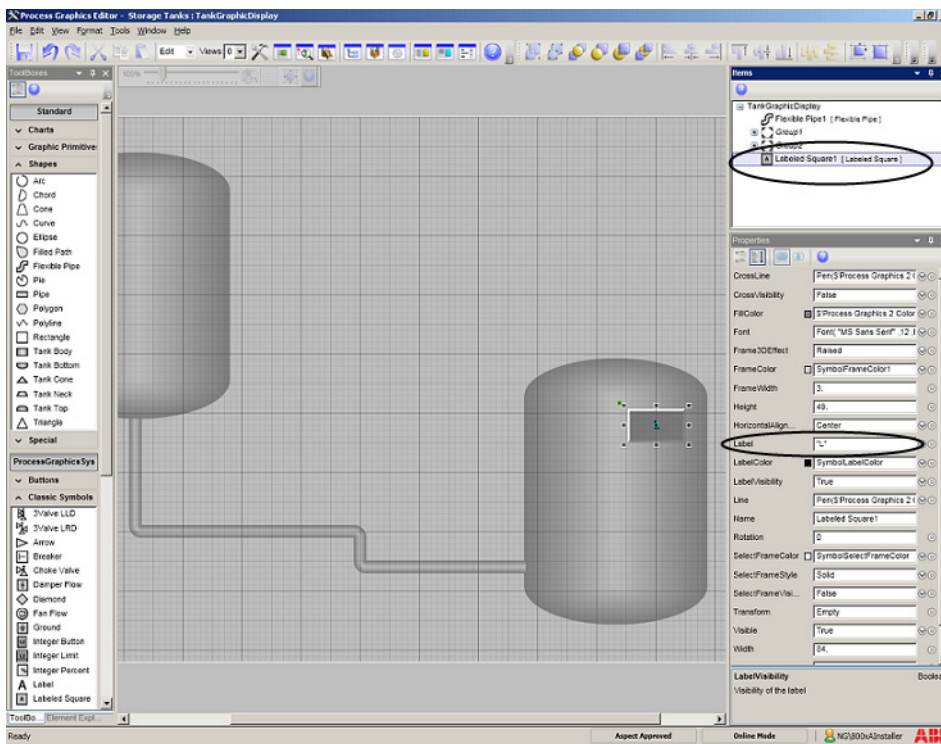
Figure 154. Graphic Items and Property Window

3.   Click ⊙ to the right of input field of **Label** property to open the **Expression Editor**.
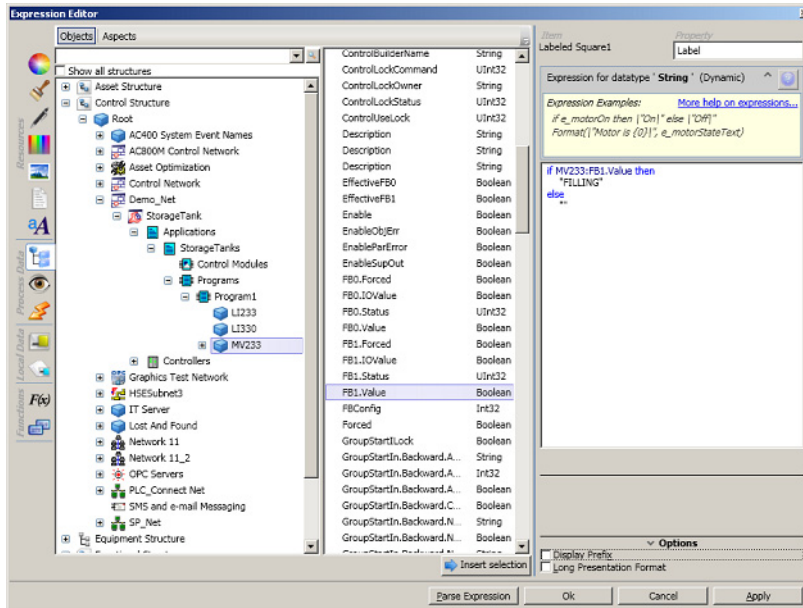
Figure 155. Expression Editor with the Expression

4.   Click the *Properties* tab 📇 in *Process Data*. Browse to **MV233** in the **Control Structure**.

5.   Select *FB1.Value* (this property is True when the valve is open) and create the following expression.

```
if MV233:FB1.Value then
     "FILLING"
else
     ""
```

6.   Click **Apply** and then click **OK** to save the changes. This expression is displayed in the **Label** property.

Changing the Label Color

This section helps the user to set the color to display the value of **Label** property. To set the label color:

1. Click ⊙ in **LabelColor** property.
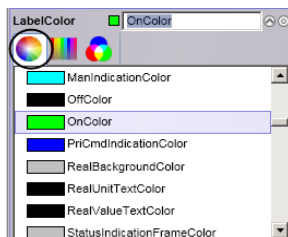
Figure 156. Color Editor

2. Click 🌈 and select a color; in this example the logical color **OnColor** from **AC800M/C Colors**.

3. Select **File > Save** to save the changes.

How to add Graphic Elements to the Graphic Display

Element explorer in Graphics Builder is used for adding one or more graphic elements to a graphic display.

Execute the following steps to add the MV233 (Valve of type *MyValveUni*), FlowDirection, LI233 (SignalReal), and LI330 (SignalReal) graphic elements to the **TankGraphicDisplay** graphic display.

1. Click the **Element Explorer** tab.
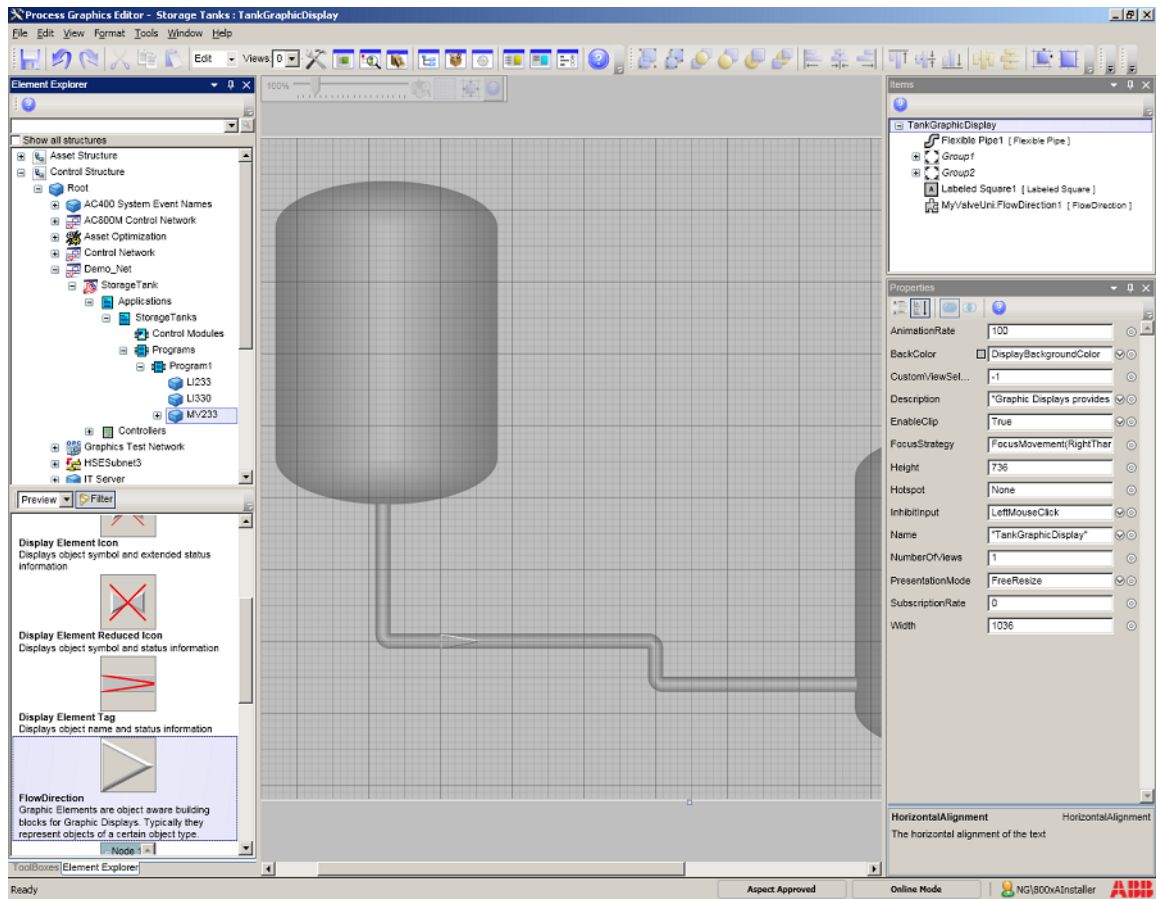
Figure 157. Element Browser

2.   Select the **MV233** object, from the **Control Structure**. Drag and drop the
     FlowDirection element into the edit panel.

     For more information on creating the **FlowDirection** graphic element, refer to
     How to build a Graphic Element on page 346.

3.   Select **LI233** and **LI330** objects from the **Control Structure**. Drag and drop
     the **Display Element Bar** into the edit panel.
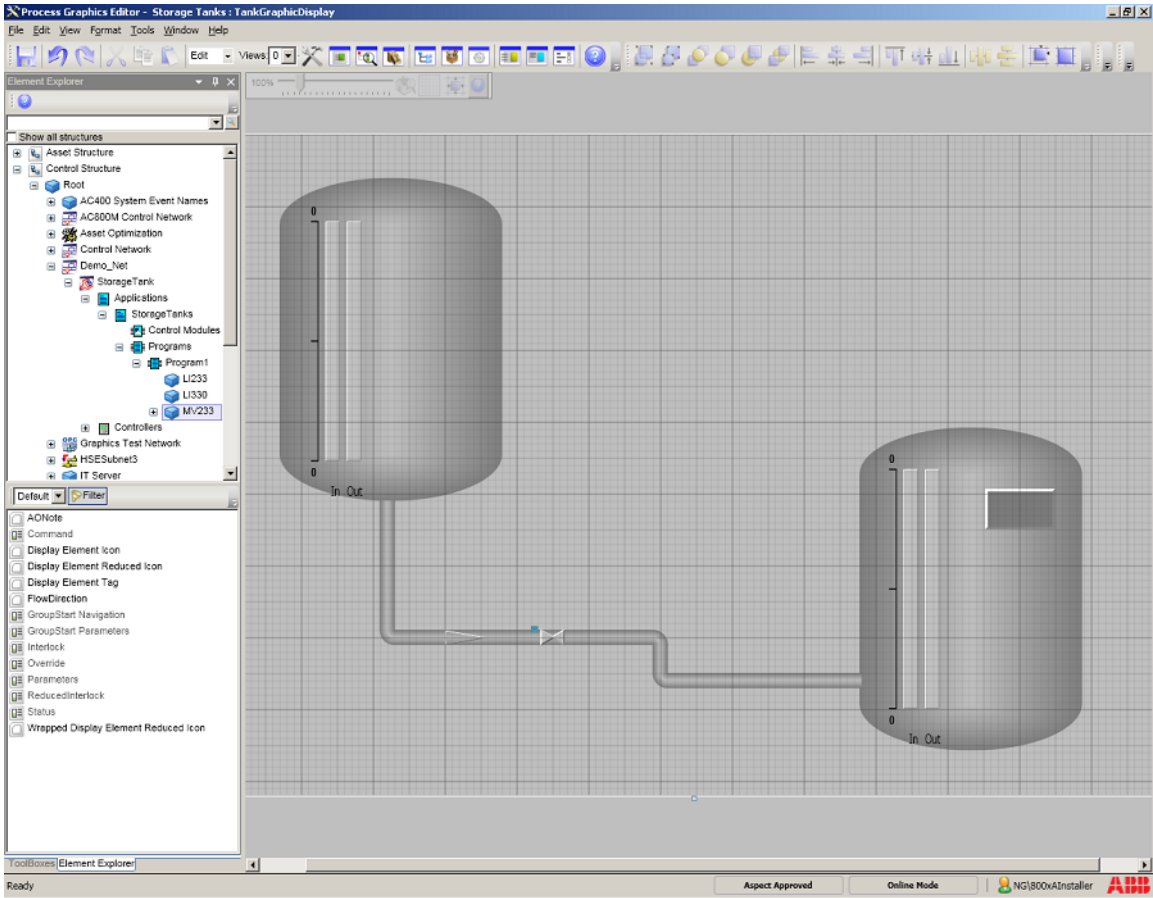
Figure 158. Graphic Display including all the Graphic Elements

Resize all the graphic elements for the display to appear as shown in Figure 158.
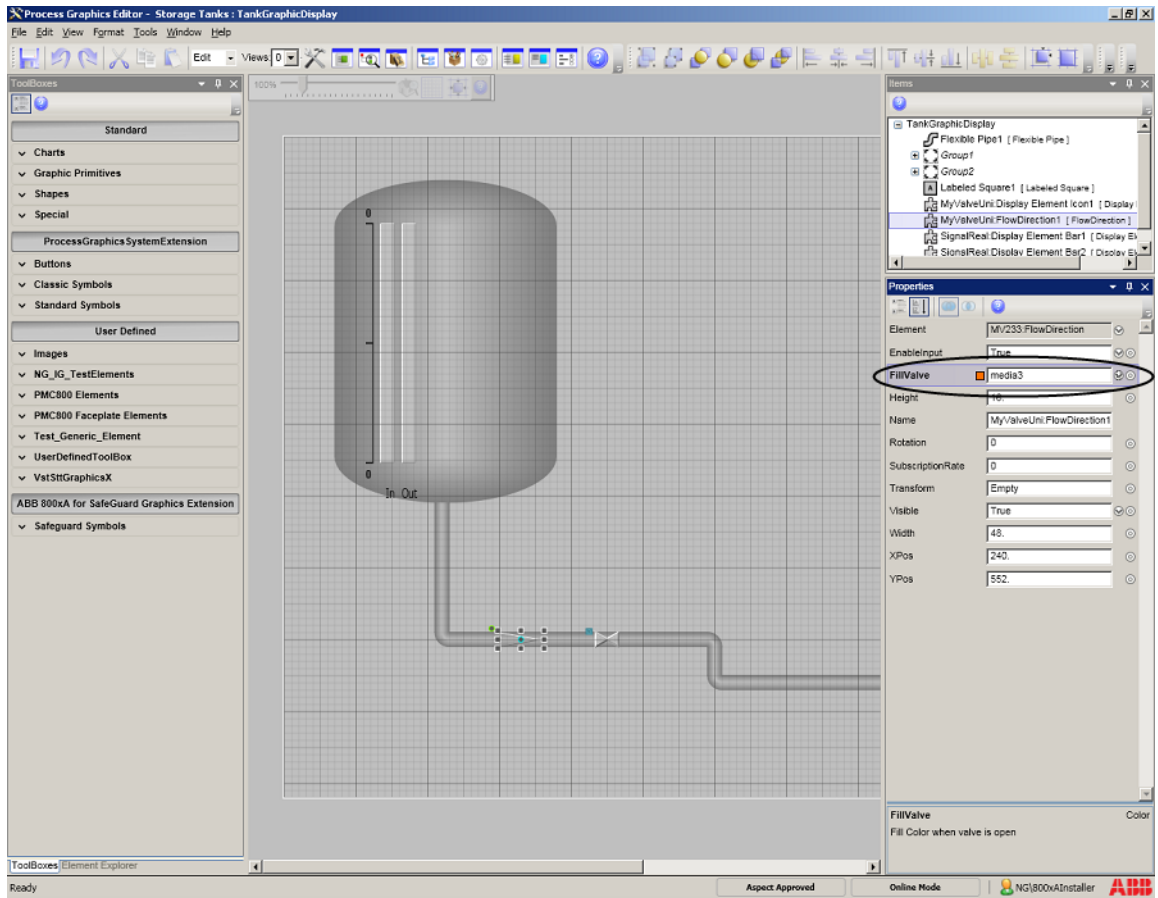
Figure 159. Graphic Display including the Input Property

4.   Select **File > Save** to save the changes.

Figure 160 shows the preview of graphic display when the valve is open.
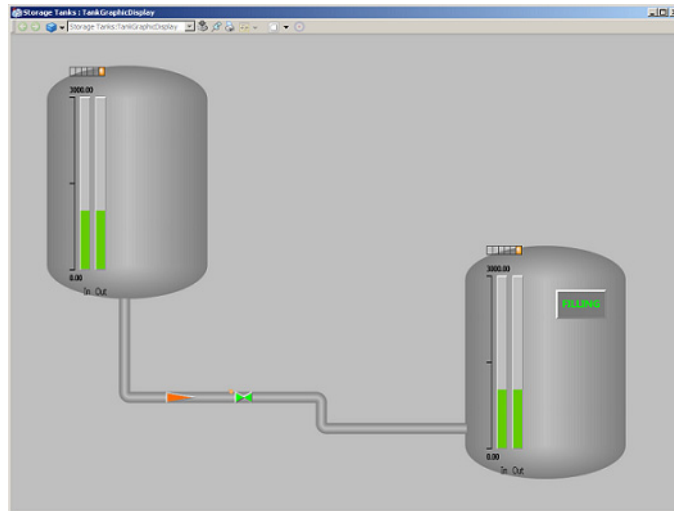
Figure 160. The Graphic Display when the Valve is open

Figure 161 shows the preview of graphic display when the valve is closed.
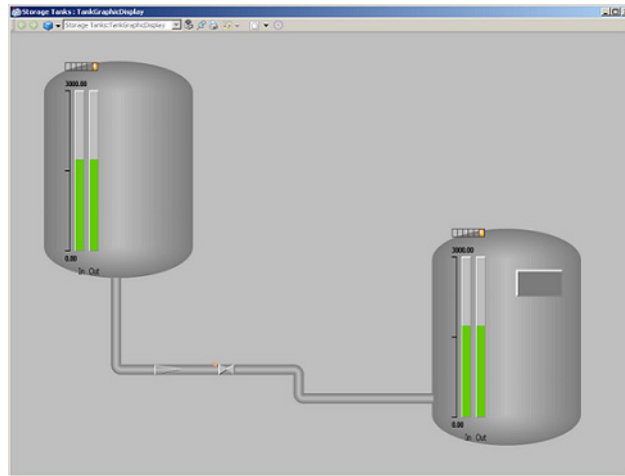
Figure 161. The Graphic Display when the Valve is closed

Advanced Tutorial

Advanced Tutorial explains the following:

- Creating charts
- Configuring buttons
- Configuring drag handles and rotate handles
- Using late binding
- Using list/combo boxes
- Using the MultiSelectionDew
- Creating animations

ℹ️ Manually install the demo feature. Execute the batch file
LoadNewGraphicsDemo.bat located in the folder **C:\ Program Files\ ABB
Industrial IT\ Operate IT\ Process Portal A\ demo**.

The demo appears as **New Graphics Demo** in **Functional Structure > Root Domain**.
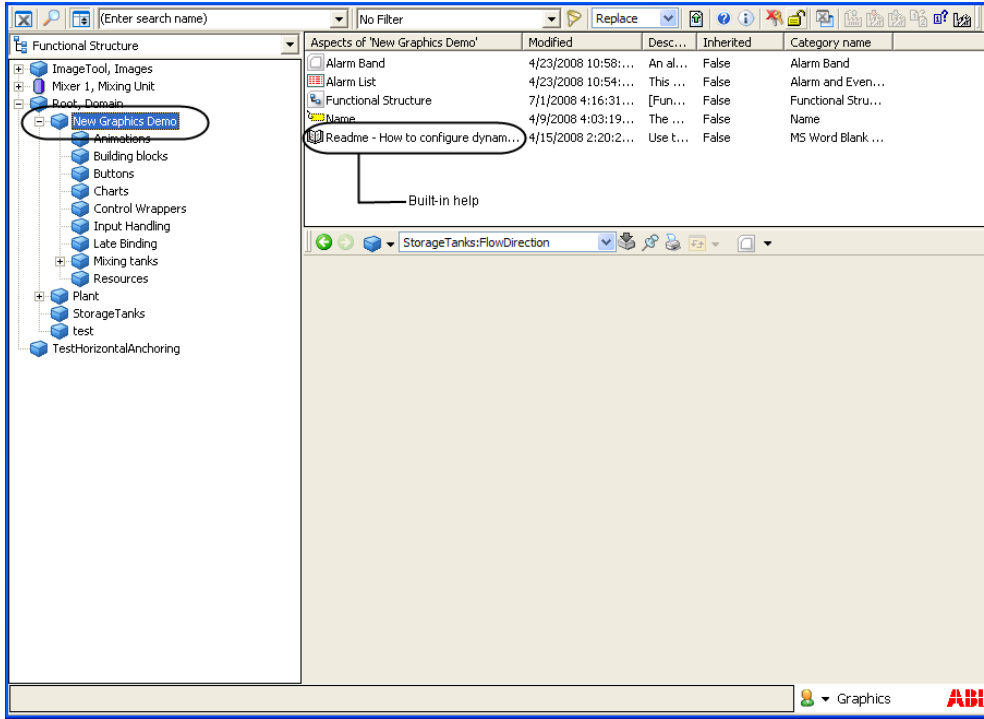
Figure 162. Demo Graphics in Functional Structure

> **ℹ** For information on configuring dynamic data for all aspects, read the built-in online help in **Demo Graphics**. Refer Figure 162.

This tutorial describes the examples in **Demo Graphics**.

For more information on the properties of different charts, handles, and buttons, refer to Appendix A, Standard Building Blocks.

For more information on assigning property values to expressions, refer to Expression Editor on page 64.

For more information on adding graphic items to the edit panel and modifying the properties, refer to Section 2, Graphics Builder.

How to configure and use Charts

Configuration of XYGraph, Trend, XYPlot, and RadarChart, included in the **XY, Radar, and Trend** graphic display (in **New Graphics Demo > Charts)** are described in this section.
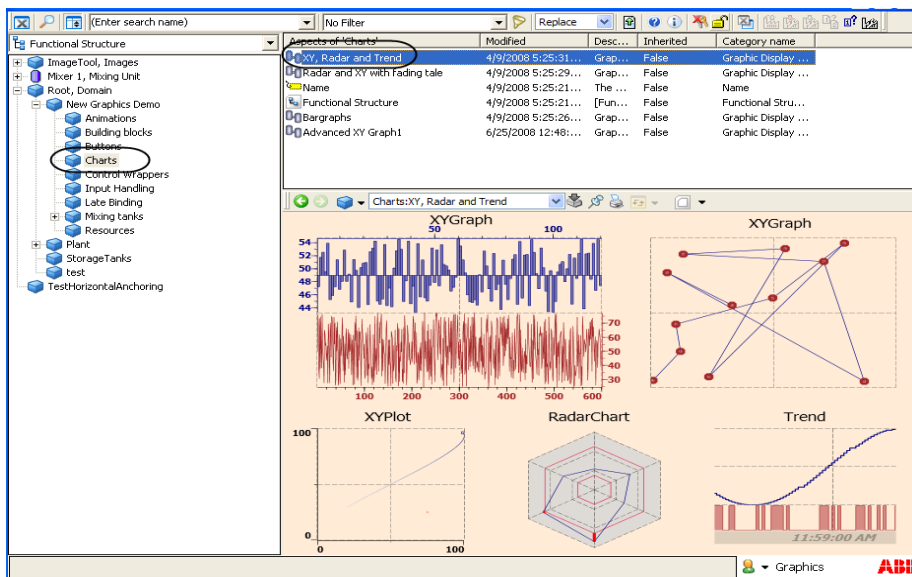
Figure 163. Charts

Figure 164 shows the main view of the **XY, Radar,** and **Trend** display.
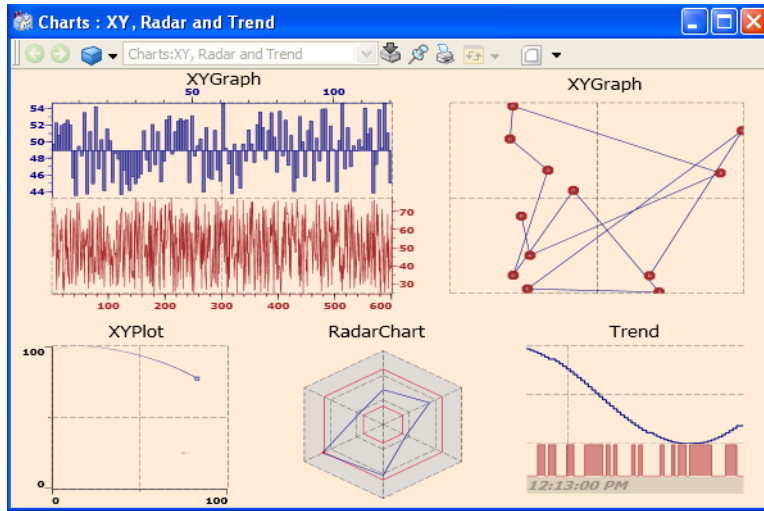
Figure 164. Main View of Charts

XYGraph

Execute the following steps to configure this XYGraph.
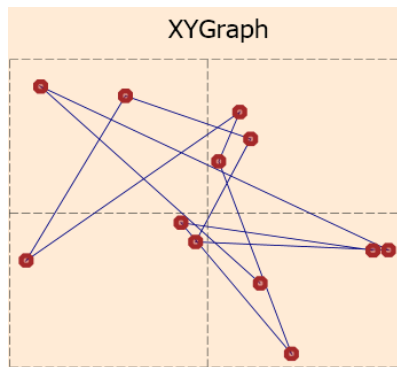
1. Select **XYGraph** from **Charts** in the toolbox. Drag and drop this chart into the edit panel.

2. Assign values to the properties as mentioned below.

   – **NumberOfTraces** = 2

   – Set a **RealArray** property value for **Trace01CurrentValue**, **Trace02CurrentValue**, and **xAxisValueArray** using the Expression Editor.

   – **Trace02DrawMode** = Point

   – **xAxisValueArrayInUse** = True

XYPlot

Execute the following steps to configure this XYPlot.
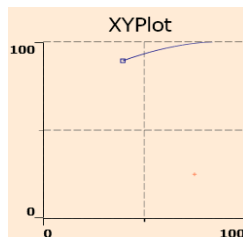
1. Select **XYPlot** from **Charts** in the toolbox. Drag and drop this chart into the edit panel.

2. Select **Scale Vertical** and **Scale Horizontal** from **Graphic Primitives** in the toolbox and place it in the edit panel.

3. Assign values to the properties of **XYPlot** as mentioned below.

   – **MarkerSize** = 5

   – Set aspect property values for **xAxisValue** and **yAxisValue** using the Expression Editor.

   – **xMinValue** = 0, **yMinValue** = 0

   – **xSetPointValue** = 75, **ySetPointValue** = 25

RadarChart

Execute the following steps to configure this RadarChart.
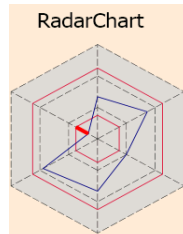
1.   Select **RadarChart** from **Charts** in the toolbox. Drag and drop this chart into the edit panel.

2.   Assign values to the properties of **RadarChart** as mentioned below.

   –   **NumberOfRadarSpokes** = 6

   –   Set aspect property values for the current value of all the spokes. (For example, **Spoke01CurrentValue**).

   –   **Spoke01HighLimit** = 75, **Spoke01LowLimit** = 25, and **Spoke01MinValue** = 0. Enter the same values for the remaining five spokes.

Trend

Execute the following steps to configure this Trend.
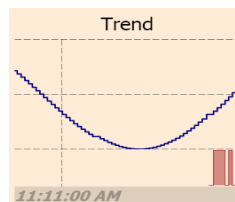
1.   Select **Trend** from **Charts** in the toolbox. Drag and drop this chart into the edit panel.

2.   Assign values to the properties of **Trend** as mentioned below.

- **NumberOfTraces** = 2

- Set aspect property values for **Trace01CurrentValue**, **Trace02CurrentValue**.

- **Trace01DrawMode** = Stepped, **Trace02DrawMode** = Filled

- Set **Trace01MinValue** and **Trace02MinValue** as 0.

- **Trace01NormValue** = 50

How to configure and use DragHandle and RotateHandle

This section guides the user for using DragHandle and RotateHandle input items.

Configuration of the graphic elements **DragHandleH** and **RotateHandle2** (**New Graphics Demo > Input Handling)** is described in this section. Adding these graphic elements to the **DirectEntryWindowTest** graphic display shows the working of drag handle and rotate handle.
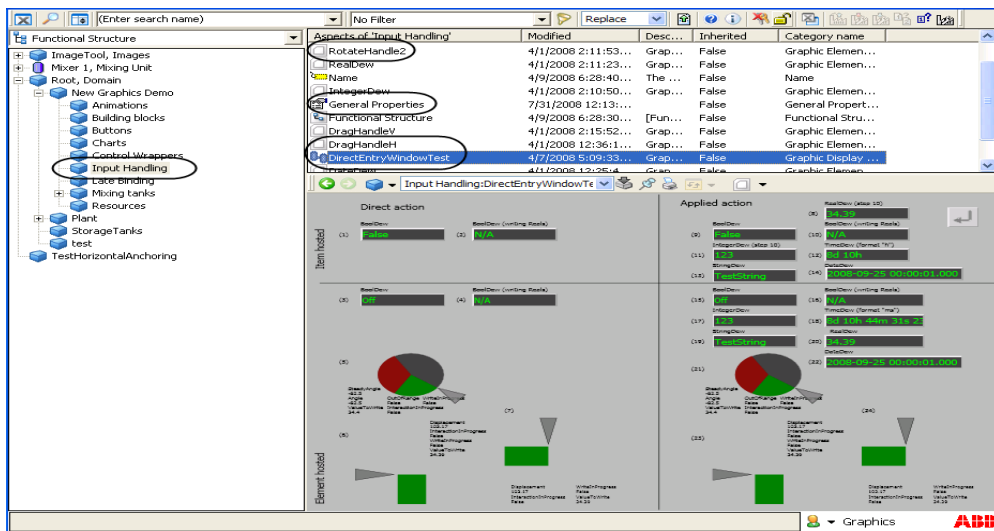
Figure 165. Input Items

Figure 166 shows the main view of the **DirectEntryWindowTest** display.
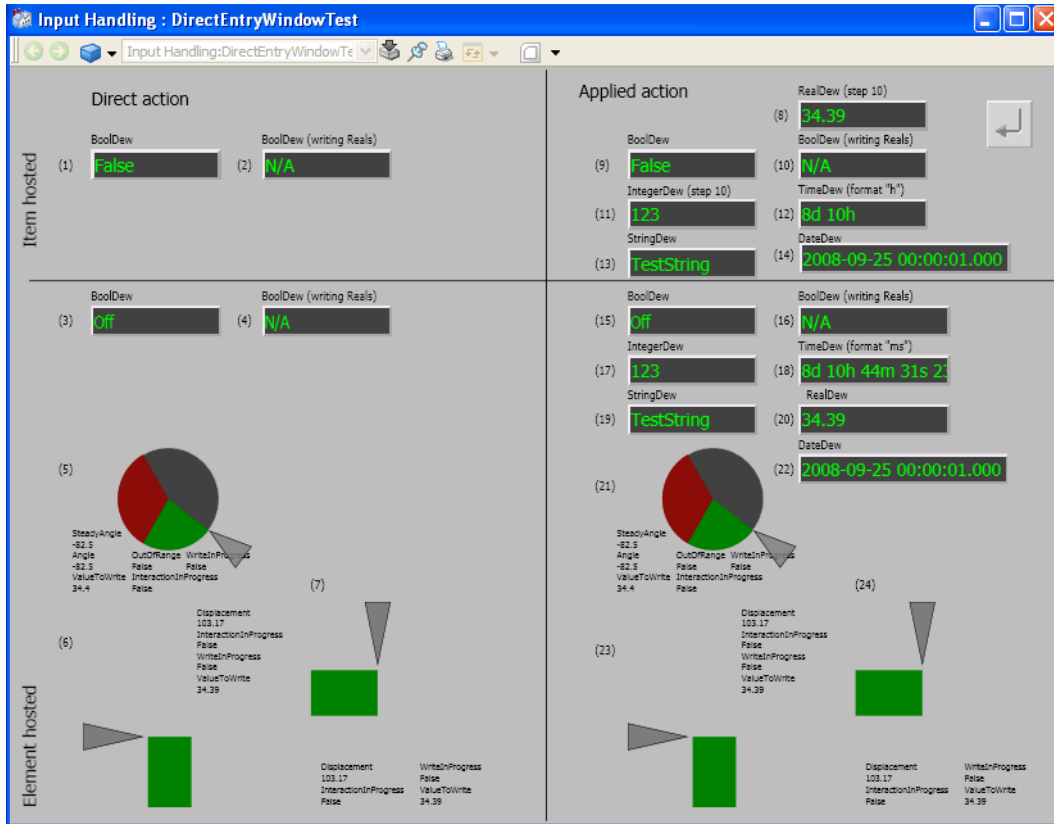
Figure 166. Main View of Input Handling

RotateHandle

This section helps the user to create **RotateHandle2** graphic element and add it to **DirectEntryWindowTest** graphic display.
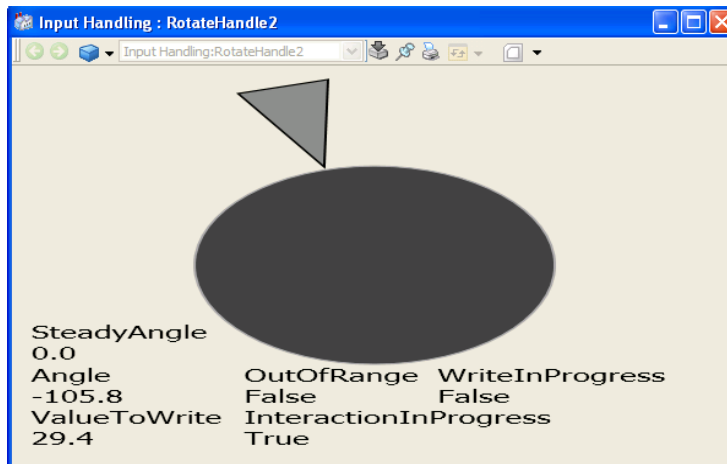
Figure 167. Rotate Handle

The graphic element shown in Figure 167 contains input properties where the user sets the maximum and minimum value, start angle and stop range for rotation. It also contains text items displaying the values of out terminals of this input item during the rotation of the handle.

For more information on out terminals and properties of rotate handle, refer to Input Items on page 488.

In this example, user can rotate the handle around the ellipse. The pie primitives indicate minimum and maximum limit of values for the rotation. The red pie indicates a range where the values are out of range. If the rotation is done within the minimum and maximum limit, it is displayed in Green.

Execute the following steps to configure the **RotateHandle2** graphic element.

1. Configure **InhibitInput** and set it to **All**.

2. Assign **EnableClip** to **True**.

3. Drag and drop the **Ellipse** and **Polygon** primitives (**Toolboxes > Shapes**) into the edit panel.

4. Drag and drop two **Pie** primitives (**Toolboxes > Shapes**) into the edit panel.

5. All three primitives should have the same height and width. Set the **Height** and **Width** properties to 150.

6. Select **Format > Order > Send Backward** to keep the Ellipse on top and Pie behind the Ellipse.

7. Drag and drop six **Text** primitives (**Toolboxes > Graphic Primitives)** into the edit panel.

8. Create the following input properties.

    –   **Action** of data type *Action* and a default value of *SystemDefault*.

    –   **EndAngle** of data type *Real* and a default value of *360*.

    –   **StartAngle** of data type *Real* and a default value of *0*.

    –   **MaxValue** of data type *Real* and a default value of *100*.

    –   **MinValue** of data type *Real* and a default value of *0*.

    –   **PropRef** of data type *PropertyRef*.

    For more information on creating input properties, refer to Input Properties on page 81.

9. Add the **RotateHandle** input item to the graphic element. Click on the element background to get **ElementHosted** as the only choice.

    For more information on adding input items, refer to Input Items on page 488.

10. Assign values to the properties of rotate handle.

    –   **ClickTargetItem =** Name of the polygon

    –   **MaxValue =** *MaxValue* (input property)

    –   **MinValue =** *MinValue* (input property)

    –   **Range =** *StartAngle - EndAngle* (input properties)

    –   **Target =** *PropRef* (input property)

    –   **PivotX =** 150 and **PivotY =** 150

    –   Assign this expression to the **Value** property. Use **Real** function to convert the value of the property to **Real** data type.

Real (*PropRef*#Value)

11. Assign values to the properties of **Pie1**.

    – **FillColor** = Green

    – **StartAngle** = *StartAngle* input property

    – Assign this expression to the **StopAngle** property.

    *StartAngle + (EndAngle-StartAngle) * (Real(PropRef#Value) -*

    MinValue) / (MaxValue - MinValue)

    where *StartAngle*, *EndAngle*, *PropRef*, *MinValue*, and *MaxValue* are input
    properties.

12. Assign values to the properties of **Pie2**.

    – **FillColor** = Red

    – **StartAngle** = *EndAngle* input property

    – **StopAngle** = *StartAngle* input property.

13. Assign this expression to **Transform** property of polygon.

    RotateAt(150,150, RotateHandle1.Angle - StartAngle)

14. Assign following values to **Text** property of each text primitive respectively.

    – "SteadyAngle \n" + FormatReal(RotateHandle1.SteadyAngle, 1)

    – "Angle \n" + FormatReal(RotateHandle1.Angle, 1)

    – "ValueToWrite \n" + FormatReal(RotateHandle1.ValueToWrite, 1)

    – "OutOfRange \n" + String(RotateHandle1.OutOfRange)

    – "InteractionInProgress \n" + String(RotateHandle1.InteractionInProgress)

    – "WriteInProgress \n" + String(RotateHandle1.WriteInProgress)

Execute the following steps to add **RotateHandle2** graphic element to the
DirectEntryWindowTest graphic display.

1. Add the **RotateHandle2** graphic element using the Element Explorer. For
   more information on adding graphic elements to a display, refer to Element
   Explorer on page 55.

2.　Assign values to the properties of the graphic element.

–　**Action** if a value other than *SystemDefault* is required.

–　**StartAngle** = -120

–　**EndAngle** = 120

–　**MaxValue** = 100

–　**MinValue** = 0

–　Assign a general property value to **PropRef**.

DragHandle

This section helps the user to create **DragHandleH** graphic element and add it to **DirectEntryWindowTest** graphic display.
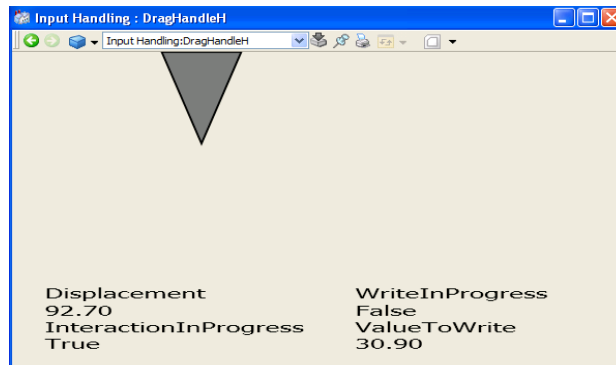
Figure 168. Drag Handle

The graphic element shown in Figure 168 contains input properties where the user sets maximum and minimum value for drag. This graphic element also contains text items displaying the values of out terminals of this input item while dragging the handle.

For more information on out terminals and properties of drag handle, refer to Input Items on page 488.

In this example, the user can drag the handle horizontally. There is a hidden bar primitive that indicates value of the drag. Color of the bar changes to Green while dragging towards the maximum value and it changes to Black while dragging towards the minimum value.

Execute the following steps to configure the **DragHandleH** graphic element.

1. Drag and drop **Polygon** (**Toolboxes > Shapes**) and **Bar** (**Toolboxes > Graphic Primitives**) primitives into the edit panel.

2. Drag and drop four **Text** primitives (**Toolboxes > Graphic Primitives)** into the edit panel.

3. Create the following input properties.

    –   **EnableDew** of data type *Boolean* and a default value of *False*.

    –   **MaxValue** of data type *Real* and a default value of *100*.

    –   **MinValue** of data type *Real* and a default value of *0*.

    –   **PropRef** of data type *PropertyRef*.

    For more information on creating input properties, refer to Input Properties on page 81.

4. Add **DragHandle** input item to the graphic element. Click on the element background to get **ElementHosted** as the only choice.

    For more information on adding input items, refer to Input Items on page 488.

5. Assign values to the properties of drag handle.

    –   **ClickTargetItem =** Name of the polygon

    –   **EnableRealDew =** *EnableDew* (input property)

    –   **MaxValue =** *MaxValue* (input property)

    –   **MinValue =** *MinValue* (input property)

    –   **Target =** *PropRef* (input property)

    –   Assign this expression to the **Value** property.

        Real (*PropRef*#Value)

6.   Assign values to the properties of **Bar**.

–   **BarValue** = Real (*PropRef*#Value)

7.   Assign this expression to **Transform** property of polygon.

Move(DragHandle1.Displacement - 20, 0)

8.   Assign following values to **Text** property of each text primitive respectively.

–   "Displacement \n" + FormatReal(DragHandle1.Displacement, 2)

–   "InteractionInProgress \n" + String(DragHandle1.InteractionInProgress)

–   "WriteInProgress \n" + String(DragHandle1.WriteInProgress)

–   "ValueToWrite \n" + FormatReal(DragHandle1.ValueToWrite, 2)

FormatReal function converts the value and keeps the number of decimals to two.

Execute the following steps to add **DragHandleH** graphic element to the **DirectEntryWindowTest** graphic display.

1.   Add the **DragHandleH** graphic element using the Element Explorer. For more information on adding graphic elements to a display, refer to

2.   Assign values to the properties of the graphic element.

–   **Action** = SystemDefault

–   **EnableDew** = False

–   **MaxValue** = 100

–   **MinValue** = 0

–   Assign a general property value to **PropRef**.

How to configure buttons

This section describes about configuration of buttons in **Buttons** graphic display (**New Graphics Demo > Buttons**). This includes Checkbox, RadioButton, UpDownButton, PushButton, VerbButton, and AspectViewButton.
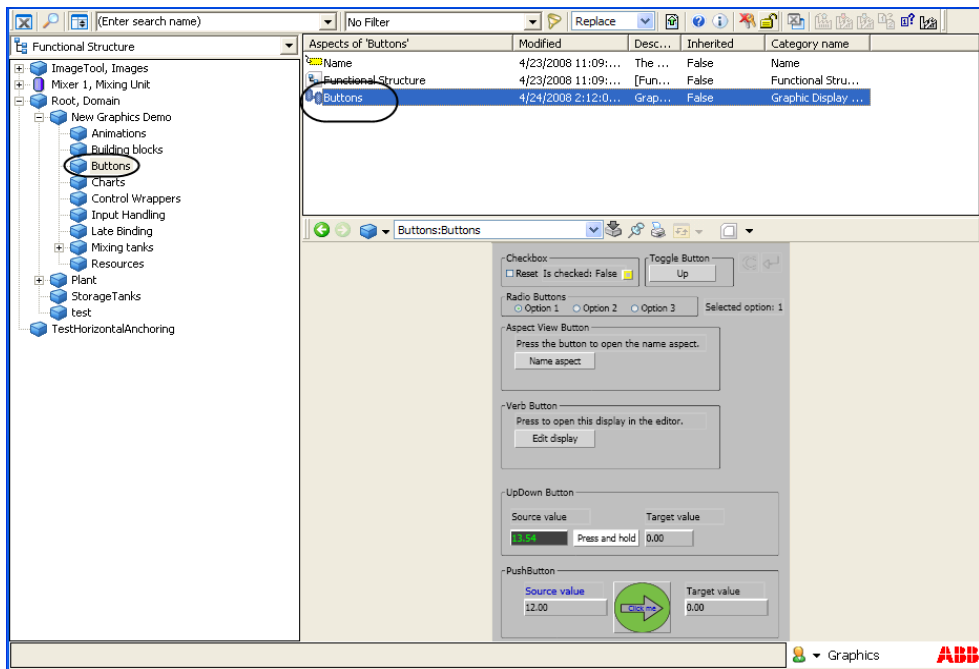
Figure 169. Buttons

Figure 170 shows the main view of the **Buttons** display.
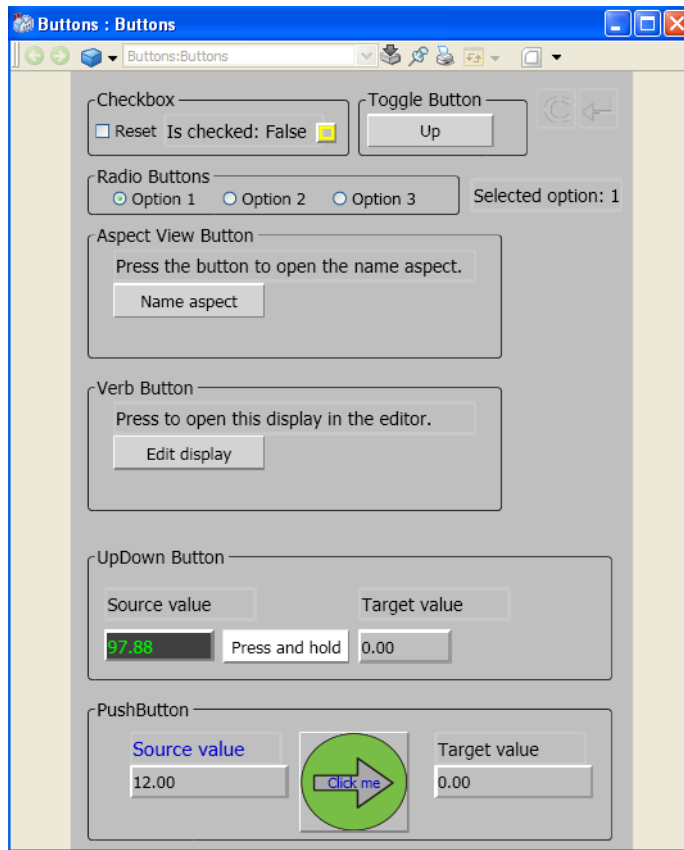
Figure 170. Main View of Buttons

Checkbox

The *Checkbox* group contains a checkbox, a text item, and an indicator. When the checkbox is selected, value of the text changes to **True** and indicator changes to **On** status. Similarly if the user turns off the indicator, the text value changes to **False** and the checkbox is cleared.

Checkbox
☐ Reset   Is checked: False   ▪
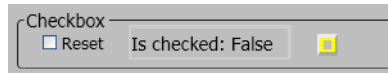
An expression variable stores the value of checkbox. It stores **True** while selecting and **False** while clearing the checkbox. This value is displayed in the text item.

Execute the following steps to configure the graphic items.

1. Select **Check Box (Toolboxes > Buttons**), **Text** (**Toolboxes > Graphic Primitives)**, and **Indicator** (**Toolboxes > Graphic Primitives)**. Drag and drop this button into the edit panel.

2. Create an expression variable *b* of data type *Boolean* and value *False*. For more information on creating expression variables, refer to Expression Variables on page 75.

3. Assign values to the properties of checkbox.

    – **IsSet** = b (expression variable)

    – **Target** = b (expression variable)

4. Assign this expression to the **Text** property of the text item.

    "Is checked:" + string (b)

5. Assign values to the properties of the indicator.

    – PresentationValue = b (expression variable)

    – **OffPropertyRef** = b (expression variable)

    – **OnPropertyRef** = b (expression variable)

Radio Button

The *RadioButtons* group contains three radio buttons and a text item. The text item displays the option number on selecting the respective option.

Radio Buttons
⦿ Option 1   ○ Option 2   ○ Option 3   Selected option: 1
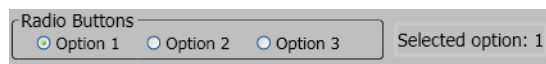
An expression variable stores the value of selected option. For example, if *Option1* is selected, the expression variable stores *1*. This value is displayed in the text item.

Execute the following steps to configure the option button.

1. Select **Radio Button (Toolboxes > Buttons**), and **Text (Toolboxes > Graphic Primitives)**. Drag and drop three radio buttons into the edit panel.

2. Create an expression variable *selectedoption* of data type *Integer*. Set the default value for *selectedoption* as *1*. This selects *Option 1* by default in the real workplace.

    For more information on creating expression variables, refer to Expression Variables on page 75.

3. Assign values to the properties of all the three option buttons.

    –   **Target** = selectedoption (expression variable)

    –   **Value** = 1, 2 and 3 for option 1, option 2, and option 3 respectively.

4. Assign this expression to the **Text** property of the text item.

    "Selected Option" + string (selectedoption)

Aspect View

The *Aspect View Button* group contains an aspect view button. Click **Name Aspect** to open the configuration view of **Name** aspect in **New Graphics Demo > Buttons**.
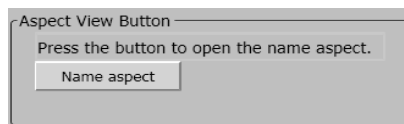
Execute the following steps to configure the aspect link button.

1. Select **Aspect View Button (Toolboxes > Buttons**). Drag and drop this button into the edit panel.

2. On the **AspectView** property, open the **Expression Editor**. Click the Views icon, browse to the graphics display **Buttons** and select **Config View** of the **Name** aspect.

Verb Button

The *Verb Button* group contains a verb button. Click **Edit display** to open the **Buttons** graphic display in the Process Graphics editor.
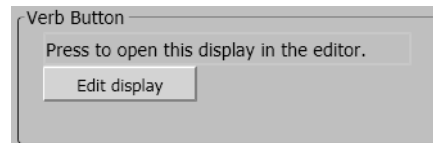
Execute the following steps to configure the verb button.

1.   Select **Verb Button (Toolboxes > Buttons**). Drag and drop this button into the edit panel.

2.   On the **VerbReference** property, open the **Expression Editor**. Click the Verbs icon, browse to the graphics display **Buttons** and select the verb **Edit**.

Up/Down Button

The *UpDown Button* group contains an up/down button, a text item and an input field. The value in the text box is written to the input field while clicking and holding the up/down button.

The text item displays value of an aspect property. This value is stored in an expression variable. On clicking the up/down button, this value is displayed in input field.

Execute the following steps to configure the up/down button.

1.   Select **Up/Down Button (Toolboxes > Buttons**). Drag and drop this button into the edit panel.

2.   Create an expression variable *target* of data type *Real* and a default value *0*. For more information on creating expression variables, refer to Expression Variables on page 75.

3.   Assign values to the properties of text item.

  –   In the **Text** property, use the Expression Editor to select the property of an aspect. For more information on using the expression editor, refer to Expression Editor on page 64.

> Use the **FormatReal** function to display values containing decimals. For more information on functions, refer to Expression Functions on page 233.

4.   Assign values to the properties of the Up/Down button.

  –   **OnDownTarget** and **WhileDownTarget** = target (expression variable)

  –   **OnDownValue** and **WhileDownValue** = the aspect property selected for the text item as specified in Step3.

  –   **Text** = "Press and Hold"

5.   Assign values to the properties of input field.

  –   **PropertyReference** = target (expression variable)

Push Button

The *Push Button* group contains a push button, and two input fields (*Source Value* and *Target Value*). The value of the input field *Source Value* is written to *Target Value* on clicking the push button.

An expression variable stores the value of *Source Value*. This value is written to the *Target Value*. The *Target Value* is written to the input field in *UpDown Button* group (refer Up/Down Button on page 385).

Execute the following steps to configure the push button.

1.   Select **Push Button** (**Toolboxes > Buttons**). Drag and drop this button into the edit panel.

2. Create an expression variable *source* of data type *Real* and a default value *12*. For more information on creating expression variables, refer to Expression Variables on page 75.

3. Assign values to the properties of input field (*Source Value*).

    –   **PropertyReference** = source (expression variable)

4. Assign values to the properties of the Push button.

    –   **Target** = target (expression variable)

    –   **Value** = source (expression variable)

    –   **Text** = "Click me"

5. Assign values to the properties of input field (*Target Value*).

    –   **PropertyReference** = target (expression variable)

How to use Late Binding

This section provides the procedure to configure the graphic display **LateBinding** (**New Graphics Demo > Late Binding**).

This graphic display explains the usage of single reference late binding, resource late binding, and array reference late binding.

The **Views** tab in the graphic display explains the single reference and array reference late binding.

The **Verbs** tab in the graphic display explains the single reference late binding.

The **Properties** tab in the graphic display explains the single reference and array reference late binding.

The **Resources** tab in the graphic display explains the resource late binding, such as text and color.

A **General Properties** aspect is used to save the parameters used in this display. Modifying the values of these properties also update the values in the graphic display during runtime.

This graphic display contains a *List View* which displays the aspect views based on the object path and aspect details specified in general properties.

It also contains a *Property View* which displays the values of aspect object properties based on the object path and aspect details specified in general properties,

The user can select a view from the view list and click **Open Selected View** to display the corresponding view of the selected object. For example, this displays a faceplate on selecting a faceplate view of an aspect from the list.

The user can also select a property from the property list and view the value of the corresponding property.

The graphic display contains expression variables that are used for assigning values to the view list during runtime. The text items display the parameter values used for the late binding functions. For more information on late binding functions, refer to Functions for Late Binding on page 258.
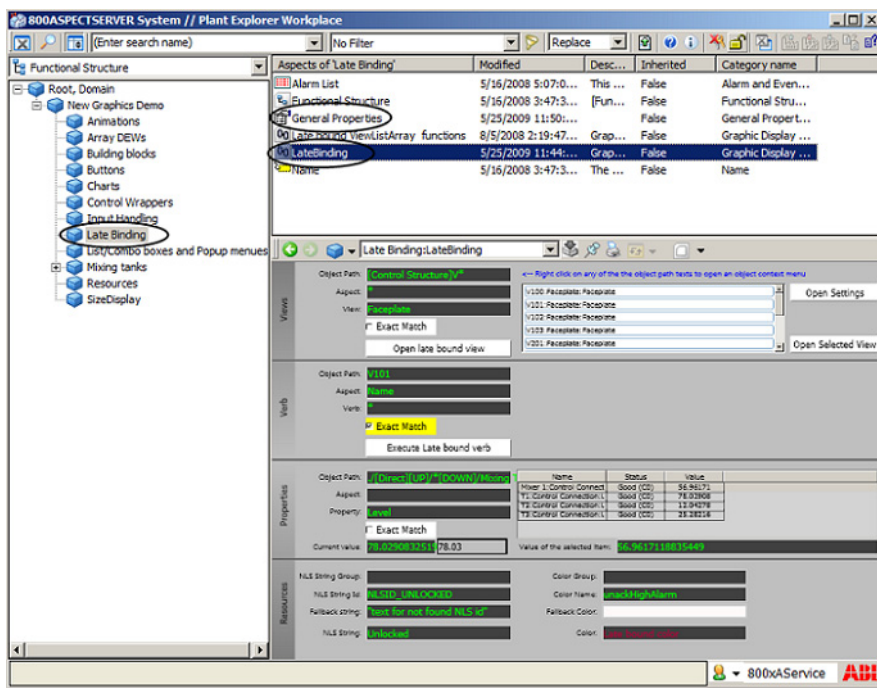
Figure 171. Late Binding

Figure 172 shows the main view of the **LateBinding** display.
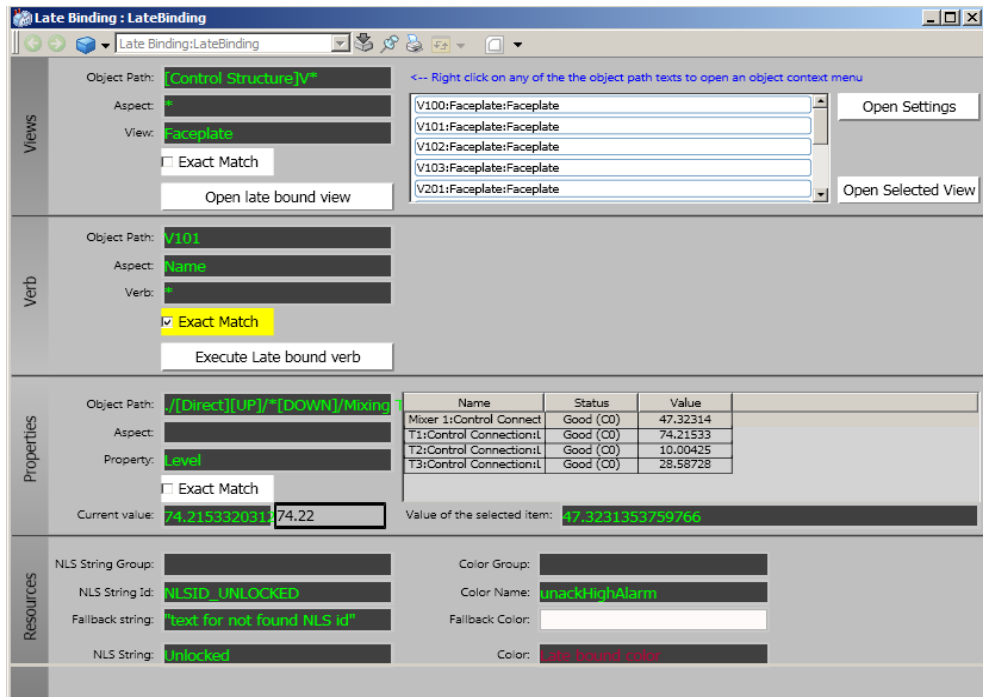
Figure 172. Main View of Late Binding

Late Binding in Aspect Views

The following example explains the configuration of array reference late binding in aspect views.
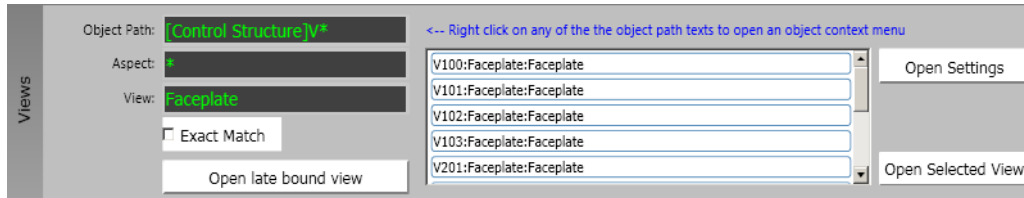
Figure 173. Views

Execute the following steps:

1. Create a **General Properties** aspect with the following properties:

    –   **ViewAspect** (**Data Type** = String and **Value** = *).

    –   **ViewName** (**Data Type** = String and **Value** = Faceplate).

    –   **ViewObjectPath** (**Data Type** = String and **Value** = [ControlStructure]V*). For more information on the syntax and examples, refer to Table 65.

    –   **viewUniqueMatch** (**Data Type** = Boolean and **Value** = False).

    These properties are referenced in the Graphics Builder. The values of the properties can be modified during runtime.

2. Create an expression variable *views* of data type *ViewRefArray*.

    Assign the expression *LateBoundViewRefArray (ViewObjectPath*, *ViewAspect*, *ViewName)*. This function is used for array reference late binding. All parameters used in this expression are general properties.

    For more information on creating expression variables, refer to Expression Variables on page 75.

3. Drag and drop **Text** primitives (**Toolboxes > Graphic Primitives)** into the edit panel. Label the primitives as **Object Path**, **Aspect**, and **View** as shown in Figure 173.

4. Drag and drop a **View List** primitive (**Toolboxes > Graphic Primitives**) into the edit panel.

5. Drag and drop a **Check Box** (**Toolboxes > Buttons**) and three **Aspect View Button** (**Toolboxes > Buttons**) into the edit panel.

6. Assign the following values to the **Text** property of the text primitives.

    –   *ViewObjectPath* (general property) for the **Object Path** text.

    –   *ViewAspect* (general property) for the **Aspect** text.

    –   *ViewName* (general property) for the **View** text.

7. Assign the following values to the properties of Check Box.

    –   **IsSet** = *viewUniqueMatch* (general property).

    –   **ResetText** = "Exact Match".

    –   **SetText** = "Exact Match".

    –   **ResetValue** = False.

    –   **SetValue** = True.

    –   **Target** = *viewUniqueMatch* (general property).

8. Assign the following values to the properties of Aspect View button 1.

    –   **AspectView** = LateBoundViewRef (*ViewObjectPath*, *ViewAspect*, *ViewName*, *viewUniqueMatch*).

        where, *ViewObjectPath*, *ViewAspect*, *ViewName*, and *viewUniqueMatch* are general properties.

    –   **Text** = "Open Late Bound View".

9. Assign the following values to the properties of Aspect View button 2.

    –   **AspectView** = $'Late Binding:General Properties:Main View'.

        This expression is used to display the *Main View* of the **General Properties** aspect.

    –   **Text** = "Open Settings".

10. Assign the following values to the properties of Aspect View button 3.

    –   **AspectView** = $'View List1.SelectedView'.

    –   **Text** = "Open SelectedView".

11. Assign the following values to the properties of View List.

   – **ControlType** = ListBox.

   – **ViewPresentationFormat** = ObjectAspectView.

   – **ViewReference** = *views* (expression variable).

Late Binding in Properties

The following example explains the configuration of single reference and array reference late binding in aspect object properties.
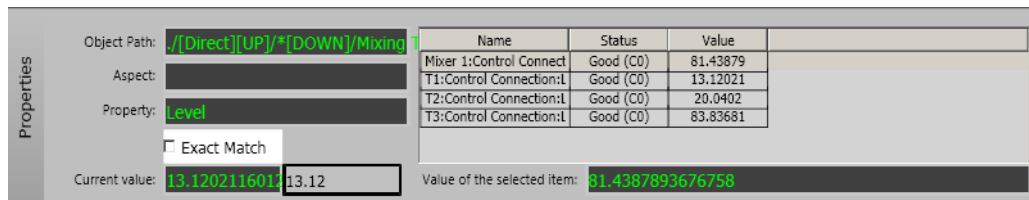
Figure 174. Properties

Execute the following steps:

1. Create a **General Properties** aspect with the following properties:

   – **PropertyAspect** (**Data Type** = String).

   – **PropertyName** (**Data Type** = String and **Value** = Level).

   – **PropertyObjectPath** (**Data Type** = String and **Value** = ./[Direct][UP]/*[DOWN]/Mixing Tanks/*). For more information on the syntax and examples, refer to Table 65.

   – **propertyUniqueMatch** (**Data Type** = Boolean and **Value** = False).

   These properties are referenced in the Graphics Builder. The values of the properties can be modified during runtime.

2. Create an expression variable *properties* of data type *PropertyRefArray*.

Assign the expression *LateBoundPropertyRefArray (PropertyObjectPath*, *PropertyAspect*, *PropertyName, 0*). This function used for array reference late binding. All parameters used in this expression are general properties.

For more information on creating expression variables, refer to Expression Variables on page 75.

3. Drag and drop **Text** primitives (**Toolboxes > Graphic Primitives)** into the edit panel. Label the primitives as **Object Path**, **Aspect**, **Property**, **Current Value**, and **Value of the selected item** as shown in Figure 174.

4. Drag and drop a **Property List** primitive (**Toolboxes > Graphic Primitives**) into the edit panel.

5. Drag and drop a **Check Box** (**Toolboxes > Buttons**) into the edit panel.

6. Assign the following values to the **Text** property of the text primitives.

   – *PropertyObjectPath* (general property) for the **Object Path** text.

   – *PropertyAspect* (general property) for the **Aspect** text.

   – *PropertyName* (general property) for the **View** text.

7. Assign the following expression to the **Text** property of the **Current Value** text.

   String (LateBoundPropertyRef(*PropertyObjectPath*, *PropertyAspect*, *PropertyName*, *propertyUniqueMatch*, *0*)#Value)

   where, *PropertyObjectPath*, *PropertyAspect*, *PropertyName* and *propertyUniqueMatch* are general properties.

8. Assign the following expression to the **Text** property of the **Value of the selected item** text.

   String (*properties*[$'PropertyList1.SelectedProperty']#Value)

9. Assign the following values to the properties of Check Box.

   – **IsSet** = *propertyUniqueMatch* (general property).

   – **ResetText** = "Exact Match".

   – **SetText** = "Exact Match".

   – **ResetValue** = False.

– **SetValue** = True.

– **Target** = *propertyUniqueMatch* (general property).

10. Assign the following values to the properties of Property List.

– **ViewPresentation** = ObjectAspectProperty.

– **PropertyColumnConfiguration** = NameStatusValue.

– **PropertyReferences** = *properties* (expression variable).

Late Binding in Resources

The following example explains the configuration of resource reference late binding.
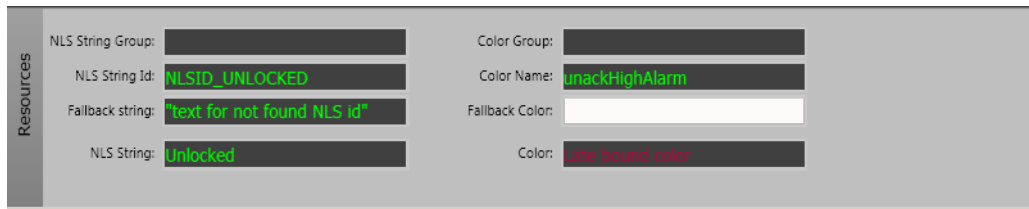
Figure 175. Resources

Execute the following steps:

1. Create a **General Properties** aspect with the following properties:

– **NLSStringFallback** (**Data Type** = String and **Value** = "text for not found NLS id").

– **NLSStringGroup** (**Data Type** = String).

– **ColorGroup** (**Data Type** = String).

– **ColorName** (**Data Type** = String and **Value** = unackHighAlarm).

– **NLSStringId** (**Data Type** = String and **Value** = NLSID_UNLOCKED).

These properties are referenced in the Graphics Builder. The values of the properties can be modified during runtime.

2. Drag and drop **Text** primitives (**Toolboxes > Graphic Primitives)** into the edit panel. Label the primitives as **NLS String Group**, **NLS String Id**, **Fallback String**, **NLS String**, **Color Group**, **Color Name**, **Fallback Color**, and **Color** as shown in Figure 175.

3. Assign the following values to the **Text** property of the text primitives.

    –   *NLSStringGroup* (general property) for the **NLS String Group** text.

    –   *NLSStringId* (general property) for the **NLS String Id** text.

    –   *NLSStringFallback* (general property) for the **Fallback String** text.

    –   *ColorGroup* (general property) for the **Color Group** text.

    –   *ColorName* (general property) for the **Color Name** text.

    –   "Late bound color" for the **Color** text.

4. Assign the following expression to the **Text** property of the **NLS String** text.

    NlsTextFromIdent (*NLSStringId*, *NLSStringGroup*, *NLSStringFallback)*

    where, *NLSStringId*, *NLSStringGroup* and *NLSStringFallback* are general properties.

5. Assign the following expression to the **TextColor** property of the **Color** text.

    LogicalColorFromName (*Color Name*, *ColorGroup*, Snow)

    where, *Color Name* and *ColorGroup* are general properties.

How to use List/Combo boxes

This section describes the procedure to configure and use the list/combo boxes.

Example 1

The List primitive shown in Figure 176 contains the items, *Alarm list*, and *Web* . The *Alarm list* item displays the main view of the Alarm and Event List aspect. The *Web* item displays the *List/Combo boxes and Popup menus* graphic aspect (**New Graphics Demo > List/Combo boxes and Popup menus**) in the Graphics Builder.
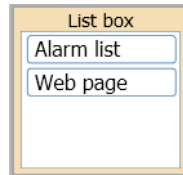
Figure 176. List

Execute the following steps to configure the **List** primitive:

1. Drag and drop the **List** primitive (**Toolboxes > Graphic Primitives**) into the edit panel.

2. Add the following element hosted input items into the edit panel:

    – Aspect View Invoker

    – Verb Invoker

3. Assign values to the following properties of **Aspect View Invoker**:

    – **Name** = AE

    – **ViewReference** = '$'vw::.:AE List:MainView

    – **Event** = OnDemand

4. Assign values to the following properties of **Verb Invoker**:

    – **Name** = Web

    – **VerbReference** = $'vr::.:List/Combo boxes and Popup menus:Edit'

    – **Event** = OnDemand

5. Assign values to the following properties of **List**:

    – **NoOfEntries** = 2

    – **Entry1** = ItemEntry ("AE", "Alarm list", True)

    – **Entry2** = ItemEntry ("Web", "Web ", True)

Example 2

Consider the following example containing a List primitive that is Generic. This example includes a **General Properties** aspect. Any value selected in the list is written to the property in **General Properties** aspect.

Execute the following steps to configure the List primitive:

1. Create a **General Properties** aspect with the following properties:

    –   **GP_Str** (**Data Type** = String).

2. Drag and drop the **List** primitive (**Toolboxes > Graphic Primitives**) into the edit panel.

3. Assign values to the following properties of **List**:

    –   **ControlType** = ComboBox

    –   **ItemSelectionEvent** = Generic

    –   **SelectedName** = GP_Str

    –   **NoOfEntries** = 2

    –   **Entry1** = ItemEntry ("ALL", "ALLVal", True)

    –   **Entry2** = ItemEntry ("IP", "IPVal ", True)

In this example, the value *ALL* is written to the general property **GP_Str** on selecting *ALLVal* in the list. Similarly, the value *IP* is written to the general property **GP_Str** on selecting *IPVal* in the list.

How to use the MultiSelectionDew input item

This section provides an example to use the MultiSelectionDew input item.

Create a **General Properties** aspect with the following properties:

–   **GP_Int1** (**Data Type** = Integer).

–   **GP_Real** (**Data Type** = Real).

–   **GP_Str** (**Data Type** = String).

–   **GP_Int2** (**Data Type** = Integer).

–   **GP_Int3** (**Data Type** = Integer).

Example 1

This is an example of a MultiSelectionDew containing the selection items aligned horizontally.

1. Drag and drop a **Text** primitive ((**Toolboxes > Graphic Primitives**) into the edit panel.

2. Add the item hosted input item **MultiSelectionDew** for the **Text** primitive.

3. Assign values to the following properties of **MultiSelectionDew**.

    –   **ContentAlignment** = Horizontal

    –   **Items** = MultiSelection (ItemContent (GP_Int1, 1234, "Item1", Enabled), ItemContent (GP_Real, 567.987, "Item2", Disabled), ItemContent (GP_Str, "Hello", "Item3", Enabled)

In runtime, click the Text item to invoke the configured MultiSelectionDew. This appears as shown in Figure 177.

Click **Item1** to write the value *1234* to the general property **GP_Int1**. Similarly, click **Item3** to write the value *Hello* to the general property **GP_Str**. **Item2** appears disabled.
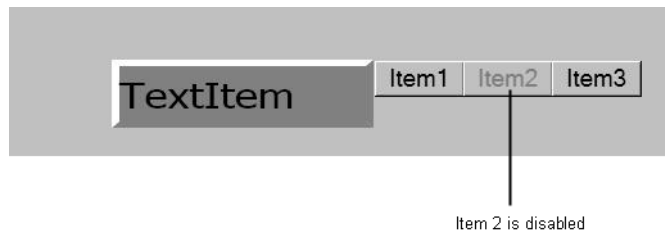
Figure 177. MultiSelectionDew containing Horizontal alignment of selection items

Example 2

This is an example of a MultiSelectionDew containing the selection items aligned vertically.

1. Drag and drop a **Text** primitive ((**Toolboxes > Graphic Primitives**) into the edit panel.

2. Add the item hosted input item **MultiSelectionDew** for the **Text** primitive.

3. Assign values to the following properties of **MultiSelectionDew**.

 – **ContentAlignment** = Vertical

 – **Items** = MultiSelection (ItemContent (GP_Str, "This is a string", "Item3", Enabled), ItemContent (GP_Int2, 2000, "Item4", Enabled), ItemContent (GP_Int3, 1000, "Item5", Hidden)

In runtime, click the Text item to invoke the configured MultiSelectionDew. This appears as shown in Figure 178.

Click **Item3** to write the value *This is a string* to the general property **GP_Str**. Similarly, click **Item4** to write the value *2000* to the general property **GP_Int2**. **Item5** is hidden.
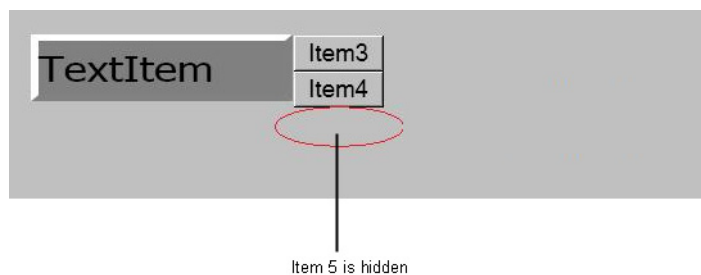
Figure 178. MultiSelectionDew containing Vertical alignment of selection items

How to create Animations

This section describes about the animation created in **New Graphics Demo > Animations**. The graphic element **Fan** is added to the **Animations using Now** graphic display.

This animation contains a rotating fan and a text item displaying different values of rotation.
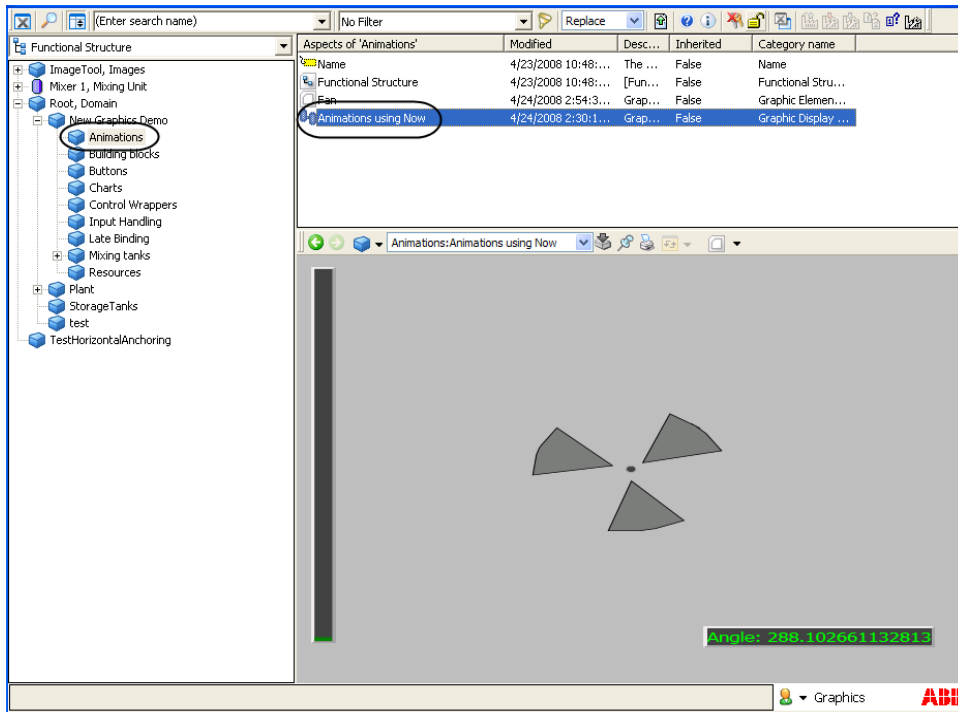
Figure 179. Animations

Figure 180 shows the main view of the **Animations using Now** display.
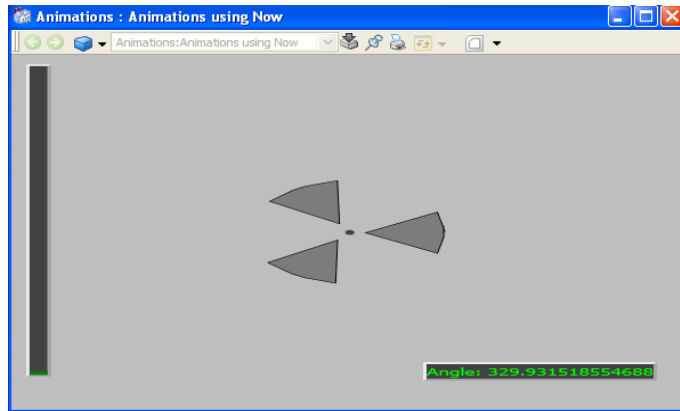
Figure 180. Main View of Animations

Execute the following steps to configure the **Fan** graphic element.

1.   Place an **Ellipse** and three **Polygons** (**Toolboxes > Shapes**) into the edit panel.
     Refer Figure 180.

2.   Assign values for the properties of the graphic element.

     –   **AnimationRate** = 10

Execute the following steps to add the **Fan** graphic element to **Animations using
Now** graphic display.

1.   Add the **Fan** graphic element using the Element Explorer. For more
     information on adding graphic elements to a display, refer to Element Explorer
     on page 55.

2.   Place a **Text** item (**Toolboxes > Graphic Primitives**) into the edit panel.

3.   Create the following expression variables.

     –   **speed** of data type *Real* and value *10*.

     –   **rotation** of data type *Real* with the following expression.

         _Now*speed%360

For more information on **_Now**, refer to Local Variables on page 268.

For more information on creating expression variables, refer to Expression Variables on page 75.

4. Assign the following expression to the **Transform** property of the graphic element.

    –   RotateAt (_Width/2, _Height/2. *rotation*)

5. Assign values to the properties of the text item.

    –   **Text** = "Angle:" + String(*rotation*)

Appendix A Standard Building Blocks

This section describes the building blocks such as graphic items and input items available for graphic aspects.

> **i** Graphics Builder displays the description for all properties of graphic items and input items as a tooltip corresponding to each property and also at the bottom of the **Properties** window.

Graphic Primitives

Graphic Primitives are the basic building blocks used for building the graphic aspects. These primitives are selected from **Toolboxes > Graphic Primitives** in the **View** menu of the Graphics Builder.

Bar

The Bar primitive has the ability to show a vertically or a horizontally placed bar graph symbol. It is used to present a value, which varies within a desired range.
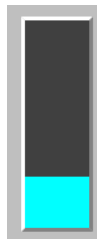
Figure 181. Bar

Limit Styles

The **NoOfLimits** property defines the total number of limits required for an instance of the bar item. For example, if the **NoOfLimits** is 3, the user can enter LimitColor, LimitStyle, LimitFill, and LimitValue for 3 limits. LimitStyle is a pictorial representation of the LimitValue. Table 85 describes the limit styles available for the Bar.

Table 85. Limit Styles
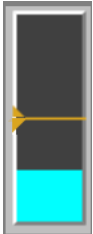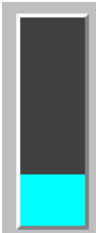
Name	Presentation
LeftSideLeft	
LeftSideRight	
LeftSideSymmetric	

Table 85. Limit Styles (Continued)

Name	Presentation
RightSideLeft	
RightSideRight	
RightSideSymmetric	
Invisible	

Conveyor
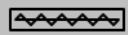
A Conveyor primitive represents a conveyor graphic. It can be animated and shown in different styles.

The **Type** property specifies the style of a conveyor and can be belt, pan, rollway, screw, spillage, or vibrating as shown in Table 86.

ℹ️ Overuse of this graphic primitive may cause high CPU load on the nodes where this primitive is used.

Table 86. Types of Conveyor

Type	Figure
Belt	
Pan	
Rollway	
Screw	
Spillage	
Vibrating	

Elevator

An Elevator primitive represents an elevator graphic. It can be animated and have different visual styles at the top and bottom.

ℹ️ Overuse of this graphic primitive may cause high CPU load on the nodes where this primitive is used.

Figure 182. Elevator

ErrorIndicator

The ErrorIndicator is used to indicate the quality of subscribed data.

If the ErrorIndicator has the **DataQuality** property configured to a dynamic expression value, runtime mode displays the actual representation. The Edit mode is a static mode representation of the primitive.

The ErrorIndicator is used in the Bar, RangeBar, and Text primitives.

Table 87. Error Presentation Images

Data Quality	Presentation
BAD, PROPERTYNOT-FOUND	
UNCERTAIN	

Table 87. Error Presentation Images (Continued)

Data Quality	Presentation
GOOD, NOTINITIALIZED, REFERENCENOTSET	
NOTPERMITTEDTO-READ	

Groupbox

A Groupbox primitive is used to group a set of graphic items.

The following are the differences between a Groupbox primitive and a VBPG frame control.

- Add a set of graphic items to the frame control. This frame control acts as a container of these graphic items.

  The Groupbox primitive is a separate graphic item and does not act as a container of graphic items grouped within this primitive.

- Movement of the graphic items within the frame control is done relative to the frame.

  Movement of graphic items grouped within a Groupbox is done with respect to the size of the edit panel.
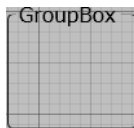
Figure 183. Group Box

Image

The Image primitive is used to add an image to the graphic aspect. Standard image formats like .bmp, .jpg, .gif, .ico, and .tif are supported.

Figure 184. Image

Indicator

The Indicator primitive is used to enable an input operation using an indicator. It takes two values, On and Off. Indicator can write different values to aspect object type properties on clicking On or Off.

The **ApplyStyle** property specifies if the indicator should work in a direct or applied mode.

Input Bar

The Input Bar primitive is used to enable an input operation using a bar.

The current value is shown in the bar and it can include one to eight limits. Each limit can be assigned different appearance and value.

While performing a drag operation in the bar, it displays the dragged value. The user can edit a value using the input field.

The Input bar always works in the applied mode.

Input RangeBar

The Input RangeBar primitive is used to enable input through a RangeBar. The current value is displayed in the RangeBar, including the two limits. The user can set a different appearance and value for each limit. This primitive has an input field that is used to edit a value. During drag operation, the dragged value is displayed in the range bar.

The Input RangeBar works only in the applied mode. The user cannot apply a value if it exceeds the limits.

Input Field

The Input Field primitive is used to enable an input operation. It accepts input of different data types such as String, Boolean, DateTime, Time, Integer and Real. Accepting different data types is controlled by the **DataType** property.

Use the **OverrideValue** property to override the value presentation through the **PropertyReference** property. If the **OverrideValue** property is not equal to "", it will be used for display instead of using the formatted **PropertyReference** value. This property overrides the current presentation.

For example, the **OverrideValue** property can be used to display an error text for an input field that is set to show values of *Real* type. Another example is to use a different presentation format other than the default such as a presentation of value 1000000 in E-format.

For more information on WriteSpecification data type, refer to WriteSpecification on page 216.

List

The List primitive provides a configurable list of input items or generic items. The items are added to the list while designing the graphic aspect. The user can specify the number of items to appear in the list and the caption to display each list item.

The input items referred to from the list must be element hosted input items and the **Event** property of the input items must be set to **OnDemand**. Setting the **Event** property to **OnDemand** avoids executing the input items for reasons other than being triggered by the List item.

The List primitive can be represented as a list box or a combo box. This is controlled by the **ControlType** property.

The items displayed in the list can be generic items or references to input items. This is selected using a property called **ItemSelectionEvent**. The value of this property can be *Generic* or *InvokeInputItem*. For more information on input items, refer to Input Items on page 127.

If the value is selected as *InvokeInputItem*, each item in the list will be an input item, that is, the item in the list can correspond to, for example, Aspect View Invoker or Verb Invoker. The specified input item is invoked on selecting the list item. While invoking an input item, name of the list item is compared with the name of element hosted input item. If this input item is found, it will be invoked. Otherwise no input item will be invoked.

If the value is selected as *Generic*, the items in the list are considered as text strings. No specific action is executed on selecting an item from the list.

For examples on configuring a List primitive, refer to How to use List/Combo boxes on page 395.

SelectedIndex is the current selection index. The index is changed when the user changes the selection. This value can also be written to the process.

InitialSelectedIndex is the index which is used to connect the current selection to a process value. If the process value changes, the selection is also updated.
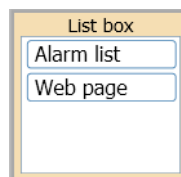
Figure 185. List

Property List

The Property List primitive provides a list of references to different aspect object type properties and displays information such as status, timestamp and value.

Property references are assigned using late bound expression function called **LateBoundPropertyRefArray**. This expression is assigned to the **PropertyReferences** property during the design of the graphic aspect.

The **StatusColors** property specifies the background color of the column displaying the status. The color depends on data quality of the property. The color for data quality representation is taken in the following order; Good, Bad, Uncertain, and Unknown.

The **StatusTexts** property specifies the user-defined texts set for different status depending on data quality of the property.

Name	Status	Value	Timestamp
General Properties:ABoolean	Good (C0)	False	06:39:29
General Properties:AFloat	Good (C0)	49.1	06:39:29
General Properties:AString	Good (C0)	14	06:39:29
General Properties:Real2	Good (C0)	83.1	06:39:29
General Properties:Real	Good (C0)	83.1	06:39:29
General Properties:String	Good (C0)	0	06:39:29

Figure 186. Property List

RangeBar

The RangeBar is used to present a value similar to a bar. It has two limit areas that indicate the value intervals in the top and bottom of a bar item.

Figure 187. Range Bar

Roll Animation

A Roll Animation primitive is a visual primitive placed in front of a tank or pipe to indicate rotation. It can be displayed using lines or dots.
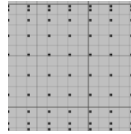
Figure 188. Roll Animation in dots
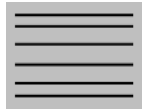
Figure 189. Roll Animation in lines

Rolling Text

A Rolling Text primitive is used to display scrolling texts in graphic displays. The text can be scrolled from left or right.

The **ScrollOption** property specifies how the scrolling should be done. This can be StartToEnd, StartToStart, or AtOnce.

Scale Horizontal

Scale Horizontal is a primitive which shows a horizontal scale. The scale can contain the start value, end value, minor and major ticks. The background of the scale is transparent. The scale is used with the other primitives like Bar.

Format for labeling the scale is specified in the **LabelFormat** property.
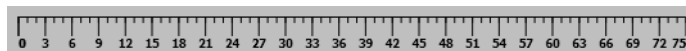
Figure 190. Scale with Scale Type as FixedTickIntervals
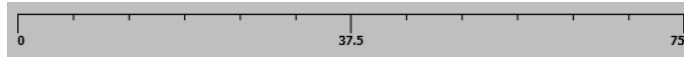
Figure 191. Scale with Scale Type as NumberOfTicks

Scale Vertical

Scale Vertical is a primitive which shows a vertical scale. The scale can contain the start value, end value, minor and major ticks. This primitive is used as a supplement to other primitives like Bar. The background of the scale is transparent.

Format for labeling the scale is specified in the **LabelFormat** property.
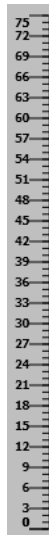
Figure 192. Scale with Scale Type as FixedTickIntervals

Figure 193. Scale with Scale Type as NumberOfTicks

Figure 194. Scale with Scale Type as NumberOfTicksOuter

Scrollable Text

A Scrollable Text primitive is used to display the text with a scrollbar included.

A horizontal scrollbar appears automatically if the text exceeds the width of the primitive. A vertical scrollbar also appears if the text exceeds the height of the primitive.

TextWrap property indicates the wrapping of texts.

Set the value of **TextWrap** to *Wrap* to word wrap the presented text and continue on a new line if the text exceeds the width of this primitive. Word wrapping is not done if the value of **TextWrap** is set to *NoWrap*.

State Indicating Symbol

The State Indicating Symbol primitive indicates a boolean value (On or Off).

Figure 195. State Indicating Symbol

Text

A Text primitive is used to display texts in the graphic displays. Bad quality or uncertain quality of data is specified by **DataQuality** property of this primitive.

TextPath property specifies the orientation of the text. This can be vertical or horizontal.

TextMode property indicates the wrapping of texts. Set the value of **TextMode** to *True* to word wrap the presented text and continue on a new line if the text exceeds the width of Text primitive.

Word wrapping is not done if the value of **TextMode** is set to *False*.

Figure 196. Text

View List

The View List primitive provides a list of view references. These references are retrieved through the late bound expression function called **LateBoundViewRefArray**. This function is assigned to the **ViewReferences** property of this primitive.

The **ViewPresentationFormat** property specifies the presentation of the view reference name. The full presentation contains object name, aspect name, and view name, but the user can select the combinations.

When **EnableNavigation** is set to *True*, the view list navigates to view reference selected by the user. This navigation uses the mode specified in **PresentationMode** property.
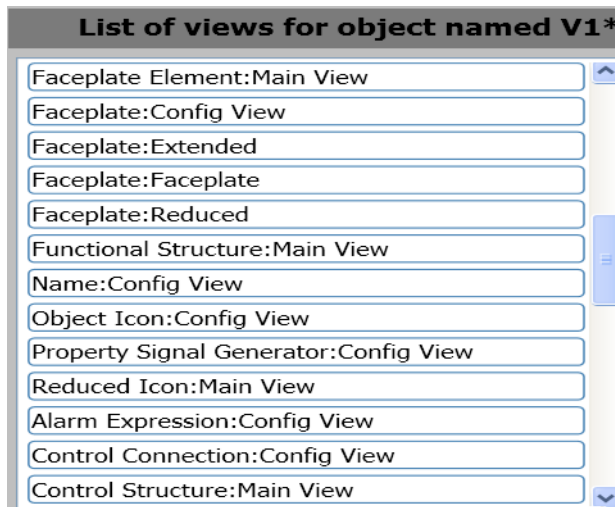
Figure 197. View List

Shapes

Shapes include the basic shapes such as Rectangle, Triangle, Tank Symbols, which are used for building the graphic aspects. The Shape controls are selected from **Toolboxes > Shapes** in the **View** menu of the Graphics Builder.

Arc

An Arc is a simple primitive element displaying a part of an ellipse. It is drawn in the anticlockwise direction.

In the Graphics Builder, the round handles appearing on the two ends of the arc can be dragged for changing the angle of the arc. The angle of the arc is also controlled by the properties **StartAngle** and **StopAngle**.

Figure 198. Arc

Chord

A Chord is a closed figure bounded by the intersection of an ellipse and a line segment.

The round handles appearing on the two ends of the chord can be dragged for changing the angle of the chord.

Figure 199. Chord

Cone

The Cone primitive draws a cone consisting of four vertices connected by straight lines.

The **ConeStrength** property specifies the size of cone top expressed in percentage of width of the bottom. If set to 100, the cone top will have the same size as the cone bottom, and the cone will be drawn as a pipe with flat ends.

To change the cone strength, select the cone and drag the yellow colored circle.
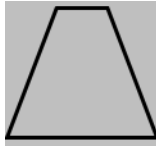
Figure 200. Cone

Curve

A curve contains a list of points, which are connected in a free hand drawing style format.

The **Tension** property controls the smoothness of the curve, based on a value between 0 and 1. When it is 0, the curve is composed of straight lines connecting the points.
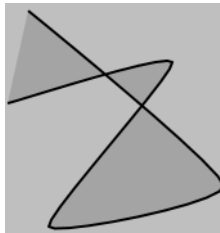
Figure 201. Curve

Ellipse

An Ellipse primitive is used for drawing an ellipse.
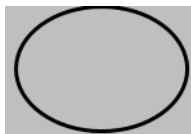
Figure 202. Ellipse

FilledPath

The FilledPath primitive is a combination of lines, arcs, and curves. This primitive is used to present Scalar Vector Graphics (SVG) by applying coordinates to the *Path* property.

Figure 203. FilledPath

Table 88 describes the syntax for defining different paths.

Table 88. Path Syntax

Command	Format	Description
Line	L x,y	Draws a straight line from the current point to the specified point. The line command consists of an "L" or an "l" followed by x-coordinate and y-coordinate values.
Cubic Bezier	C x1,y1,x2,y2,x3,y3	Draws a cubic bezier curve from the current point to the point *x3,y3* using the two specified control points *x1,y1* and *x2,y2*. The control point *x1,y1* determines the beginning of the curve, and control point *x2,y2* determines the end of the curve.

Table 88. Path Syntax (Continued)

Command	Format	Description
Close	Z	Ends the current path segment and draws a straight line from the current point to the initial point of the current segment.
Move	M x,y	Establishes a new current point. Each path segment should begin with a Move command. Subsequent Move commands indicate the start of a new subpath. This command consists of an "M" or "m" followed by x-coordinate and y-coordinate values.
Arc	A xr,yr,rx,flag1,flag2,x,y	Draws an elliptical arc from the current point to the specified point *x,y*. The size and orientation of the ellipse are defined by *xr*, *yr*, and *rx*. *xr* defines the x-radius, *yr* defines the y-radius, and *rx* defines the x-axis rotation in degrees indicating how the ellipse is rotated relative to the current coordinate system. The center of the ellipse is calculated automatically. *flag1* and *flag2* indicate which arc to be used. If *flag1* is 1, one of the two larger arc sweeps is chosen. If *flag1* is 0, one of the smaller arc sweeps is chosen. If *flag2* is 1, the arc is drawn in a positive-angle direction. If *flag2* is 0, the arc is drawn in a negative-angle direction.

For example, the path expression, "M 250,180 C 250,160 300,50 280,200 L 330, 300 L 300 300 A 50,50 0 1 1 250,180", displays the following path.
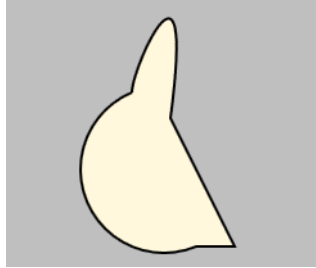
FlexiblePipe

The FlexiblePipe primitive allows the user to draw connected pipes. Draw the desired flexible pipe and double-click to complete the drawing.

For more information on editing a Flexible Pipe, refer to Editing a Polygon, Polyline, Flexible Pipe, or High Performance Pipe on page 43.

Figure 204. Flexible Pipe

Pie

The Pie primitive is a closed figure bounded by intersection of an ellipse and two line segments towards the center of the ellipse.

Figure 205. Pie

Pipe

The Pipe primitive draws a pipe with area fill. The pipe can be cut 45 degrees in one or both ends. Pipe elements can also be combined.

The **BottomRightEnd** property controls the appearance of bottom or right end of the pipe depending on the orientation.

The **TopLeftEnd** property controls the appearance of top or left end of the pipe depending on the orientation.

Figure 206. Pipe

Polygon

The Polygon primitive is used to draw a polygon. A Polygon is a closed figure bounded by a line through a number of points. Draw the desired polygon and double-click to complete the drawing.

For more information on editing a Polygon, refer to Editing a Polygon, Polyline, Flexible Pipe, or High Performance Pipe on page 43.
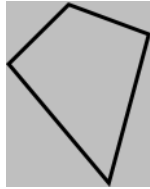
Figure 207. Polygon

Polyline

The Polyline primitive is used to draw a shape built from a sequence of connected line segments. Draw the desired polyline and double-click to complete the drawing.

For more information on editing a Polyline, refer to Editing a Polygon, Polyline, Flexible Pipe, or High Performance Pipe on page 43.

The **StartSymbol** property specifies the symbol to be shown at the start of a line. For example, Figure 209 shows a polyline with **StartSymbol** set to **Wide**.

The **EndSymbol** property specifies the symbol to be shown at the end of a line. For example, Figure 210 shows a polyline with **StartSymbol** set to **Arrow**.

The **XStart** and **YStart** properties specify the starting position of the Polyline.

The **XEnd** and **YEnd** properties specify the ending position of the Polyline.

> The properties **XStart**, **YStart**, **XEnd**, and **YEnd** will not appear if the **PointList** property is connected to an Expression Variable, Input Property, or any other property of an aspect object.
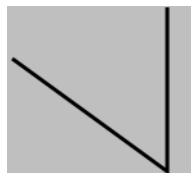
Figure 208. PolyLine
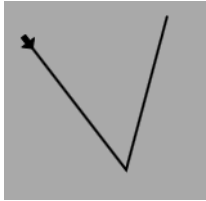
Figure 209. PolyLine with StartSymbol as Wide

Figure 210. PolyLine with EndSymbol as Arrow

Rectangle

The Rectangle primitive is used to draw a rectangle with a frame and an area fill.

Figure 211. Rectangle

The corners of the rectangle can be sharp (90 degrees) or rounded. The **Round** property specifies the percentage of rounding of the corners. The value 0 results in no rounding and 100 results into a fully rounded rectangle.

Triangle

The Triangle primitive illustrates a triangle shape.

Figure 212. Triangle

Tank Shapes

Tank Shapes contain a set of graphic primitives that are used to construct a tank object. This includes the following primitives:

- TankBody - This is used as the body of tank object.
- TankBottom - This is used as the bottom of tank object.
- TankCone - This is used as the top of tank object.
- TankNeck - This is used as the bottom of tank object.
- TankTop - This is used as the top of tank object.
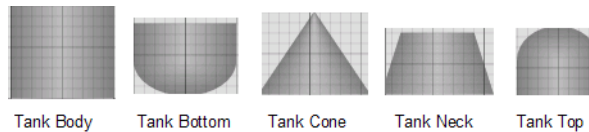
Figure 213. Tank Symbols

Charts

The following are different charts available in the Graphics Builder.

- Pie Chart, which is a closed figure created by several pie spokes that are connected using two straight lines and a connecting top arc.
- Radar Chart, which is a primitive that displays spider web like a grid with a number of configurable spokes, each representing a different property.

- Trend, which is a primitive that displays historical data in a graphical way. It can display upto 99 sets of historical data.

- XY Graph, which is a primitive that displays arrays of data in a graphical way. It can display upto 99 arrays.

- XY Plot, which is a primitive that displays a curve based on an X-value and a Y-value. A time span is used to configure how long the curve should be.

These charts are selected from **Toolboxes > Charts** in the **View** menu of the Graphics Builder.

Trend

A Trend primitive is used to display historical data in a graphical manner. Upto 99 sets of historical data can be displayed if there exists a log; otherwise trimdata is displayed by the Trend primitive.

The data to be displayed is configured by specifying an OPC property or a history log. Binary signals are considered different from analog.

The **BinaryMode** property specifies how the binary signals are drawn. This can be Stacked, Overlapping, or StackedOverlapping.

The **BinaryYMargin** property specifies the margin above and below each binary signal. This is a percentage of trend area height.

The **TraceXMaxValue** property specifies the maximum visible value for the trace. In case of a binary signal, this will be a string that represents True. For example, "On", "Open", or "Running".

The **TraceXMinValue** property specifies the minimum visible value for the trace. In case of a binary signal, this will be a string that represents False. For example, "Off", "Closed", or "Stopped".

The **TraceXCurrentValue** property specifies the data source for the trace. It can be configured with properties of data type *HistoryReference* or *OPC Property*.

The **TraceXSourceType** property determines the data source type for trace. The value can be *HistoricalLog* or *Trim*.

The **TraceXUseBufferedData** property determines whether to initialize the trend item with historical data or not for the trace. The value can be *True* or *False*. This

property is only applicable when **TraceXSourceType** is set to *Trim* and when there exists a historical log for the referenced property. After initialization with historical data, trim data will be traced.

Table 89 shows the results obtained for the combination of values of **TraceXCurrentValue** and **TraceXSourceType** properties.

Table 89. Combination of TraceXCurrentValue and TraceXSourceType

TraceXCurrentValue	TraceXSourceType	Action
OPC property with Log configuration	HistoricalLog	Historical log will be traced in the Trend.
OPC property with Log configuration	Trim	Trim data will be traced in the Trend.
OPC property	HistoricalLog	Trim data will be traced in the Trend.
OPC property	Trim	Trim data will be traced in the Trend.

⚠️ The Trend primitive does not work in Live mode of the Graphics Builder.
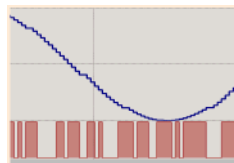
Figure 214. Trend

XY Graph

The XY Graph primitive is used to display arrays of data in a graphical manner. This primitive can display upto 99 arrays. The data to be displayed is configured by specifying an OPC property.

This primitive supports only real arrays (for example, using the *MakeRealArray* function).
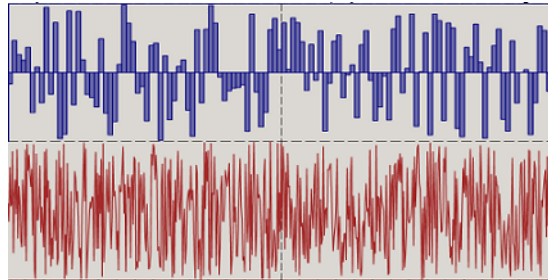
Figure 215. XY Graph

XY Plot

The XY Plot primitive is used to display a curve based on the intersection point of X-axis and Y-axis values.

This graph also has a set point symbol. The **xSetPointValue** and **ySetPointValue** properties specify the setppoint values for X-axis and Y-axis respectively. The **SetPointPen** property specifies the line color and width for the plotting the setpoint values in the chart.

The XY Plot primitive does not work in Live mode of the Graphics Builder.
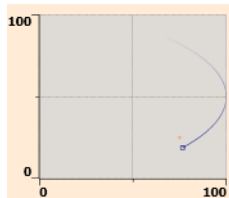
Figure 216. XY Plot

Radar Chart

A Radar Chart primitive is used to display a grid with configurable number of spokes upto 99. Each spoke represents a property, with maximum and minimum values, and high limit and low limit. Visual indicators are displayed if the property value is outside the limits.
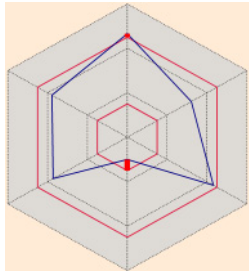
Figure 217. Radar Chart

Pie Chart

The Pie Chart primitive is used to display data in the form of a pie chart. It is possible to have a variable number of pies upto 99 in this chart. Each pie should have a value equal to or greater than zero.

Figure 218. Pie Chart

Common Properties for the Primitives

The following are some of the properties, which exist for all the primitives.

Name

The **Name** property specifies the name of the graphic item or input item. This is unique for each item.

Visible

The **Visible** property of the item can be set to *True* or *False*. The graphic item is visible if the property is set to *True*. This property can also have an expression that evaluates to *False* which hides the item.

The **Force Visible** option is available for the graphic element or graphic display. This is displayed in the context menu of the element/display in graphics item list. Selecting this option sets the visibility to *True* for all items regardless of the expression evaluation result. This setting is also forced when setting the builder to *Live* mode. This option is not possible to use in real workplace.

Line

The properties of the lines in the primitives are controlled by the Pen data type. This controls the line color and line width. For more information, refer to Pen on page 198.

Angle

The primitive elements Arc, Chord, and Pie have round handles at the start and end points. The angles can be changed by dragging the round handles using the mouse or by entering the desired values in the **StartAngle** and **StopAngle** properties.

Point List

The primitive elements Polyline, Polygon and Curve are defined by a number of connected points. The user can add or delete points in the **Point List** property to change the look of the element.

Transform

The **Transform** property is used for transforming or moving the graphic item based on the specified points. The item can also be rotated to a specified degree of angle.

Height and Width

The height and width of the graphic items can be changed by giving values for the **Height** and **Width** properties.

Rotation

The angle in which the graphic item is to be rotated can be specified in the **Rotation** property.

Position

The graphic item can be placed anywhere in the edit panel. The position of the item can be changed by specifying values for the **XPos** and **YPos** properties.

SnapsToDevicePixels

This property can be set to *True* or *False*. Pixel snap rendering is enabled for an item if this is set to *True*.

SelectFrameLine

The appearance of the line drawn while selecting the item, is controlled by this property. Set the line color and line width for the item.

SelectFrameVisible

This property controls the visibility of the frame while selecting the item. It can be set to *True* or *False*.

InnerShadeColor

This property is relevant for the tank shapes and specifies the inner color for the tank.

OuterShadeColor

This property is relevant for the tank shapes and specifies the outer color for the tank.

3D Frame Properties

All rectangle shaped primitive elements can show a 3D frame around the element. The 3D frame is controlled by FrameWidth, FrameColor, and 3D-Effect.

Table 90. 3D Frame Properties

Name	Type	Default	Description
Fram3DEffect	ENUM	RAISED	3D effect of the frame.
FrameColor	COLOR	RGB(200,200,200)	Color of the 3D frame.
FrameWidth	INTEGER	1	Width of the 3D frame.

The 3D effect of the frame can be RAISED, SUNKEN or FLAT relative to the display background.

The color for each segment of the frame is automatically calculated based on the selected frame color. The light source is placed to the top-left corner of the display. Assigning the same color for *FrameColor* and *FillColor* properties provides a good visual result.

It is important to use a frame color with the right luminosity. Luminosity of a color can be tuned by using the expression functions, **Brighten** and **Darken**. For example, the predefined color Blue is brightened by 20% if the expression is given as Brighten (Blue, 20).

Buttons

This section describes buttons included in **Buttons** in the toolbox. Buttons can be selected from **Toolboxes** in the **View** menu of the Graphics Builder.

Buttons in the graphic aspect help the operators to interact with the system.

Most buttons are defined to work in direct action or applied action. A button in an applied mode does not perform any function directly when it is pushed, but performs only if there is an apply operation.

The buttons used in the process graphics are:

• Push Button

- Up/Down Button

- Radio Button

- Check Box

- Toggle Button

- Apply Button

- Cancel Button

- Aspect View Button

- Verb Button

- Verb Button

ℹ️ The user profile **AppliedButtonAction** controls whether the buttons should be executed with an applied action or a direct action. This is applicable only if **Action** property of the button is set to **SystemDefault**.

The user profile can be set through **Graphics Profile Values PG2** aspect in **User Structure**.

Push Button

The Push Button performs a simple write operation to the target property. The write operation is performed when the left mouse button is pressed, and then released over the Push Button.

This button appears Sunken when the button is pressed or when a requested write operation is pending or in progress.

For more information on WriteSpecification data type, refer to WriteSpecification on page 216.

Up/Down Button

The UpDown Button can perform write operations to the target properties upon mouse down and/or mouse up, and while mouse down. It can also perform a repeated write operation to a property that starts after a configured delay and repeats at a configured rate.

UpDown Button is shown sunken when the button is pressed or when a requested write operation is in progress.

When a property is set in **WhileDownTarget**, writing of the value begins when the left mouse button is pressed and ends when the left mouse button is released, or when the mouse is no longer over the UpDown Button. This is a repeated write operation.

If **WhileDownInitialDelay** is 0, that is, there is no initial delay, the first write will be performed after the number of milliseconds specified in **WhileDownInterval** has elapsed.

Table 91. UpDownButton Properties

Name	Type	Default	Description
OnUpTarget	PropertyRef	null	Reference to the property of an aspect object. The write operation will be performed to this object when the left mouse button is released.
OnUpValue	VARIANT	Empty	The value to be written to OnUpTarget.
OnDownTarget	PropertyRef	null	Reference to the property of an aspect object. The write operation will be performed to this object when the left mouse button is pushed down.
OnDownValue	VARIANT	Empty	The value to be written to OnDownTarget.
WhileDownTarget	PropertyRef	null	Reference to the property of an aspect object. The write operation will be repeatedly performed to this object as long as the left mouse button is pushed down.
WhileDownValue	VARIANT	Empty	The value to be written to WhileDownTarget.
WhileDownInitialDelay	INTEGER	0	The initial delay in milliseconds before the first write to WhileDownTarget is performed.
WhileDownInterval	INTEGER	1000	The interval in milliseconds between consecutive write operations to WhileDownTarget.

Radio Button

Several radio buttons can be grouped together into a radio button group. The group is used to control multiple values, by presenting exactly one button as selected at a time.

The **GroupIndex** property specifies the radio button group to which the current radio button belongs to.

> Set the **GroupIndex** as -1 to signify that the current Radio Button item does not belong to the radio button group.
>
> Set the **GroupIndex** as any other number (other than -1) to signify that the current Radio Button item belongs to the radio button group together with other items having the same group index.

The **Checked** property controls the presentation of the button when a value has been given by the operator. The button is presented as checked when set to *True*.

Check Box

The Check Box is used to select between two values (the **SetValue** and **ResetValue**).

The Check Box has two targets, **Target** and **Target2** which can be connected to aspect object type properties. The Check Box can be used with one target connected or both the targets connected.

If both targets are connected to aspect object type properties, the **IsSet** property should also be connected. The state of the Check Box is then monitored by **IsSet**. If **IsSet** is not connected, the state of the Toggle Button is monitored by the **Target** property.

If the **IsSet** property is connected, this property is used to override the visual appearance. Otherwise, the property reference in **Target** is used. Accordingly, the values in **SetValue** or **ResetValue** are written to **Target** or **Target2**.

Toggle Button

The Toggle Button is used to select between two values (the **UpValue** and **DownValue**).

The Toggle Button has two targets, **Target** and **Target2** which can be connected to aspect object type properties. Toggle Button can be used with one target connected or both the targets connected.

If both targets are connected to aspect object type properties, the **IsSet** property should also be connected. The state of the Toggle Button is then monitored by **IsSet**. If **IsSet** is not connected, the state of the Toggle Button is monitored by the **Target** property.

If the **IsSet** property is connected, this property is used to override the visual appearance. Otherwise, the property reference in **Target** is used. Accordingly, the values in **UpValue** or **DownValue** are written to **Target** or **Target2**.

States of Toggle Button. The following are states of the Toggle Button.

• **Down**

The button will be in the Down state if target has True value. The value set for **DownText** property is displayed on the Toggle Button.

• **Up**

The button will be in the Up state if target has False value. The value set for **UpText** property is displayed on the Toggle Button.

Modes of Operation. The following are different modes of operation for the Toggle Button.

• **Interactive mode**

The **Interactive mode** starts when the user clicks the toggle button until the mouse is released. The presentation of the toggle button during the interactive mode depends on the mouse click, that is, if the button was in the Up state, it changes to Down on clicking the button and vice versa.

No action takes place until mouse on the Toggle Button is released. If the mouse is hovered outside the Toggle Button in the Down state, the button goes to the **Steady State Presentation mode** and a write operation is not done while releasing the mouse.

A write operation is done only when releasing the mouse over the Toggle Button. The interactive state is prolonged until a write response is received. After receiving

the response, the button goes to the **Steady State Presentation mode** without considering the completion of the write operation.

• **Steady State Presentation mode**

The Toggle Button will be in the **Steady State Presentation mode** when it is not in the InteractiveMode.

Apply Button

An apply operation is performed on clicking the Apply button. This button becomes enabled when a pending input operation that is governed by the Apply button exist. For more information, refer to Session Handling on page 133.

Cancel Button

A Cancel button cancels any pending input operation. For more information, refer to Session Handling on page 133.

Aspect View Button

The Aspect View Button is used to request for the invocation of any view of an aspect. The **PresentationMode** property specifies the target area for presenting the referenced aspect. This can be Overlap, Replace, Overlap_Preserve, or Base.

Feature Pack Functionality _____

The out terminal **IsPreviousDisplay** is *True* when the aspect view referenced from the **Aspect View Button** is the previous display. The previous display is the aspect view that is placed before the current aspect view in the history list.

The **ScreenAware** property enables the ability for the user to select the target screen for the aspect view invocation. The button is divided into rectangular sections; each section determining a target screen.

An orange border appears around a section when the mouse hovers over it.

The number of sections is the same as the number of physical monitors (screens) available for a workstation except for the following situations when the section presentation is not supported by the button.

• **PresentationMode** property is set to **Replace**.

•   **PresentationMode** property mode is set to **Base** and the configured number of screens in the **WorkplaceLayout** aspect is 1.

The maximum number of screens supported for a button is four.

_____

Verb Button

The VerbButton is used to request for the execution of an aspect verb.

> Right-click on any aspect in the workplace. The list of options that appear in the context menu are *Aspect Verbs*.
>
> *Acknowledge Visible Alarms* verb works only if the alarm list is open.

Classic Symbols

Classic Symbols are a set of predefined commonly used process symbols. Classic Symbols can be selected from **Toolboxes > Classic Symbols** in the **View** menu of the Graphics Builder. This is a legacy library.

Table 92 gives the list of classic symbols.

Table 92. Classic Symbols

Name	Figure
3Valve LLD	
3Valve LRD	

Table 92. Classic Symbols (Continued)

Name	Figure
Arrow	
Breaker	
Choke Valve	
Damper Flow	
Diamond	
Fan Flow	
Ground	
Integer Button	

Table 92. Classic Symbols (Continued)
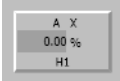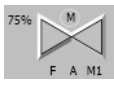
Name	Figure
Integer Limit	
Integer Percent	
Label	
Labeled Square	
Mixer	
Motcon Direction	
Motcon Status	
Motor	

Table 92. Classic Symbols (Continued)

Name	Figure
Motor Multispeed	
Pump Flow	
Reg Loop1	
Reg Loop2	
Threeway Valve	
Valve	
Valve Integer	
Valvecon Status	

Standard Symbols

Standard Symbols are graphic element aspects, which contains a display element core. Standard symbol cores are building blocks that contain properties, which can be connected to process values. These are selected from **Toolboxes** in the **View** menu of the Graphics Builder. **Standard Symbols** in the toolbox contains the following:

- Bar

- Icon

- Reduced Icon

- Status Box Horizontal

- StatusBox Vertical

- Tag

- Value

- One Direction Motor

- Two Direction Motor

- Valve

- Status Box Compact

The properties of the elements, which contain prefix "Cfg" are configuration properties.

All the color properties are logical colors. The logical colors for Display elements are defined in **Display Element Colors** aspect in the **Workplace Structure > Plant Explorer Workplace**.

Table 93 describes the different indications in the status box of the core elements.

Table 93. Indications

Condition	Description	Symbol
Alarm State	Alarm acknowledged (steady) or alarm unacknowledged (flashing).	
Manual Mode	Manual mode.	
Forced Mode	Input or output forced mode.	
Local Mode	Local mode, internal mode, panel mode, external setpoint.	
Action Mode	Priority command active, Interlock command active, or Action from voting logic.	
Disabled / Inhibited Mode	Disabled alarms or inhibited actions	

Bar

The Bar generic element displays one or two values and status information. It also displays the parameter error information and operator messages.
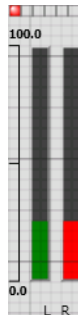
Figure 219. Bar

Table 94. Bar Properties

Name	Type	Default	Description
Action	BOOLEAN	False	Status box. Refer to Table 93 for more information.
AlarmState	INTEGER	2	Status box. Refer to Table 93 for more information.
CfgFillStyle	ENUM	Solid	Fill style for the bars. This can be solid, or transparent.
CfgFrame3Deffect	ENUM	Raised	3D effect for the frame. This can be Raised, Flat, or Sunken.
CfgFrameWidth	INTEGER	1	Frame width of the bars.
CfgScaleVisible	BOOLEAN	True	Visibility of the scale in the bars.
CfgShowOnlyRightBar	BOOLEAN	False	If *True,* it displays only the bar that appears to the right.
CfgStatusBoxVisible	BOOLEAN	True	Visibility of the status box indication.
Disabled	BOOLEAN	False	Status box. Refer to Table 93 for more information.
EnableRightBar	BOOLEAN	True	If *True,* the bar appearing on the right is enabled.
Forced	BOOLEAN	False	Status box. Refer to Table 93 for more information.
Fraction	INTEGER	1	Number of decimals to be displayed for the values in the scale.
LeftBarColor	COLOR	Green	Color of the left bar.

Table 94. Bar Properties (Continued)

Name	Type	Default	Description
LeftBarDataQuality	DataQuality	Good	Data quality of the left bar. This can be Good, Uncertain, Bad, NotPermittedToRead, NotInitialized, ReferenceNotSet, or PropertyNotFound. Refer to ErrorIndicator on page 407 for more information.
LeftBarMax	REAL	100	Maximum value of the left bar.
LeftBarMin	REAL	0	Minimum value of the left bar.
LeftBarText	STRING	"L"	Text to appear below the left bar.
LeftBarValue	REAL	25	Value for the left bar.
LeftBarSignalStatus	INTEGER	192	Signal status of the left bar. This is similar to DataQuality and accepts integer value.
Local	BOOLEAN	False	Status box. Refer to Table 93 for more information.
Manual	BOOLEAN	False	Status box. Refer to Table 93 for more information.
OpNote	BOOLEAN	False	Operator message.
ParError	BOOLEAN	False	Parameter error indication.
RightBarColor	COLOR	Green	Color of the right bar.
RightBarDataQuality	DataQuality	Good	Data quality of the right bar. This can be Good, Uncertain, Bad, NotPermittedToRead, NotInitialized, ReferenceNotSet, or PropertyNotFound. Refer to ErrorIndicator on page 407 for more information.
RightBarMax	REAL	100	Maximum value of the right bar.
RightBarMin	REAL	0	Minimum value of the right bar.

Table 94. Bar Properties (Continued)

Name	Type	Default	Description
RightBarText	STRING	"L"	Text to appear below the right bar.
RightBarValue	REAL	25	Value for the right bar.
RightBarSignalStatus	INTEGER	192	Signal status of the right bar. This is similar to DataQuality and accepts integer value.
Additional properties for the Bar are: • Name • Transform • Rotation • Position • Height and Width • Visible			

Icon

The Icon generic element displays values and status information. It also displays the parameter error information.

The following are the status information displayed in the four corners of the primitive.

- Upper Left - Displays object modes such as Man / Auto.

- Upper Right - Displays blocked alarms and operator messages.

- Lower Left - Displays backtracking, tuning, set points.

- Lower Right - Displays alarm conditions.

The inner square indicates the area where the different symbols for the object will be shown. The most appropriate way to show these symbols is to make use of a multi layered generic element, which shows one layer for each available symbol.

These generic elements should be used with an object type symbol. This primitive should have the following size properties to fit in the icon core.

- – XPos = 24

- – YPos = 16

- – Height = 72

- – Width = 72

Table 95. Icon
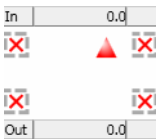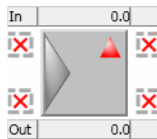
Icon Core	Symbol	Display Element Icon

Table 96. Icon Properties

Name	Type	Default	Description
CfgLowerLineFillStyle	ENUM	Solid	Fill style of the lower line in the element. This can be Solid or Transparent.
CfgLowerLineFont	FONT	Tahoma	Font for the text appearing in the lower line.
CfgLowerLineFrame3D Effect	ENUM	Raised	3D effect for the lower line frame. This can be Raised, Flat, or Sunken.
CfgLowerLineFrameWi dth	INTEGER	1	Frame width of the lower line.
CfgLowerLineTextWidt h	INTEGER	20	Text width in percentage of the total width of the lower line.

Table 96. Icon Properties (Continued)

Name	Type	Default	Description
CfgLowerLineUnitWidth	INTEGER	20	Unit width in percentage of the total width of the lower line.
CfgLowerLineVisible	BOOLEAN	True	Visibility of the lower line.
CfgUpperLineFillStyle	ENUM	Solid	Fill style of the upper line in the element. This can be Solid or Transparent.
CfgUpperLineFont	FONT	Tahoma	Font for the text appearing in the upper line.
CfgUpperLineFrame3D Effect	ENUM	Raised	3D effect for the upper line frame. This can be Raised, Flat, or Sunken.
CfgUpperLineFrameWidth	INTEGER	1	Frame width of the upper line.
CfgUpperLineTextWidth	INTEGER	20	Text width in percentage of the total width of the upper line.
CfgUpperLineUnitWidth	INTEGER	20	Unit width in percentage of the total width of the upper line.
CfgUpperLineVisible	BOOLEAN	True	Visibility of the upper line.
DataQuality	DataQuality	Good	Data quality of the element. This can be Good, Uncertain, Bad, NotPermittedToRead, NotInitialized, ReferenceNotSet, or PropertyNotFound. Refer to ErrorIndicator on page 407 for more information.
ImageLowerLeft	Image		Image to appear in the lower left corner.
ImageLowerRight	Image		Image to appear in the lower right corner.
ImageUpperLeft	Image		Image to appear in the upper left corner.
ImageUpperRight	Image		Image to appear in the upper right corner.

Table 96. Icon Properties (Continued)

Name	Type	Default	Description
LowerLineDataQuality	DataQuality	Good	Data quality of the lower line of the element. This can be Good, Uncertain, Bad, NotPermittedToRead, NotInitialized, ReferenceNotSet, or PropertyNotFound. Refer to ErrorIndicator on page 407 for more information.
LowerLineFraction	INTEGER	2	Number of decimals to appear for the value in the lower line.
LowerLineText	STRING	"Out"	Text to appear in the lower line.
LowerLineSignalStatus	INTEGER	192	Signal status of the lower line. This is similar to DataQuality and accepts integer value.
LowerLineUnit	STRING	""	Unit to appear in the lower line.
LowerLineVisible	BOOLEAN	True	Visibility of the lower line.
LowerLineValue	REAL	0	Value to appear in the lower line.
SignalStatus	INTEGER	192	Signal status of the element. This is similar to DataQuality and accepts integer value.
ToolTipLowerLeft	STRING	""	Tooltip to appear in the lower left corner.
ToolTipLowerRight	STRING	""	Tooltip to appear in the lower right corner.
ToolTipUpperLeft	STRING	""	Tooltip to appear in the upper left corner.
ToolTipUpperRight	STRING	""	Tooltip to appear in the upper right corner.
ParError	BOOLEAN	False	Parameter error indication.
UpperLineFraction	INTEGER	2	Number of decimals to appear for the value in the upper line.

Table 96. Icon Properties (Continued)

Name	Type	Default	Description
UpperLineText	STRING	"Out"	Text to appear in the upper line.
UpperLineUnit	STRING	""	Unit to appear in the upper line.
UpperLineVisible	BOOLEAN	True	Visibility of the upper line.
UpperLineValue	REAL	0	Value to appear in the upper line.
UpperLineDataQuality	DataQuality	Good	Data quality of the upper line of the element. This can be Good, Uncertain, Bad, NotPermittedToRead, NotInitialized, ReferenceNotSet, or PropertyNotFound. Refer to ErrorIndicator on page 407 for more information.
UpperLineSignalStatus	INTEGER	192	Signal status of the upper line. This is similar to DataQuality and accepts integer value.

Additional properties for the Icon are:

- Name
- Transform
- Rotation
- Position
- Height and Width
- Visible

Reduced Icon

The Reduced Icon generic element displays values and status information. It also displays the parameter error information and operator messages.

Table 97 contains the figures of Reduced Icon Core based on the value of the **CfgStatusBoxDirection** property.

Table 97. Reduced Icon

Figure	CfgStatusBoxDirection
	Up
	Down
	Right
	Left

The inner square indicates the area where the different symbols for the object type will be shown. The most appropriate way to show these symbols is to make use of a multi layered generic element which shows one layer for each available symbol. These generic elements should be used with an object type symbol. This primitive should have the following size properties to fit in the reduced icon core.

- – XPos = 10
- – YPos = 10
- – Height = 40
- – Width = 40

Table 98. Reduced Icon

Icon Core	Symbol	Display Element Icon

Table 99. Reduced Icon Core Properties

Name	Type	Default	Description
Action	BOOLEAN	False	Status box. Refer to Table 93 for more information.
AlarmState	INTEGER	2	Status box. Refer to Table 93 for more information.
CfgStatusBoxDirection	ENUM	Up	Direction of the status box. This can be Up, Down, Right, or Left.
CfgStatusBoxVisible	BOOLEAN	True	Visibility of the status box indication.
DataQuality	DataQuality	Good	Data quality of the element. This can be Good, Uncertain, Bad, NotPermittedToRead, NotInitialized, ReferenceNotSet, or PropertyNotFound. Refer to ErrorIndicator on page 407 for more information.
Disabled	BOOLEAN	False	Status box. Refer to Table 93 for more information.
Forced	BOOLEAN	False	Status box. Refer to Table 93 for more information.
Local	BOOLEAN	False	Status box. Refer to Table 93 for more information.

Table 99. Reduced Icon Core Properties (Continued)

Name	Type	Default	Description
Manual	BOOLEAN	False	Status box. Refer to Table 93 for more information.
OpNote	BOOLEAN	False	Operator message.
ParError	BOOLEAN	False	Parameter error indication.
SignalStatus	INTEGER	192	Signal status of the element. This is similar to DataQuality and accepts integer value.

Additional properties for the Reduced Icon are:
 	
    •   Name
    •   Transform
    •   Rotation
    •   Position
    •   Height and Width
    • Visible			

Status Box Horizontal

The Status Box Horizontal generic element displays status information. It displays a horizontal layout of the status box. Table 93 describes the different indications in the status box.

Figure 220. Status Box Horizontal

Table 100. Status Box Horizontal Properties

Name	Type	Default	Description
Action	BOOLEAN	False	Status box.
AlarmState	INTEGER	2	Status box.
Disabled	BOOLEAN	False	Status box.
Forced	BOOLEAN	False	Status box.
Local	BOOLEAN	False	Status box.
Manual	BOOLEAN	False	Status box.
Additional properties for the Status Box Horizontal are: • Name • Transform • Rotation • Position • Height and Width • Visible			

Status Box Vertical

The Status Box Vertical generic element displays status information. It displays a vertical layout of the status box. Table 93 describes the different indications in the status box.

Figure 221. Status Box Vertical

> **i** The Indications and Properties of Status Box Vertical primitive, are same as that
> of Status Box Horizontal primitive. Refer to Table 93 and Table 100 for the
> indications and properties respectively.

Status Box Compact

The Status Box Compact generic element displays status information. Table 93
describes the different indications in the status box.

Figure 222. Status Box Compact

> **i** The Indications and Properties of Status Box Compact primitive, are same as that
> of Status Box Horizontal primitive. Refer to Table 93 and Table 100 for the
> indications and object type properties respectively.

Tag

The Tag generic element displays name of the object and status information. It also
displays the parameter error information and operator messages.
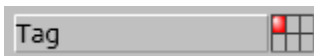
Figure 223. Tag

Table 101. Tag Properties

Name	Type	Default	Description
Action	BOOLEAN	False	Status box. Refer to Table 93 for more information.
AlarmState	INTEGER	2	Status box. Refer to Table 93 for more information.
CfgFillStyle	ENUM	Solid	Fill style for the text field. This can be solid, or transparent.
CfgFont	FONT	Tahoma	Font for the text appearing in the element.
CfgFrame3Deffect	ENUM	Raised	3D effect for the frame. This can be Raised, Flat, or Sunken.
CfgFrameWidth	INTEGER	1	Frame width of the text field.
CfgStatusBoxVisible	BOOLEAN	True	Visibility of the status box indication.
Disabled	BOOLEAN	False	Status box. Refer to Table 93 for more information.
DataQuality	DataQuality	Good	Data quality of the text field. This can be Good, Uncertain, Bad, NotPermittedToRead, NotInitialized, ReferenceNotSet, or PropertyNotFound. Refer to ErrorIndicator on page 407 for more information.
Forced	BOOLEAN	False	Status box. Refer to Table 93 for more information.
Local	BOOLEAN	False	Status box. Refer to Table 93 for more information.
Manual	BOOLEAN	False	Status box. Refer to Table 93 for more information.
OpNote	BOOLEAN	False	Operator message.
ParError	BOOLEAN	False	Parameter error indication.

Table 101. Tag Properties (Continued)

Name	Type	Default	Description
Tag	STRING	"Tag"	Name to appear in the text field.
SignalStatus	INTEGER	192	Signal status of the element. This is similar to DataQuality and accepts integer value.
Additional properties for the Tag are: • Name • Transform • Rotation • Position • Height and Width • Visible			

Value

The Value generic element displays value and status information. It also displays the parameter error information and operator messages.
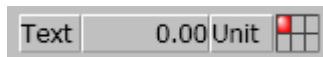
Figure 224. Value

Table 102. Value Properties

Name	Type	Default	Description
Action	BOOLEAN	False	Status box. Refer to Table 93 for more information.
AlarmState	INTEGER	2	Status box. Refer to Table 93 for more information.
CfgFillStyle	ENUM	Solid	Fill style for the text field. This can be solid, or transparent.
CfgFont	FONT	Tahoma	Font for the text appearing in the element.
CfgFrame3Deffect	ENUM	Raised	3D effect for the frame. This can be Raised, Flat, or Sunken.
CfgFrameWidth	INTEGER	1	Frame width of the text field.
CfgStatusBoxVisible	BOOLEAN	True	Visibility of the status box indication.
CfgTextWidth	INTEGER	20	Text width in percentage of the total width.
CfgUnitWidth	INTEGER	20	Unit width in percentage of the total width.
Disabled	BOOLEAN	False	Status box. Refer to Table 93 for more information.
DataQuality	DataQuality	Good	Data quality of the text field. This can be Good, Uncertain, Bad, NotPermittedToRead, NotInitialized, ReferenceNotSet, or PropertyNotFound. Refer to ErrorIndicator on page 407 for more information.
Forced	BOOLEAN	False	Status box. Refer to Table 93 for more information.
Fraction	INTEGER	1	Number of decimals to be displayed for the values in the field.
Local	BOOLEAN	False	Status box. Refer to Table 93 for more information.

Table 102. Value Properties (Continued)

Name	Type	Default	Description
Manual	BOOLEAN	False	Status box. Refer to Table 93 for more information.
OpNote	BOOLEAN	False	Operator message.
ParError	BOOLEAN	False	Parameter error indication.
Text	STRING	"Text"	Name to appear in the text field.
SignalStatus	INTEGER	192	Signal status of the element. This is similar to DataQuality and accepts integer value.
Unit	STRING	"Unit"	Unit of the element to be displayed.

Additional properties for the Value are:

- Name
- Transform
- Rotation
- Position
- Height and Width
- Visible

One Direction Motor

One Direction Motor generic element is a motor having one feedback. The *CfgType* property controls appearance of the motor. For more information, refer to *MotorUni* in *System 800xA, Control, AC 800M Binary and Analog Handling (3BSE035981*)*.

Figure 225. One Direction Motor

Table 103. One Direction Motor Properties

Name	Type	Default	Description
CfgDirection	DIRECTION	Right	Controls the direction of the symbol.
CfgFrameWidth	INTEGER	8	Configuration of the background.
CfgType	ENUM	Uni	Appearance of the motor.
Running	BOOLEAN	False	Specifies whether the motor is on or off. True indicates that the motor is running.
Additional properties for the One Direction Motor are: • Name • Transform • Rotation • Position • Height and Width • Visible			

Two Direction Motor

Two Direction Motor generic element is a motor having two feedbacks. The *CfgType* property controls appearance of the motor. For more information, refer to *MotorBi* in *System 800xA, Control, AC 800M Binary and Analog Handling (3BSE035981*).*

Figure 226. Two Direction Motor

Table 104. Two Direction Motor Properties

Name	Type	Default	Description
BiDirectional	BOOLEAN	False	Specifies if the motor is single directional or bi directional.
CfgDirection	DIRECTION	Right	Controls the direction of the symbol.
CfgFrameWidth	INTEGER	8	Configuration of the background.
CfgType	ENUM	Bi	Appearance of the motor.
Running1	BOOLEAN	False	Feedback of the motor.
Running2	BOOLEAN	False	Feedback of the motor.
Additional properties for the Two Direction Motor are: - Name - Transform - Rotation - Position - Height and Width - Visible			

Valve

Valve generic element is a valve having one feedback. The *CfgType* property controls appearance of the valve. For more information, refer to *ValveUni* in *System 800xA, Control, AC 800M Binary and Analog Handling (3BSE035981*).*

Figure 227. Valve

Table 105. Valve Properties

Name	Type	Default	Description
CfgOrientation	ORIENTATION	Horizontal	Controls the orientation of the symbol. This can be vertical or horizontal.
CfgFrameWidth	INTEGER	4	Configuration of the background.
CfgType	ENUM	Valve	Appearance of the motor.
Open	BOOLEAN	False	Specifies whether the valve is open or closed.
Additional properties for the Valve are:			

- Name
- Transform
- Rotation
- Position
- Height and Width
- Visible

Special

Select **Toolboxes > Special** from the **View** menu of the Graphics Builder to get the following primitives.

- ActiveX Wrapper
- AspectView Wrapper
- Camera View
- Capability Diagram
- VBPG Element Wrapper
- Symbol Factory Bar
- Symbol Factory Symbol

Usage of wrappers affect the performance of the system.

The following are the limitations of wrappers:

- Logical colors and Logical fonts are not supported.

- Rotation and skewing are not functional for elements that contain wrappers.

- Certain properties of datatype **Brush** and events are not supported.

- Item hosted input items are not supported. An exception is **Aspect View Wrapper** (refer to Aspect View Wrapper on page 464).

Common Properties for Wrappers

The following are some properties available for all the wrappers.

EnableInput

This property specifies whether input is enabled in the wrapper. If this property is set to *True*, all mouse operations are forwarded to the wrapped control. Otherwise, the operations are forwarded to the graphic aspect through the wrapper.

ActiveX Wrapper

The ActiveX Wrapper is used to add an ActiveX control to the graphic aspect.

To configure the ActiveX control to be used, right-click on the ActiveX Wrapper and select **Edit**. The user can only configure the ActiveX controls that are registered on the system.

Aspect View Wrapper

The Aspect View Wrapper is used to add any aspect view of a selected graphic aspect.

Select the **View Reference** property to configure the aspect view to be used.

The Aspect View Wrapper primitive does not work in Live mode of the Graphics Builder. This executes only in the real workplace.

Consider the following scenario describing the usage of **EnableInput** property of Aspect View Wrapper. An item hosted input item **AspectViewInvoker** is used along with the Aspect View Wrapper.

In this example, clicking the Aspect View Wrapper invokes the **Main View** of a trend display during runtime. The **Only Trend** view of a trend display appears on the Aspect View Wrapper during runtime.

1) Add an **Aspect View Wrapper.**

2) Set the **ViewReference** property as **Only Trend** view of a **Trend Display** aspect.

3) Set the **EnableInput** property of **Aspect View Wrapper** to **False**.

3) Add an item hosted input item **AspectViewInvoker** to this wrapper.

4) Set the **ViewReference** property of **AspectViewInvoker** as **Main View** of a **Trend Display** aspect.

5) Select **File > Save** to save the graphic display.

NOTE: Setting **EnableInput** property of the **Aspect View Wrapper** to **False** does not transfer the mouse operations to **Trend Display** aspect. The wrapper takes the control during runtime. This allows the user to click the wrapper to invoke the **Main View** of the **Trend Display**.

If the **EnableInput** property of the **Aspect View Wrapper** is set to **True**, the mouse operations are transferred to the **Trend Display** aspect. In this scenario, the **Main View** of the trend display is not invoked on clicking the wrapper.

In cases where there is a need to assign an expression containing an **if-then-else** clause to the **ViewReference** property, then how to write the expression should be considered. Consider the following expression (assigned to ViewReference):

```
if condition then
   View1
else
   View2
```

In the above expression, condition is a dynamic clause evaluating to a boolean value while View1 and View2 are constant view references.

An issue with this statement is that the condition often evaluates to a *no value* before evaluating to a value of true or false. The statement **if-then-else** considers the condition = *no value* as equal to *false*. This leads to an immediate activation of the aspect view referenced by View2 and this is not correct because the condition may next evaluate to true. This leads to an unecessary long call-up time, because View2 is first activated and later the presentation is changed to the aspect view referenced by View1. The behaviour may also lead the operator to conclude incorrectly.

Change the expression to the following to resolve this problem:

```
if !condition#HasValue then
  null
else if <condition> then
  View1
else
  View2
```

By this, View2 is not erroneously activated before condition evaluates to a value.

Camera View

Feature Pack Functionality _____

This section applies to the System 800xA 5.1 Feature Pack release only.

The Camera View item appears only if the VideONet Connect for 800xA is installed in the system. For more information on the Camera View, refer to *VideONet Connect for 800xA User Manual (2PAA109407*).*

_____

VBPG Element Wrapper

The VBPG graphic aspects can be referenced using the VBPG Element Wrapper primitive. To configure the VBPG aspect to be used, right-click on the VBPG Element Wrapper and select **Edit**.

This primitive should be used only if the user cannot migrate the VBPG aspect to the corresponding PG (Process Graphics) aspect.

Following is an example that explains how to use Standard Input Handling functionality for a wrapped VB element. It is necessary to wrap the VBPG wrapper in a graphic element and set **EnableInput** to **False** on the VBPG wrapper. This forwards every mouse event to the graphic element and call up the faceplate on left mouse click or bring up the object type context menu on right mouse click.

1. Create a graphic element *WrapperExample* in an object type.

2. Add a VBPG wrapper to this graphic element. Configure a VBPG aspect to this wrapper.

3. Add an instance of this object type in the **Control Structure**.

4. Create a graphic display in the **Functional Structure**.

    Add an instance of the graphic element *WrapperExample* into this display.

    At the real workplace, left-click the graphic element to invoke the faceplate.

Symbol Factory Controls

Symbol Factory Controls contain a set of symbols which can be used in graphic aspects. This includes **Symbol Factory Bar** and **Symbol Factory Symbol**.

Symbol Factory Bar adds a bar item.

Symbol Factory Symbol adds a graphic symbol. This can be symbols such as motors, valves, and tanks.

Right-click on the control and select **Edit** to edit the symbol or bar.

Figure 228 shows the item configuration dialog of a symbol.
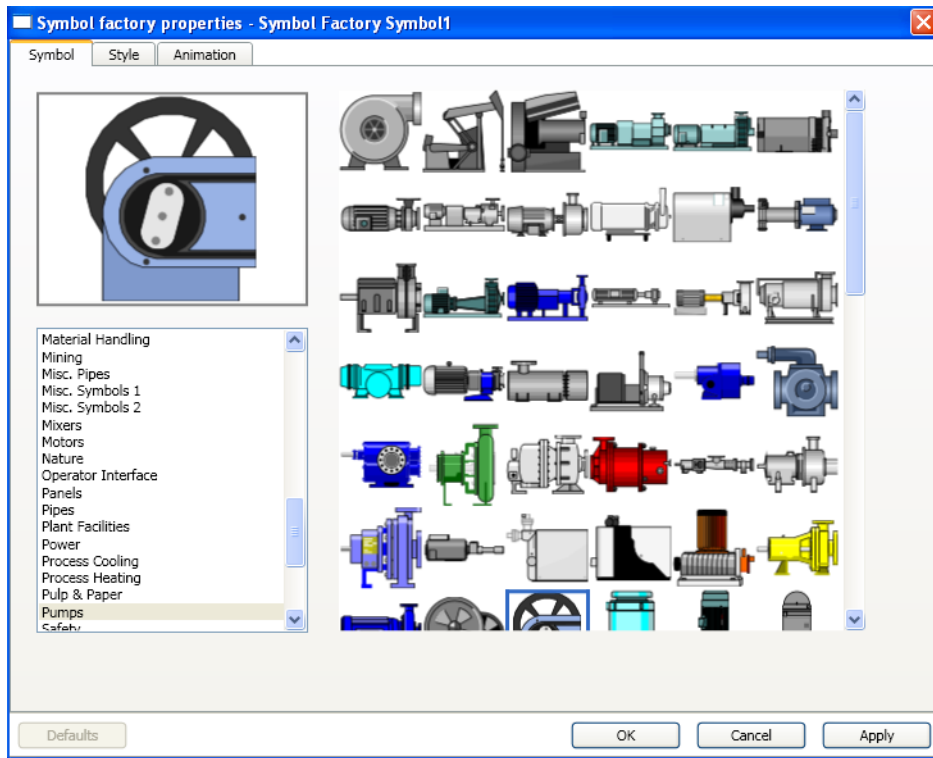
Figure 228. Symbol Factory Symbol

Figure 229 shows the item configuration dialog of a bar.
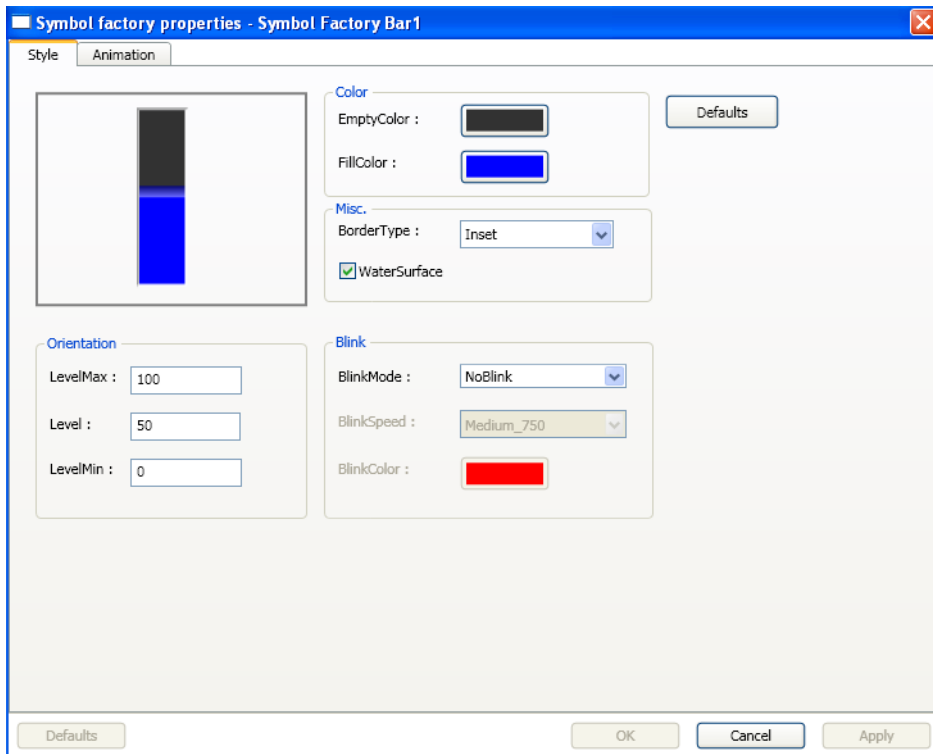
Figure 229. Symbol Factory Bar

Symbol tab allows the user to select a specific symbol to be used for the graphic aspect. Symbols are divided into different categories (for example, mixers) and each category contains a set of symbols. This tab is available only for **Symbol Factory Symbol**.

Style tab allows the user to set the appearance of the symbol.

Animation tab allows the user to give animation effects for the symbol. The animation mode, band count, band style, and breakpoints (only for analog animation mode) can be specified.

Navigation

 i   This section applies to the System 800xA 5.1 Feature Pack release only.

Select **Toolboxes > Navigation** in the **View** menu of the Graphics Builder to get the following items:

- TwoScreenNavigate

- ThreeScreenNavigate

- FourScreenNavigate

TwoScreenNavigate

The TwoScreenNavigate generic element is used to invoke an aspect view. This element allows the user to define two targets, that is, panes where the aspect view is to be displayed.

A yellow rectangular border appears when the mouse hovers to the left or right of the element (see Figure 230). Click the left or right marking area of the element to determine the target to be used.

The **ViewReference** property is used to configure the aspect view to be invoked.

The **LeftTargetID** and **RightTargetID** properties specify the identities of the requested target panes.

A typical usage is that **LeftTargetID** refers to a pane on the left monitor and **RightTargetID** refers to a pane on the right monitor,

The user may also configure **LeftTargetID** and **RightTargetID** to point to any two panes, for example two panes on one monitor.

 i   It is important that the View Class for the invoked aspect is compatible with the configuration of the target pane.

Figure 230. TwoScreenNavigate with mouse hovered to the left or right of the element

ThreeScreenNavigate

The ThreeScreenNavigate generic element is used to invoke an aspect view. This element allows the user to define three targets, that is, panes where the aspect view is to be displayed.

A yellow rectangular border appears when the mouse hovers to the left, right, or center of the element (see Figure 231). Click the left, right, or center marking area of the element to determine the target to be used.

The **ViewReference** property is used to configure the aspect view to be invoked.

The **LeftTargetID**, **RightTargetID**, and **CenterTargetID** properties specify the identities of the requested target panes.

A typical usage is that **LeftTargetID** refers to a pane on the left monitor, **RightTargetID** refers to a pane on the right monitor, and **CenterTargetID** refers to a pane on the central monitor.

The user may also configure **LeftTargetID**, **RightTargetID**, and **CenterTargetID** to point to any three panes, for example three panes on one monitor.

It is important that the View Class for the invoked aspect is compatible with the configuration of the target pane.
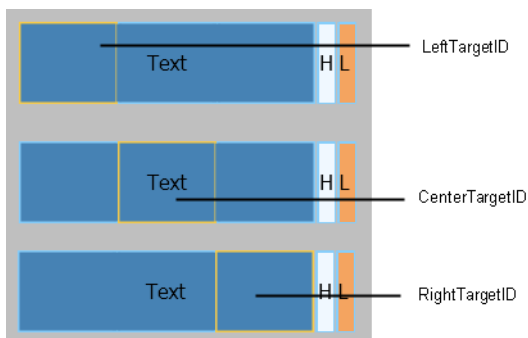
Figure 231. ThreeScreenNavigate with mouse hovered to the left, center, or right of the element

FourScreenNavigate

The FourScreenNavigate generic element is used to invoke an aspect view. This element allows the user to define four targets, that is, panes where the aspect view is to be displayed.

A yellow rectangular border appears when the mouse hovers to the bottom left, bottom right, top left, or top right of the element (see Figure 232). Click the bottom left, bottom right, top left, or top right marking area of the element to determine the target to be used.

The **ViewReference** property is used to configure the aspect view to be invoked.

The **BottomLeftTargetID**, **BottomRightTargetID**, **TopLeftTargetID**, and **TopRightTargetID** properties specify the identities of the requested target panes.

A typical usage is that **BottomLeftTargetID** refers to a pane on the bottom left monitor, **BottomRightTargetID** refers to a pane on the bottom right monitor, **TopLeftTargetID** refers to a pane on the top left monitor, and **TopRightTargetID** refers to a pane on the top right monitor.

The user may also configure **BottomLeftTargetID**, **BottomRightTargetID**, **TopLeftTargetID**, and **TopRightTargetID** to point to any four panes, for example four panes on one monitor.

ℹ️   It is important that the View Class for the invoked aspect is compatible with the configuration of the target pane.

Figure 232. FourNavigate with mouse hovered to the bottom left, bottom right, top left, or top right of the element

High Performance

ℹ️   This section applies to the System 800xA 5.1 Feature Pack release only.

Select **Toolboxes > High Performance** in the **View** menu of the Graphics Builder to get the following items:

- High Performance Pipe
- High Performance Trend
- High Performance Trend 2
- High Performance Bar
- High Performance FP Bar
- High Performance Voltmeter

- • High Performance Profile Indication

- • High Performance Profile Indication Map

- • High Performance Interlock

- • High Performance Alarm Indication

- • High Performance Radar 3 Spokes

- • High Performance Radar 4 Spokes

- • High Performance Radar 5 Spokes

- • High Performance Z Symbol

High Performance Pipe

The High Performance Pipe primitive allows the user to draw connected pipes. Draw the desired pipe and double-click to complete the drawing. For more information on editing a High Performance Pipe, refer to Editing a Polygon, Polyline, Flexible Pipe, or High Performance Pipe on page 43.

The user can also change the corner radius of the pipe using the **InnerRadius** property.
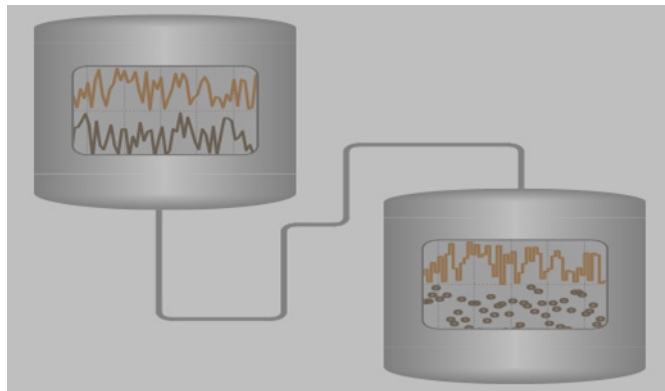
Figure 233. A graphic display containing a High Performance Pipe

High Performance Trend

A High Performance Trend primitive is used to display historical data in a graphical manner. Up to 100 sets of historical data can be displayed if there exists a log; otherwise trim data is displayed by the High Performance Trend primitive.

The data to be displayed is configured by specifying an OPC property or a history log.

The **FrameRadius** property specifies the corner radius of the rounded frame in the trend.

The **TraceXLimits** property specifies the high and low limit values in different colors using the **LimitsArray** function. For more information on the **LimitsArray** and **Limits** function, refer to Table 52 and Table 59.

For description on the other properties of the High Performance Trend primitive, refer to Trend on page 427.
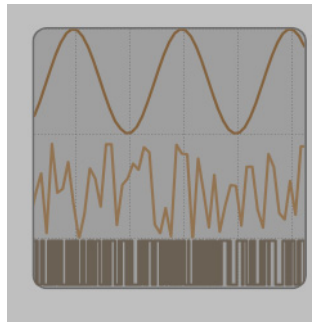
Figure 234. High Performance Trend

High Performance Alarm Indication

The High Performance Alarm Indication element provides information of the most important actual alarm. Alarms are resolved in the following order:

• Highest priority active unacknowledged alarm.

• Highest priority active acknowledged alarm.

- Highest priority inactive unacknowledged alarm.

- No alarm.

Alarm Priority is represented in different numbers and also in different shapes. The presentation in shapes is based on the value set in **AlarmPresentation** property. For a given Alarm Priority, the Alarm State is represented with the same shape but with a different fill color.

The alarm state and alarm priority can be specified in the **AlarmState** and **AlarmPriority** properties respectively.

The **MaxPriority** property specifies the maximum alarm priority to be displayed. The priorities above the **MaxPriority** value will not be displayed if the **LimitPriority** property is set to *True*.

Figure 235. High Performance Alarm Indication when AlarmPresentation is Type2 and AlarmPriority is set to 1, 2, 3, and 4 respectively

High Performance Bar

The High Performance Bar is used to visualize the PID Controller objects. This bar displays the actual value, alarm limits, and the normal value range in the format of a bar. The value is indicated by an arrow on left side and may include a numeric text. The normal value range is highlighted by blue rectangle.

A status indication is included in this bar. The user can also set the visibility of the status indicator.

The input and output values can be set through the **InputValue** and **OutputValue** properties respectively. The user can also specify the input and output value ranges, the visibility and appearance of the output value, the ranges and appearance of the input variance, and the normal value range.

The limit value areas change color automatically based on the input value set in the **InputValue** property.

The limit properties such as visibility, color for the active and default limits, can be defined for the maximum and minimum limits.

The **BackgroundColor** property is used as the fill color for the bar.

The **ShowInputRect** shows a rectangle on the bar that contains the input value and **ShowInputArrow** controls the visibility of the arrow on the bar that points to the input value.

The **ShowStatusIndication** property controls the visbility of the status indicator. The size of the status indicator can be set through the **StatusIndSize** property.

The **Forced** property specifies the Forced state in the status indication of the bar.

The **MagnEnabled** property is used to enable the magnifier function. This allows the user to zoom in the display for a defined Start range and End range.

The **SPValue** property specifies the value of the set-point (SP). The **SPDiamondVisible** property sets the visibility of the set-point value in the form of a diamond symbol. The user can also set the color and size for the diamond symbol.

The **Interlock** property shows whether the object has an interlock.

The **QualityCodeCC** and **QualityCodeOPC** properties show the status of the signal. If the signal has a *Quality* or *Status* property (in case of structured data type such as *RealIO*), set the same to **QualityCodeCC** property to represent the signal quality. If no such properties are available, set the **QualityCodeCC** property to 192 and set the *QualityDetail* subproperty of the signal to the **QualityCodeOPC** property.
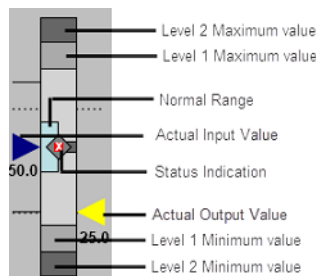
Figure 236. High Performance Bar

High Performance FP Bar

A High Performance FP Bar is used to visualize PID Controller when an input value is required from the user.

The **InputValueEnabled**, **SetpointValueEnabled**, and **OutputValueEnabled**, properties enable the user interaction with the primitive at runtime. If these properties are set to *True*, the user can specify the values for input, set-point, or output. A white border also appears around the Input Value, Setpoint Value, or OutputValue indicators respectively.

The **InputValueShowDew**, **SetpointValueShowDew**, and **OutputValueShowDew** enables a Direct Entry Window (DEW) during runtime displaying the corresponding values.

The **InputPropRef**, **OutputPropRef**, and the **SetpointPropRef** properties indicate a reference to an aspect object property to which the input, output, or set-point values that are specified in the respective DEWs can be written to.

Limit properties such as visibility, maximum and minimum range, color for the active limits and default limits, can be defined for High and Low limits.

The **BarQualityCode** shows the data quality indication of the bar.

The magnifier function is not available for the High Performance FP Bar.

For more information on the other properties, refer to
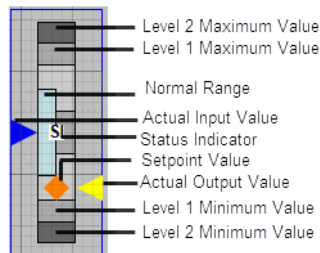
Figure 237. High Performance FP Bar

High Performance Interlock

The High Performance Interlock is used to display interlocks and overridden interlocks. This is indicated in the form of a text or a diamond symbol specified in the **Symbol** property.

The **CondInhibited** property specifies whether a condition is inhibited. The text and the text color when the Condition is Inhibited can be set using the **CondInhibitedText** and **CondInhibitedTextColor** properties. It is also possible to show the interlock in different colors if the condition inhibited is *True*.

The active interlocks and the corresponding text to be displayed can be set in **IlockX** and **ILockXText** property. These properties can be set for three interlocks. The fill color and the text color can be set for the interlock states in the **InterlockColor** and **InterlockText** property.

The fill color, the text to be displayed, and the text color can be set for no interlocks and overridden interlocks in the **NormalState** and **Override** properties respectively.

The active priority commands and the corresponding text to be displayed can be set in **PrioCmdX** and **PrioCmdXText** property. These properties can be set for three priority commands. The fill color and the text color can be set for the priority command states in the **PrioCmdColor** and **PrioCmdText** property.

The **EnableWhenActive** property enables the visibility when there is no active interlock or priority commands.

Figure 238. High Performance Interlock where A - Active Interlock, B - Overridden Active Interlock, C - Overridden Inactive Interlock, D - Inhibited Interlock

High Performance Profile Indication

The High Performance Profile Indication is used to give a quick view of several analog values, typically the temperature in different areas of a tank.

This primitive shows the limits as single lines.

The appearance settings for this primitive such as the outline color and width or color of the limit curves can be defined.

The **NumberOfSegments** property indicates the number of segments to be configured for this primitive. The minimum number of segments that can be configured is 3 and the maximum number is 5.

If the **NumberOfSegments** is 3, values can be configured for 3 segments.

Each segment can be configured for the value, minimum and maximum limit, High, High High, Low, and Low Low limit values.

The segments can also be configured for the normal minimum and maximum limits. The properties **SegmentXLimitUsedH**, **SegmentXLimitUsedHH**, **SegmentXLimitUsedL**, **SegmentXLimitUsedLL**, **SegmentXNormalRangeUsed** indicate whether the respective limit values are used or not.

The **NumericValues** property display the numeric value for each segment. The **ShowNumericValues** property controls the visibility of the numeric value for the segments.

The **ViewReference** property indicates a reference to an aspect view for the segment. Click the segment during runtime, and the configured aspect view will be displayed.
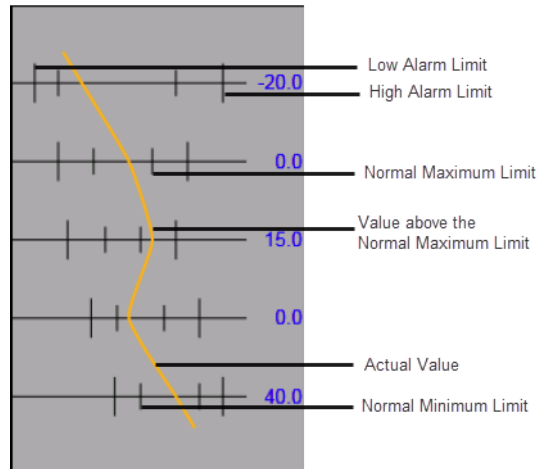
Figure 239. High Performance Profile Indication

High Performance Profile Indication Map

The High Performance Profile Indication Map is used to give a quick view of several analog values, typically the temperature in different areas of a tank.

This primitive shows the limits as curves.

For information on the properties of this primitive, refer to High Performance Profile Indication on page 479.
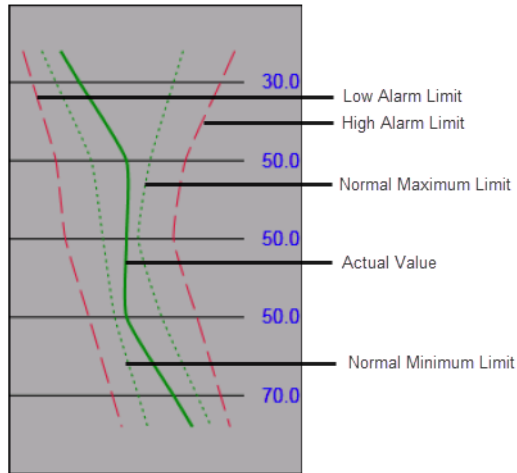
Figure 240. High Performance Profile Indication Map

High Performance Radar 3 Spokes

The High Performance Radar 3 Spokes is used to give a quick view of several analog values. This primitive contains three spokes.

Each spoke can be configured for the maximum and minimum values, high limit, and low limit, and the normal ranges. Visual indicators are displayed if the property value is outside the limit.

The **ShowValues** property controls the visibility of the values of the spokes.

The **ViewReference** property indicates a reference to an aspect view for the spoke.

For description on the other properties of the High Performance Radar 3 Spokes primitive, refer to Radar Chart on page 430.
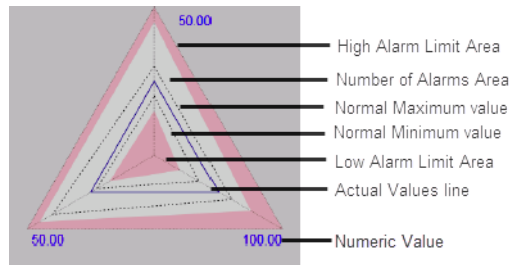
Figure 241. High Performance Radar 3 Spokes

High Performance Radar 4 Spokes

The High Performance Radar 4 Spokes is used to give a quick view of several analog values. This primitive contains four spokes.

Each spoke can be configured for the maximum and minimum values, high limit, and low limit, and the normal ranges. Visual indicators are displayed if the property value is outside the limit.

The **ShowValues** property controls the visibility of the values of the spokes.

The **ViewReference** property indicates a reference to an aspect view for the spoke.

For description on the other properties of the High Performance Radar 4 Spokes primitive, refer to Radar Chart on page 430.
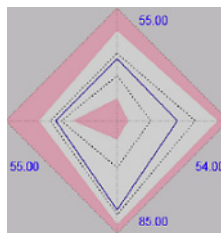
Figure 242. High Performance Radar 4 Spokes

Refer to Figure 241 for information on the different areas of the Radar Spoke.

High Performance Radar 5 Spokes

The High Performance Radar 5 Spokes is used to give a quick view of several analog values. This primitive contains five spokes.

Each spoke can be configured for the maximum and minimum values, high limit, and low limit, and the normal ranges. Visual indicators are displayed if the property value is outside the limit.

The **ShowValues** property controls the visibility of the values of the spokes.

The **ViewReference** property indicates a reference to an aspect view for the spoke.

For description on the other properties of the High Performance Radar 5 Spokes primitive, refer to Radar Chart on page 430.
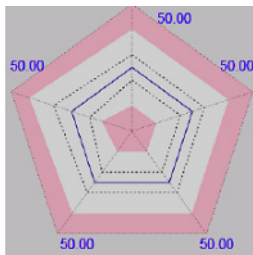
Figure 243. High Performance Radar 5 Spokes

Refer to Figure 241 for information on the different areas of the Radar Spoke.

High Performance Trend 2

A High Performance Trend 2 is used to display the actual value, alarm limits, and the normal value range in the format of a Trend. The **Description** property specifies a short description for the trend. This is displayed to the top left corner of the trend.

The normal value range is displayed in dotted lines.

The *Max1* and *Min1* limits are displayed in dashed lines.

The *Max2* and *Min2* limits are displayed in normal lines.

The colors for the limits, normal range and out of bound limits can be configured.

The **NumberOfTrends** property specifies the number of trends to be displayed. The maximum number of trends that can be configured is three.

There are two scales for the High Performance Trend 2, one to the left and one to the right. The scale to the left is applicable for Trend 1 and Trend 3, and the scale to the right is applicable for Trend 2.

The **TrendXProperty** specifies the data to be displayed in the trend. This is configured by specifying an OPC property or a history property.

The trace color in the trend changes automatically based on the colors set in the **OutOfBoundsColor**, **OutOfBoundsColorLimit1**, and **OutOfBoundsColorLimit2** properties. This is applicable only for Trend 1.

The color set in the **OutOfBoundsColor** property is used for the trace if the value is not within the normal values set in **NormalMaxValue** and **NormalMinValue** properties.

The color set in the **OutOfBoundsLimit1** property is used for the trace if the value is not within the *Max1* and *Min1* limits specified in the **LimitMax1Value** and **LimitMin1Value** properties respectively.

The color set in the **OutOfBoundsLimit2** property is used for the trace if the value is not within the *Max2* and *Min2* limits specified in the **LimitMax2Value** and **LimitMin2Value** properties respectively.

The properties **LimitMax1Used**, **LimitMax2Used**, **LimitMin1Used**, and **LimitMin2Used** indicate whether the respective limit values are used or not.
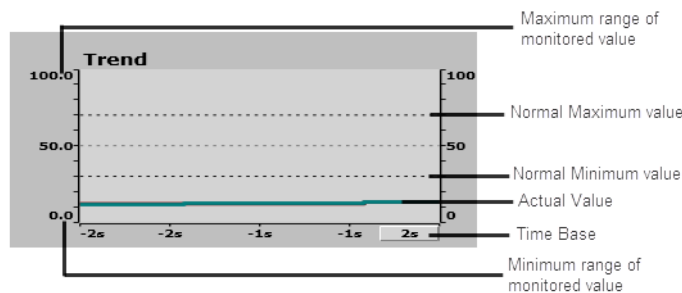
Figure 244. High Performance Trend

High Performance Voltmeter

A High Performance Voltmeter is used to display the actual value, alarm limits, and the normal value range in the format of a Voltmeter. The semi-circle in the Voltmeter is split into several arcs, where each arc represents a limit. The High, High High, High High High, Low, Low Low, and Low Low Low limits can be specified. The user can also specify the color for the respective active and inactive limits.

The properties **UsedH**, **UsedHH**, **UsedHHH**, **UsedL**, **UsedLL**, **UsedLLL** indicate whether the respective limit values are used or not.

The **MagnEnabled** property is used to enable the magnifier function. This allows the user to zoom in the display for a defined Start range and End range, and a defined Start angle and End angle.

The **Value** property specifies the value of the process value (PV). The **Max** and **Min** properties specify the range of the process value.

The state of the active and inactive limits are highlighted automatically in the High Performance Voltmeter based on the process value in the **Value** property.

The **SPValue** property specifies the value of the set-point (SP). The **SPVisible** property controls the visibility of the set-point value. The **SPDevVisible** property controls the visibility of the deviation (PV - SP).

The **StatusOPC** property specifies the OPC signal status. The **StatusForced** property specifies the Forced state in the status indication of the voltmeter. The **StatusCC** property specifies the control connection quality code for the status indication.
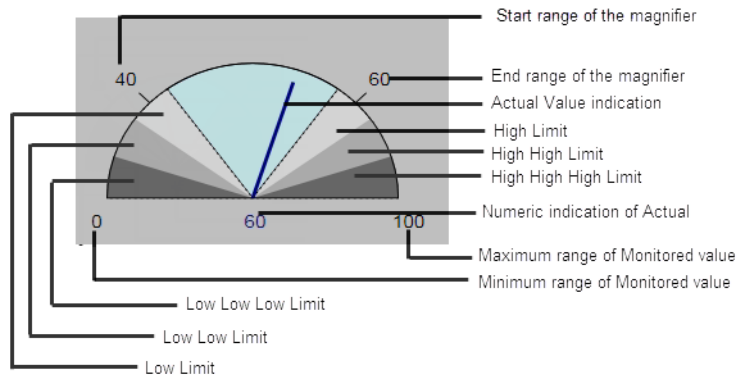
Figure 245. High Performance Voltmeter

High Performance Z Symbol

The High Performance Z Symbol is used as an error indicating element for binary signal, normally from a fail-safe control.

The alarm state and alarm priority can be specified in the **AlarmState** and **AlarmPriority** properties respectively. The frame colors can be configured for an active acknowledged alarm, active unacknowledged alarm, inactive unacknowledged alarm, and no alarm.

The **BinarySignal** property specifies the binary signal to be shown.

The **BinaryOffFillColor** property specifies the fill color for the element if the binary signal is *False*. The **BinaryOnFillColor** property specifies the fill color for the element if the binary signal is *True*.

Figure 246. High Performance Z Symbol

Input Items

Input Items are building blocks that are added for a graphic item. For more information on input items, refer to Input Items on page 127.

The following is the list of properties which are available for all the input items.

Position. The input item can be placed anywhere in the edit panel. The position of the item can be changed by specifying values for the **XPos** and **YPos** properties.

Enabled. This property allows the user to give boolean values, *True* or *False*. Selecting *True* enables in the input property. Otherwise it is disabled, that is, no actions can be performed.

Name . The name of the input item.

ModifierKeys. This property specifies the modifier keyboard key that should be pressed for an action to be invoked. It is used in combination with the **MouseButton** property. This property can be *None*, *Alt*, *Control*, *Shift*, and *Windows*. The default value is *None*.

MouseButton. This property specifies the mouse button to be clicked to invoke an action. It is used in combination with the **ModifierKeys** property. This property can be *Left*, *Right*, *Middle*, *XButton1*, and *XButton2*. The default value is *Left*.

Feature Pack Functionality _____

RetainObjectAwareness. This property ains standard input handling for an object aware graphic element that has embedded graphic items with input. For more information, refer to Retain Object Awareness on page 162.

_____

Session ApplyCancel

Session Handling is a concept used for controlling input operations on the aspect. The Session ApplyCancel input item is used in the implementation of the **Apply** and **Cancel** buttons.

An applied action is executed for the graphic aspect only after clicking the **Apply** button.

On clicking the **Cancel** button, cancel operation is performed. This terminates the execution of any action on the graphic aspect.

This input item has an out terminal **ApplyButtonEnabled**. This is used in the implementation of **Apply** button to confirm whether the apply operation is enabled or not.

Table 106 gives the list of available properties for Session ApplyCancel.

Table 106. Session ApplyCancel Properties

Name	Type	Default	Description
EnableApply	Boolean	True	This confirms whether the apply action is enabled or not.
Function	Enum	Apply	This specifies whether the action is Apply or Cancel.

Drag Handle

Drag Handle input item allows entry of a new value for an aspect object type property by dragging a handle. The following are the functions of this input item.

- Animation of different graphic items using the out terminal **Displacement**.

- Performing a write operation to an aspect object type property.

Drag Handle is often used in applied mode. It will be in interactive mode till an apply or cancel operation is performed. The Drag Handle is in interactive mode when a drag operation is in progress.

Set **ClickTargetItem** to the name of a graphic item within the element to start an interactive operation on clicking the item. Setting an empty value for this property keeps the entire hosting element click sensitive.

It is relevant to use Drag Handle as an element hosted input item.

Table 107 gives the list of available properties for Drag Handle.

Table 107. Drag Handle Properties

Name	Type	Default	Description
Action	Enum	SystemDefault	Controls when the write operation is performed. For more information, refer to Session Handling on page 133.
ClickTargetItem	String	""	The name of graphic item which must dragged to start the interactive operation.
Direction	Enum	Vertical	Controls the movement of the drag, vertically or horizontally.
EnableApply	Boolean	True	This confirms whether the apply action is enabled or not.
EnableRealDew	Boolean	False	*True* enables the RealDew input item.
DewX	Float	3	Specifies the value of X-axis to display the Dew.
DewY	Float	0	Specifies the value of Y-axis to display the Dew.
FrameWidth	Float	0	Controls the mouse movement depending on the *MaxValue* and *MinValue*.
MaxValue	Float	100	Specifies the maximum value that can be written to the aspect object type property.

Table 107. Drag Handle Properties (Continued)

Name	Type	Default	Description
MinValue	Float	0	Specifies the minimum value that can be written to the aspect object type property.
Target	PropertyRef	null	Refers to the aspect property to which the values should be written.
Value	Float	0	Specifies the value to be written to *Target*. This is used to control the **Displacement** out terminal.
StepSize	Float	0	Specifies the value by which the up/down arrow will increase/decrease. If the value is 0, the step size is automatically calculated depending on the minimum and maximum values and the drag handle screen size.
NumberOfDecimals	Integer	-1	Specifies the number of decimal values to be included in the real value. If it is negative, the number of decimals is automatically calculated depending on the minimum and maximum values and the drag handle screen size.

Table 108 gives the list of out terminals used for Drag Handle.

Table 108. Out terminals of Drag Handle

Out terminal	Type	Description
Displacement	Real	Determines the distance to move the graphic item.
InteractionInProgress	Boolean	*True* specifies that an input operation is in progress.

Table 108. Out terminals of Drag Handle (Continued)

Out terminal	Type	Description
ValueToWrite	Real	Represents the currently entered value. This value is set to InitialValue on opening the direct entry window. This value can be a null value if InitialValue is NotSet.
WriteInProgress	Boolean	*True* specifies that a write operation is being executed and not completed.
ValidValue	Boolean	This value is *True* when the value entered is a Boolean value.

Rotate Handle

Rotate Handle input item allows entry of a new value for an aspect object property by rotating an item. The following are the functions of this input item.

•   Animation of different graphic items using the out terminal **Angle**.

•   Performing a write operation to an aspect object property.

Rotate Handle is in interactive mode when a rotate operation is in progress. It is often used in applied mode. This input item will be in interactive mode till an apply or cancel operation is performed.

Set **ClickTobject typeargetItem** to the name of a graphic item within the element to start an interactive operation on clicking the item. Setting an empty value for this property keeps the entire hosting element click sensitive.

It is relevant to use Rotate Handle as an element hosted input item

Table 107 gives the list of available properties for Rotate Handle.

Table 109. Rotate Handle Properties

Name	Type	Default	Description
Action	Enum	SystemDefault	Controls when the write operation is performed. For more information, refer to Session Handling on page 133.
ClickTargetItem	String	""	The name of graphic item which must rotated to start the interactive operation.
EnableApply	Boolean	True	This confirms whether the apply action is enabled or not.
MaxValue	Float	100	Specifies the maximum value that can be written to the aspect object type property.
MinValue	Float	0	Specifies the minimum value that can be written to the aspect object type property.
Range	Float	360	Specifies the angle (in degrees) that corresponds to the difference between *MaxValue* and *MinValue*.
PivotX, PivotY	Float	20, 20	The values of X-axis and Y-axis around which the rotation is performed.
Target	PropertyRef	null	Refers to the aspect property to which the values should be written.
Value	Float	0	Specifies the value to be written to *Target*. This is used to control the **Displacement** out terminal.

Table 110 gives the list of out terminals used for Rotate Handle.

Table 110. Out terminals of Rotate Handle

Out terminal	Type	Description
Angle	Real	Refers to a rotate angle. The range of this out terminal is from 0 to *Range* value.
InteractionInProgress	Boolean	*True* specifies that an input operation is in progress.
OutOfRange	Boolean	*True* specifies that the *ValueToWrite* is out of range.
SteadyAngle	Real	Specifies the angle for the value. This is a complement to *Angle* which reflects the current dragged angle.
ValueToWrite	Real	Represents the currently entered value. This value is set to InitialValue on opening the direct entry window. This value can be a null value if InitialValue is NotSet.
WriteInProgress	Boolean	*True* specifies that a write operation is being executed and not completed.
ValidValue	Boolean	This value is *True* when the value entered is a Boolean value.

Aspect View Invoker

The Aspect View Invoker input item is used for the invocation of an aspect view (such as Config View, Main View). The aspect view invocation is done based on the mouse event selected. When the configuration is for OnDemand invocation, mouse clicks and trigger notifications are ignored.

Table 111 gives the list of available properties for Aspect View Invoker.

Table 111. Aspect View Invoker Properties

Name	Type	Default	Description
EnableApply	Boolean	True	This confirms whether the apply action is enabled or not.
Event	Enum	OnMouseUp	Invokes the aspect view depending on the event selected.
PresentationMode	Enum	Overlap	Specifies the area to invoke the aspect view.
Trigger	Boolean	False	Invokes the aspect view when the value of this property changes to *True*.
ViewReference	ViewReferenceType	null	Specifies the aspect view that is to be invoked.

Verb Invoker

The Verb Invoker input item is used for the invocation of an aspect verb (For example, opening a graphic display in edit mode). The verb invocation is done based on the mouse event selected.

Table 112 gives the list of available properties for Verb Invoker.

Table 112. Verb Invoker Properties

Name	Type	Default	Description
EnableApply	Boolean	True	This confirms whether the apply action is enabled or not.
Event	Enum	OnMouseUp	Invokes the aspect verb depending on the event selected.

Table 112. Verb Invoker Properties (Continued)

Name	Type	Default	Description
Trigger	Boolean	False	Invokes the aspect verb when the value of this property changes to *True*.
VerbReference	ViewReferenceType	null	Specifies the aspect verb that is to be invoked.

Object Context Menu Invoker

The object type Context Menu Invoker input item is used for displaying the context menu of an object in the system, typically on right-click of the hosting element. The object type context menu invocation is done based on the mouse event selected.

The object type context menu can be invoked through a right-click on hosting element or through Invoke method call.

Table 113 gives the list of available object type properties for object type ContextMenu Invoker.

Table 113. Object Type Context Menu Invoker Properties

Name	Type	Default	Description
Action	Enum	SystemDefault	Controls when the write operation is performed. For more information, refer to Session Handling on page 133.
EnableApply	Boolean	True	This confirms whether the apply action is enabled or not.

Table 113. Object Type Context Menu Invoker Properties (Continued)

Name	Type	Default	Description
Event	Enum	OnMouseUp	Invokes the context menu of the object type depending on the event selected. Select *OnMouseUp* to invoke the input item on right-click of hosting element. Select *OnDemand* to call the Invoke method.
ObjectRef	ObjectReference	null	Specifies the object type to display the context menu.

Property Writer

The Property Writer input item is used to write a value to an aspect object type property. Values can also be written to more than one aspect object type property at the same time. This input item is used as an activation part in several buttons.

ℹ️ Set the **GroupIndex** as -1 to signify that the current property writer does not belong to the group.

Set the **GroupIndex** as any other number (other than -1) to signify that the current property writer belongs to the group together with other property writers having the same group index.

Table 114 gives the list of available properties for Property Writer.

Table 114. Property Writer Properties

Name	Type	Default	Description
Action	Enum	SystemDefault	Controls when the write operation is performed. For more information, refer to Session Handling on page 133.
EnableApply	Boolean	True	This confirms whether the apply action is enabled or not.

Table 114. Property Writer Properties (Continued)

Name	Type	Default	Description
Event	Enum	OnMouseUp	Executes the action depending on the event selected.
Trigger	Boolean	False	A write operation is triggered when the value changes to *True*.
NoOfTargets	Integer	1	Specifies the number of values and targets that can be configured.
Target	PropertyRef	null	Refers to the aspect object type property to which the value should be written.
Value	Real	0	Specifies the value to be written to *Target*.
TargetX	PropertyRef	null	Refers to the aspect object type property to which the value should be written. *X* refers to the number of targets.
ValueX	Real	0	Specifies the value to be written to *Target*. *X* refers to the number of targets.

Table 114. Property Writer Properties (Continued)

Name	Type	Default	Description
WriteDelay	Integer	0	Specifies the time (in milliseconds) in which the value should be written to the target property.
WriteSequence	Enum	Async	Controls how the value should be written to the target property. **Async** writes all values to properties at the same time. **AsyncDelayed** writes the values to properties one at a time, with a specific time delay. **Sync** writes the values to properties one at a time. It waits for the acknowledgement before writing a value to the next property. **SyncDelayed** writes the values to properties one at a time. It waits for the acknowledgement before writing a value to the next property. Each write operation has a specified delay interval. **SyncStop** writes the values to properties ne at a time. It waits for the acknowledgement before writing a value to the next property. If one write operation fails, the execution of all write operations are terminated. **SyncDelayedStop** writes the values to properties ne at a time. It waits for the acknowledgement before writing a value to the next property. Each write operation has a specified delay interval. If one write operation fails, the execution of all write operations are terminated.

Table 115 gives the list of out terminals used for Property Writer.

Table 115. Out terminals of Property Writer

Out terminal	Type	Description
WriteInProgress	Boolean	*True* specifies that a write operation is being executed and not completed.
WriteOperationPending	Boolean	*True* specifies that a write operation is triggered but delayed due to waiting time for an apply action.
GroupOverrideMode	Boolean	This value is *True* for all property writer input items within a radio button group.
GroupOperationInstigator	Boolean	When PropertyWriter is executing in the group mode, the *GroupOperationInstigator* is *True* for the property writer that has currently changed a value. For example, the usage of radio buttons. Several buttons work in one group. The one that has the currently changed a value will be the instigator.

Property Writer Ex

The Property Writer Ex input item is used to write values to multiple aspect object type properties by using the **WriteSpecification** property. In Property Writer, the **NoOfTargets**, **Target**, and **Value** properties are used for writing multiple values to multiple targets. In Property Writer Ex, this can be achieved by using the **WriteSpecification** property.

For more information on WriteSpecification, refer to WriteSpecification on page 216.

The **ValueMonitor** property should be used while configuring a Checkbox style button (that is, a Toggle Button or CheckBox) and Radio button style buttons (that is Radio Button and Group Box).

A dynamic expression which is connected to **ValueMonitor**, will be monitored for the change in the value. If the value of dynamic expression is changed (toggled), the value of the out terminal **InteractionInProgress** becomes **False**.

The value is monitored while performing a write operation. The **PropertyWriterEx** then sets the value of the out terminal **InteractionInProgress** to **False** to report the completion of operation.

If **ValueMonitor** is not connected, the interactive operation is cancelled, that is, **InteractionInProgress** is set to **False** when the response to the write operation is received. Connecting the **ValueMonitor** property can prolong the interactive operation in situations where there is a delay between the completion of write operation and reporting the change of value. When the **ValueMonitor** is connected, the prolonging of interactive state is supervised. If any change in value is not reported through **ValueMonitor** within 20 seconds, the interactive operation is cancelled and a button using **PropertyWriterEx** can revert to steady state.

Table 116 gives the list of out terminals used for Property Writer Ex.

Table 116. Out terminals of Property Writer Ex

Out terminal	Type	Description
InteractionInProgress	Boolean	The value is *True* from the start of a write operation until an update of target is detected.
ValueWhileInteractionInProgress	Boolean	This state variable is defined while **InteractionInProgress** is **True**. During this time period, the value of this out terminal is the inverse of the value read from **ValueMonitor** property at the start of a write operation. The value of this out terminal is undefined when **InteractionInProgress** is **False**.

Value Writer

The Value Writer input item is similar to Property Writer input item. Following are some of the functions of the Value Writer input item.

• It supports the implementation of buttons such as Up/Down button.

- It supports write operations at different mouse events such as On mouse down, On mouse up or While mouse down.

- It cannot be used for button functions controlled by an input session.

Table 117 gives the list of available properties for Value Writer.

Table 117. Value Writer Properties

Name	Type	Default	Description
EnableApply	Boolean	True	This confirms whether the apply action is enabled or not.
OnDownTarget	PropertyRef	null	Refers to the aspect object type property to which the value should be written at the mouse down event.
OnDownValue	Variant		Specifies the value to be written to *Target* during the mouse down event.
OnUpTarget	PropertyRef	null	Refers to the aspect object type property to which the value should be written at the mouse up event.
OnUpValue	Variant		Specifies the value to be written to *Target* during the mouse up event.
WhileDownInitialDelay	Integer	0	Specifies the initial time delay (in milliseconds) before the first write operation while the mouse is down.
WhileDownInterval	Integer	500	Repeats the write operation every specified milliseconds.

Table 117. Value Writer Properties (Continued)

Name	Type	Default	Description
WhileDownTarget	PropertyRef	null	Refers to the aspect object type property to which the value should be written while the mouse is down.
WhileDownValue	Variant		Specifies the value to be written to *Target* while the mouse is down.

Table 118 gives the list of out terminals used for Value Writer.

Table 118. Out terminals of Value Writer

Out terminal	Type	Description
WriteInProgress	Boolean	*True* specifies that a write operation is being executed and not completed.

Popup Menu

The Popup Menu is an input item that executes other input items from a popup menu. It can be activated using a configured combination of mouse and keyboard button. For more information on input items, refer to Input Items on page 488.

The **NoOfEntries** property defines the number of item entries.

Table 119 gives the list of out terminals used for Popup Menu.

Table 119. Out terminals of Popup Menu

Out terminal	Type	Description
InteractionInProgress	Boolean	The value is *True* while the popup menu direct entry window is open. Otherwise, the value is *False*.

Property Transfer

The Property Transfer is an input item that is used to freeze a value upon a certain value.

The **Sink** property specifies the property or expression variable to which the value is transferred. The value to be transferred to this property is specified in **Source** property.

Default Action Invoker

The Default Action Invoker input item is used to invoke the default action on an object. This object type can be set in the **ObjectReference** property.

Default Action on an object type is in most cases the default aspect of the object, typically the faceplate. The object type reference can be retrieved by any of the following:

•   Browsing to a property for example and then apply the object type property (**#Object**) on the property reference.

•   Using the LateBoundObjectRef function.

Tooltip

Process Graphics provides the possibility to add tooltips to graphic items. A tooltip is added to a graphic item by adding the Tooltip input item. This causes a tooltip to appear while pointing at the graphic item at runtime.

The Tooltip input item can also be used as an element hosted input item thereby defining a tooltip for instances of the graphic element or generic element.

The tooltip text is specified using the **Text** property of the Tooltip input item.

Direct Entry Windows

The Direct Entry Windows are input items, which allow the user to enter values for the graphic items. This displays a dialog box for entering the values, during runtime of the graphic aspect.

To add DEWs for a graphic item, refer to

> The user profile **DirectEntryWindowFontSize** controls the font size of the value typed in direct entry windows. This can be set through **Graphics Profile Values PG2** aspect in **User Structure**.

Bool Dew

Bool Dew allows the user to enter boolean values, that is, True or False. Click **On** to set the True value, and click **Off** to set the False value.

Table 120 gives the list of available properties for Bool Dew.

Table 120. BoolDew Properties

Name	Type	Default	Description
Action	Action	SystemDefault	Controls when the write operation is performed. For more information, refer to Session Handling on page 133.
OffButtonText	String	Off	The text that will be displayed on the Off button.
OnButtonText	String	On	The text that will be displayed on the On button.
OffPropertyRef	PropertyRef		Refers to an aspect object property to which the entered value is written to, when the Off button is clicked.
OnPropertyRef	PropertyRef		Refers to an aspect object property to which the entered value is written to, when the On button is clicked.
OnValue	Variant	True	The value to be written to aspect object property specified in *OnPropertyRef*.

Table 120. BoolDew Properties (Continued)

Name	Type	Default	Description
OffValue	Variant	False	The value to be written to aspect object property specified in *OffPropertyRef.*
InitialValue	Enum	NotSet	The starting value that should be displayed when the user clicks on the hosting graphic item. This can be On, Off, or NotSet.

Table 121 gives the list of out terminals used for Bool Dew.

Table 121. Out terminals of Bool Dew

Out terminal	Type	Description
InteractionInProgress	Boolean	This value changes to *True* on opening the direct entry window.
WriteInProgress	Boolean	*True* specifies that a write operation is being executed. This value changes to *False* after the completion of write operation.
ValueToWrite	Variant	Represents the currently entered value. This value is set to InitialValue on opening the direct entry window. This value can be a null value if InitialValue is NotSet.
ValidValue	Boolean	This value is *True* when the value entered is a Boolean value.

Date Dew

Date Dew allows the user to enter a date and time value for the graphic item or element.

Table 122 gives the list of available properties for Date Dew.

Table 122. DateDew Properties

Name	Type	Default	Description
PropertyRef	PropertyRef		Refers to the aspect object property to which the entered value is written to.
InitialValue	String	Current date and time	The starting value that is displayed on clicking the graphic item or element.
DisplayOption	Enum	Date	• If *Date* is selected, enter a date value for the item. • If *DateTime* is selected, enter date and time value for the item. • If *DateTimewithMilliseconds* is selected, enter a date value and can change the time upto milliseconds.

Table 123 gives the list of out terminals used for Date Dew.

Table 123. Out terminals of Date Dew

Out terminal	Type	Description
InteractionInProgress	Boolean	This value changes to *True* on opening the direct entry window.
WriteInProgress	Boolean	*True* specifies that a write operation is being executed. This value changes to *False* after the completion of write operation.
ValueToWrite	Integer	Represents the currently entered value. This value is set to InitialValue on opening the direct entry window.
ValidValue	Boolean	This value is *True* when the value entered is a Date value.

Integer Dew

This direct entry window allows the user to enter integer values for the graphic aspect.

Table 124 gives the list of available properties for Integer Dew.

Table 124. IntegerDew Properties

Name	Type	Default	Description
PropertyRef	PropertyRef		Refers to the aspect object property to which the entered value is written to.
InitialValue	Integer	0	The value that is displayed on clicking the graphic item or element.
ExtendedWrite	WriteSpecification	Empty	To write multiple values to multiple targets. **PropertyRef** is used if the **ExtendedWrite** has a value *Empty.* It supports the functions, SingleWrite, SequentialWrite, and BatchWrite.
MaxValue	Integer	100	The maximum value for the input item.
MinValue	Integer	0	The minimum value for the input item.
StepSize	Integer	1	The value by which the input value will increase on clicking the arrows.
EnableLimits	Boolean	True	If *True*, enter a value that exist between the *MinValue* and *MaxValue* specified. If *False*, enter any integer value.

Table 125 gives the list of out terminals used for Integer Dew.

Table 125. Out terminals of Integer Dew

Out terminal	Type	Description
InteractionInProgress	Boolean	This value changes to *True* on opening the direct entry window.
WriteInProgress	Boolean	*True* specifies that a write operation is being executed. This value changes to *False* after the completion of write operation.
ValueToWrite	Integer	Represents the currently entered value. This value is set to InitialValue on opening the direct entry window.
ValidValue	Boolean	This value is *True* when the value entered is an Integer value.

Real Dew

This direct entry window allows the user to enter real values for the graphic aspect.

Table 126 gives the list of available properties for Real Dew.

Table 126. RealDew Properties

Name	Type	Default	Description
PropertyRef	PropertyRef		Refers to the aspect object property to which the entered value is written to.
InitialValue	Real	0	The value that is displayed on clicking the graphic item or element.

Table 126. RealDew Properties (Continued)

Name	Type	Default	Description
ExtendedWrite	WriteSpecification	Empty	To write multiple values to multiple targets. **PropertyRef** is used if the **ExtendedWrite** has a value *Empty*. It supports the functions, SingleWrite, SequentialWrite, and BatchWrite.
MaxValue	Real	100	The maximum value for the input item.
MinValue	Real	0	The minimum value for the input item.
StepSize	Real	1	The value by which the input value will increase on clicking the arrows.
NumberofDecimals	Integer	2	The number of decimals allowed for the entry value.
EnableLimits	Boolean	True	If *True*, enter a value that exist between the *MinValue* and *MaxValue* specified. If *False*, enter any integer value.

Table 127 gives the list of out terminals used for Real Dew.

Table 127. Out terminals of Real Dew

Out terminal	Type	Description
InteractionInProgress	Boolean	This value changes to *True* on opening the direct entry window.
WriteInProgress	Boolean	*True* specifies that a write operation is being executed. This value changes to *False* after the completion of write operation.

Table 127. Out terminals of Real Dew

Out terminal	Type	Description
ValueToWrite	Real	Represents the currently entered value. This value is set to InitialValue on opening the direct entry window.
ValidValue	Boolean	This value is *True* when the value entered is a Real value.

String Dew

String Dew takes String type values, that is, the user can enter text values for the items.

Table 128 gives the list of available properties for String Dew.

Table 128. StringDew Properties

Name	Type	Default	Description
PropertyRef	PropertyRef		Refers to the aspect object property to which the entered value is written to.
InitialValue	String	" "	The value that is displayed on clicking the graphic item or element.
MaxNoOfChars	Integer	0	The maximum number of characters allowed to enter.

Table 129 gives the list of out terminals used for String Dew.

Table 129. Out terminals of String Dew

Out terminal	Type	Description
InteractionInProgress	Boolean	This value changes to *True* on opening the direct entry window.
WriteInProgress	Boolean	*True* specifies that a write operation is being executed. This value changes to *False* after the completion of write operation.
ValueToWrite	String	Represents the currently entered value. This value is set to InitialValue on opening the direct entry window.
ValidValue	Boolean	This value is always *True*.

Time Dew

Time Dew allows the user to enter time span values for the items. The values should be entered in the format as specified in Table 132.

Table 130 gives the list of available properties for Time Dew.

Table 130. TimeDew Properties

Name	Type	Default	Description
PropertyRef	PropertyRef		Refers to the aspect object property to which the entered value is written to.
InitialValue	String	" "	The value that is displayed on clicking the graphic item or element.
FormatString	String	"ms"	The time unit for presentation of the value.
MaxValue	Integer		The maximum value for the input item.
MinValue	Integer	0	The minimum value for the input item.

Table 131 gives the list of out terminals used for Time Dew.

Table 131. Out terminals of Time Dew

Out terminal	Type	Description
InteractionInProgress	Boolean	This value changes to *True* on opening the direct entry window.
WriteInProgress	Boolean	*True* specifies that a write operation is being executed. This value changes to *False* after the completion of write operation.
ValueToWrite	Integer	Represents the currently entered value. This value is set to InitialValue on opening the direct entry window.
ValidValue	Boolean	This value is *True* when the value entered is a Time value.

Table 132. Formats for TimeDew

Time Unit	Format to be used
Year	'Y'/'y'
Month	'M'
Day	'D'/'d'
Hour	'H'/'h'
Minute	'm'
Second	'S'/'s'
Millisecond	"MS"/"Ms"/"mS"/"ms"

MultiselectionDew

MultiSelectionDew is used for multiple-choice configurations. Various selection items can be added using MultiSelection and ItemEntry functions to the **Items** property. For more information, refer to MultiSelection and ItemContent respectively.

For examples on configuring a MultiSelectionDew, refer to How to use the MultiSelectionDew input item on page 397.

Table 133 gives the list of some of the available properties for MultiselectionDew.

Table 133. MultiselectionDewProperties

Name	Type	Default	Description
Action	Action	SystemDefault	Controls when the write operation is performed. For more information, refer to Session Handling on page 133.
ContentAlignment	Enum	Vertical	To align the item contents horizontally or vertically. The values can be *Horizontal* or *Vertical* respectively.
Items	MultiSelection	Empty	This allows the user to add multiple selection items to the MultiselectionDew.

MultiSelection. The MultiSelection data type is used for configuring multiple selection entries to the content of MultiSelectionDew input item. This requires the configuration of **ItemContent** data type and **ItemState** user enumeration. The user can now achieve the functionality of multiple-selection choices for writing values to the target.

The **ItemContent** datatype allows the user to specify the target and the values to be written to the target. The **Items** property of the MultiSelectionDew input item is of this datatype.

The following function is used to define multiple selection entries.

```
MultiSelection (ItemContent itemContent(), ItemContent
itemContent(),...)
```

itemContent is of data type **ItemContent**. Based on the number of *itemContent* parameters, the MultiSelectionDew input item invokes the same number of buttons. For more information, refer to MultiselectionDew on page 515.

ItemContent. The ItemContent data type is used for configuring each selection entry that appears in the MultiSelectionDew input item. This configuration includes the item name that should be displayed in runtime, the value to be written to the target (aspect object type property), the target to which the value should be written, and the state of the item.

The following function is used to configure the selection entries.

```
ItemContent (PropertyRef Target, Variant Value, String
PresentationName, ItemStatus Status)
```

Target is the aspect object type property to which the value should be written.

Value is a variant type value to be written to *Target*.

PresentationName is a display name for the item.

Status is the state of the item. This can be *Enabled*, *Disabled*, or *Hidden*. If the state is *Enabled*, the value is written to the target on clicking the item. If the state is *Disabled*, no action takes place on clicking the item. If the state is *Hidden*, the item will not be visible.

Password Dew

Feature Pack Functionality _____

This section applies to the System 800xA 5.1 Feature Pack release only.

Password Dew is an input item that takes String type values. The user can enter text values for the items, but cannot view the text in the input item. This displays only the password character instead of the string.

Table 134 gives the list of some available properties for Password Dew.

Table 134. Password Dew Properties

Name	Type	Default	Description
PropertyRef	PropertyRef		Refers to the aspect object property to which the entered value is written to.
MaxNoOfChars	Integer	0	The maximum number of characters allowed to enter.
PasswordChar	String	"."	The password character to be presented while entering the value.
EntryWindowOnTop	Boolean	False	To position the direct entry window on top of the graphic item.

Table 135 gives the list of out terminals used for Password Dew.

Table 135. Out terminals of Password Dew

Out terminal	Type	Description
InteractionInProgress	Boolean	This value changes to *True* on opening the direct entry window.
WriteInProgress	Boolean	*True* specifies that a write operation is being executed. This value changes to *False* after the completion of write operation.
ValueToWrite	String	Represents the currently entered value.
ValidValue	Boolean	This value is always *True*.

_____

Stepsize and limits in the Numeric Direct Entry Windows

The numeric DEWs support interactive operations for increasing and decreasing the value. The number by which the value increases or decreases is determined by the **StepSize** property.

Following are the ways to increase or decrease the value.

• Using spin buttons in the Direct Entry Window.

• Using the up and down arrow keys on the keyboard.

• Holding SHIFT or CTRL while using the up and down arrow keys on the keyboard.

Holding SHIFT key with the up/down arrow keys increases/decreases the value by stepsize * 10.

Holding CTRL key with the up/down arrow keys increases/decreases the value by stepsize / 10.

Appendix B Standard Building Blocks for AC 800M

This appendix explains the following AC 800M building blocks:

- AC 800M Display Elements
- AC 800M Faceplate Elements
- AC 800M Symbols

> ℹ️ The **AC 800M Display Elements**, **AC 800M Faceplate Elements**, and **AC 800M Symbols** exist only if *AC 800M Connect* system extensions are loaded in the system.

> 💡 For information on the common properties for the elements, refer to Common Properties for the Primitives on page 430.

AC 800M Display Elements

AC 800M Display Elements are generic elements, which contains a display element core. AC 800M display element cores are building blocks that contain properties, which can be connected to process values.

These elements are selected from **Toolboxes > AC 800M Display Elements** in the **View** menu of the Graphics Builder.

AC 800M Display Elements in the toolbox contains the following:

- AONote Core
- OpNote Core

AONote Core

AONote Core element displays the status (Severity and Data quality) from an Asset Reporter aspect that exist on the object.

Table 136 shows symbols for Asset Optimization information.

Table 136. Symbols Showing AO Information

Symbol	AO Information
	Severity = 1-250
	Severity = 251-500
	Severity = 501-750
	Severity = 751-1000
	OPCQuality=64 (Uncertain)
	OPCQuality=0 (Bad)

Table 137. AONote Core Properties

Name	Type	Default	Description
OPCQuality	Integer	192	If OPCQuality <> 0 and 64 then no quality indication. Refer to Table 136 for more information.
Severity	Integer	1	If Severity < 1 or Severity > 1000 then no image is presented. Refer to Table 136 for more information.

OpNote Core

OpNote Core element indicates if an operator note is available on the object.

Table 138 shows symbols for OpNote Core.

Table 138. Symbols Showing Operator Notes

Symbol	OpNote Core Information
	OpNote = True (else invisible)

Table 139. OpNote Core Properties

Name	Type	Default	Description
OpNote	Boolean	True	If OpNote = false then no image is presented. Refer to Table 138 for more information.

AC 800M Faceplate Elements

This section describes the AC 800M generic elements which are used as building blocks in the faceplate elements. The AC 800M Fonts and AC 800M Images used in the generic elements are defined as common resources.

These elements are selected from **Toolboxes > AC 800M Faceplate Elements** in the **View** menu of the Graphics Builder.

•   **AC 800M Fonts**:

The AC 800M Fonts in the faceplate elements are controlled by a number of logical fonts, specified in the **AC 800M Font Resources** aspect. Table 140 lists the fonts that exist in the resource aspect:

Table 140. AC 800M Fonts

Name	Family	Size	Weight	Description
FpGroupBox	Tahoma	8	Normal	GroupBox font
FpMessage	Tahoma	9	Bold	Faceplate message font
FpText	Tahoma	8	Normal	Normal text font
FpValue	Arial	9	Bold	Value font

•   **AC 800M Images**:

All images used in the graphic elements are stored in the **AC 800M Image Resources** aspect.

AC 800M faceplate elements in the toolbox contains the following:

•   Bar

•   BarInput

•   BarOutput1

•   BarOutput2

•   CheckBox

•   Deviation

- ErrorMode

- Indicator

- IndicatorBool

- IndicatorBoolRef

- IndicatorBoolOr

- IndicatorCheckBox

- IndicatorInputValue

- InputField

- InputValue

- IOSignalBool

- IOSignalReal

- IOStatusMessage

- PictureAspectLink

- PicturePushButton1

- PicturePushButton2

- RadioButton

- TextAspectLink

- TextLabel

- TextPushButton1

- TextPushButton2

- TrimCurveBool

- TrimCurveReal1

- TrimCurveReal2

- TrimCurveRealBool

Bar

Bar item is a single vertical bar without a scale and range.

Table 141 shows symbols for Bar.

Table 141. Symbols Showing Bar

Symbol
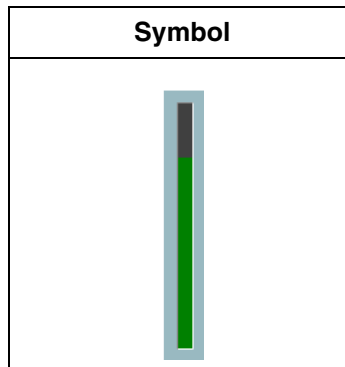
Table 142. Bar Properties

Name	Type	Default	Description
BarColor	Color	Green	Color of the bar.
BarValueRef	PropertyRef	null	Reference to an aspect object property. The value of this aspect will be displayed in the bar.
MaxBarValue	Real	100.0	Maximum value of the bar.
MinBarValue	Real	0.0	Minimum value of the bar.
Orientation	Orientation	Vertical	Direction of the bar, which can be **Vertical** or **Horizontal**.

Table 142. Bar Properties (Continued)

Name	Type	Default	Description
SignalStatus	Integer	192	Signal status of the bar.
Tooltip	String	""	Tooltip to be displayed for the bar.

BarInput

BarInput item has two vertical bars with a scale and range. BarInput is used to describe input value, manual value, and alarm limits for a real signal. This item can also describe the PV and SP values for a regulator.

Table 143 shows symbols for BarInput.

Table 143. Symbols Showing BarInput
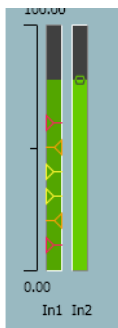
Symbol	BarInput Information
	The left bar contains 8 different limits. Range shows Maximum value (Bar1Max, Bar2Max) and Minimum value (Bar1Min, Bar2Min). Suitable position for the bar is XPos=3, YPos=3, Width=70 and Height=257.

Table 144. BarInput Properties

Name	Type	Default	Description
Bar1Color	Color	InterfaceInput	Color of left bar.
Bar1EnableInput	Boolean	False	Enables the drag handle of the left bar.

Table 144. BarInput Properties (Continued)

Name	Type	Default	Description
Bar1LimitXColor	Color	When X=1, 6, highAlarmSymbol When X=2, 5, , mediumAlarmSymbol When X=3, 4, lowAlarmSymbol When X=7, 8, RelAlarm	Color of the limit.
Bar1LimitXFilled	Boolean	False	Specifies whether the limit is filled or not.
Bar1LimitXStyle	LimitStyleEnum	When X=1, 3, 4, 6, Left When X=2,5, Right When X=7, 8, Invisible	Specifies the limit style. This can be **Right**, **Left**, or **Invisible**.
Bar1LimitXValue	Real	0.0	Value of the limit.
X can be a number from 1 to 8.			
Bar1Max	Real	100.0	Maximum value of the left bar.
Bar1Min	Real	0.0	Minimum value of the left bar.
Bar1SignalStatus	Integer	192	Signal status of the left bar.
Bar1Text	String	"In1"	Text to appear for left bar.
Bar1Tooltip	String	""	Tooltip to be displayed for the left bar.
Bar1ValueRef	PropertyRef	null	Reference to an aspect object property. The value of the property will be displayed in the bar.
Bar2Color	Color	Measure	Color of right bar.

Table 144. BarInput Properties (Continued)

Name	Type	Default	Description
Bar2EnableInput	Boolean	True	Enables the drag handle of the right bar.
Bar2Max	Real	100.0	Maximum value of the right bar.
Bar2Min	Real	0.0	Minimum value of the right bar.
Bar2SignalStatus	Integer	192	Signal status of the right bar.
Bar2Text	String	"In2"	Text to appear for the right bar.
Bar2Tooltip	String	""	Tooltip to be displayed for the right bar.
Bar2ValueRef	PropertyRef	null	Reference to an aspect object property. The value of the property will be displayed in the bar.
Bar2Visible	Boolean	True	Controls the visibility of the right bar.
Fraction	Integer	2	Number of decimals.
RangeVisible	Boolean	True	Specifies the visibility of the range.
ScaleVisible	Boolean	True	Specifies the visibility of the scale.

BarOutput1

BarOutput1 is a single horizontal bar with a scale and range and describes out value and alarm limits for a real signal.

Table 145 shows symbols for BarOutput.

Table 145. Symbols Showing BarOutput1
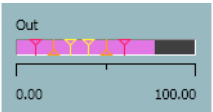
Symbol	BarOutput1 Information
 Out 0.00 100.00	The bar contains 6 different limits. Suitable position for the bar is XPos=97, YPos=183, Width=146 and Height=74.

Table 146. BarOutput1 Properties

Name	Type	Default	Description
BarColor	Color	output	Color of the bar.
BarEnableInput	Boolean	False	Enables the drag handle of the Bar.
BarLimitXColor	Color	highAlarmSymbol	Color of the limit.
BarLimitXFilled	Boolean	False	Specifies whether the limit is filled or not.
BarLimitXStyle	LimitStyleEnum	Up	Specifies the limit style. This can be **Up**, **Down**, or **Invisible**.
BarLimitXValue	Real	0.0	Value of the limit.
BarMax	Real	100.0	Maximum value of the bar.
BarMin	Real	0.0	Minimum value of the bar.
BarSignalStatus	Integer	192	Signal status of the bar.
BarText	String	"In1"	Text to appear for the bar.
BarTooltip	String	""	Tooltip to be displayed for the bar.

Table 146. BarOutput1 Properties (Continued)

Name	Type	Default	Description
BarValueRef	PropertyRef	null	Reference to an aspect object property. The value of the property will be displayed in the bar.
Fraction	Integer	2	Number of decimals.

BarOutput2

BarOutput2 element has two horizontal bars with a scale and range and describes out value for two real signal.

Table 147 shows symbols for BarOutput2.

Table 147. Symbols Showing Bar

Symbol	BarOutput2 Information
	Range shows Maximum (Bar1Max, Bar2Max), Minimum (Bar1Min, Bar2Min). Suitable position for the bar is XPos=97, YPos=147, Width=146 and Height=110.

Table 148. BarOutput2 Properties

Name	Type	Default	Description
Bar1Color	Color	InterfaceInput	Color of the upper bar.
Bar1EnableInput	Boolean	False	Enables the drag handle of the upper bar.
Bar1Max	Real	100.0	Maximum value of the upper bar.

Table 148. BarOutput2 Properties (Continued)

Name	Type	Default	Description
Bar1Min	Real	0.0	Minimum value of the upper bar.
Bar1SignalStatus	Integer	192	Signal status of the upper bar.
Bar1Text	String	"In1"	Text to appear for the upper bar.
Bar1Tooltip	String	""	Tooltip to be displayed for the upper bar.
Bar1ValueRef	PropertyRef	null	Reference to an aspect object property. The value of the property will be displayed in the bar.
Bar1Visible	Boolean	True	Controls the visibility of the upper bar.
Bar2Color	Color	Measure	Color of the lower bar.
Bar2EnableInput	Boolean	True	Enables the drag handle of the lower bar.
Bar2Max	Real	100.0	Maximum value of the lower bar.
Bar2Min	Real	0.0	Minimum value of the lower bar.
Bar2SignalStatus	Integer	192	Signal status of the lower bar.
Bar2Text	String	"In2"	Text to appear for the lower bar.
Bar2Tooltip	String	""	Tooltip to be displayed for the lower bar.
Bar2ValueRef	PropertyRef	null	Reference to an aspect object property. The value of the property will be displayed in the bar.
Fraction	Integer	2	Number of decimals.

CheckBox

CheckBox element is a checkbox with a text. The checkbox is controlled by the property **ValueRef**.

Table 149 shows symbols for CheckBox.

Table 149. Symbols Showing CheckBox
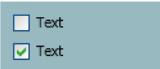
Symbol	CheckBox Information
	ValueRef = False ValueRef = True

Table 150. CheckBox Properties

Name	Type	Default	Description
EnableInput	Boolean	True	Enables interaction.
Text	String	"Text"	Text to appear to the right of the checkbox
Tooltip	String	""	Tooltip to be displayed for the check box.
ValueRef	PropertyRef	null	Reference to an aspect object property. The value of the property is connected to the check box.

Deviation

Deviation element is used for indication of the deviation (PV-SP) in Pid loops.

Table 151 shows symbols for Deviation.

Table 151. Symbols Showing Deviation

Symbol

Table 152. Deviation Properties

Name	Type	Default	Description
BarColor	Color	Deviation	Color of the Bar.
BarMax	Real	100.0	Maximum range of Pv and Sp.
BarMin	Real	0.0	Minimum range of Pv and Sp.
BarOutOfRangeColor	Color	RelAlarm	Color of the Bar when deviation is out of range.
DataQuality	DataQuality	Ok	Data quality indication. This can be **Ok**, **Bad** or **Uncertain**.
Fraction	Integer	0	Number of decimals.
PvValue	Real	0.0	Process value.
RangePercent	Real	10.0	Range of the deviation bar, that is, percentage of the BarMin to BarMax range.
SpValue	Real	0.0	Setpoint value.
Text	String	"Dev"	Text to appear for the deviation.
Unit	String	"%"	Unit to appear for the deviation.

ErrorMode

ErrorMode element is used for indicating error mode in signal objects.

Table 153 shows symbols for ErrorMode.

Table 153. Symbols Showing ErrorMode
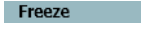
Symbol	ErrorMode Information
Through	ErrorMode = 0
Freeze	ErrorMode = 1
Predetermined	ErrorMode = 2

Table 154. ErrorMode Properties

Name	Type	Default	Description
ErrorMode	Integer	0	Controls the text presented. See Table 153.

Indicator

Indicator element is a boolean indicator with a text. The indicator is controlled by properties **FillColor** and **LineColor**.

Table 155 shows symbols for Indicator.

Table 155. Symbols Showing Indicator

Symbol
☐ Text

Table 156. Indicator Properties

Name	Type	Default	Description
DataQuality	DataQuality	Good	Data quality indication. This can be **Good**, **Bad**, or **Uncertain**.
FillColor	Brush	Transparent	Fill color of the rectangle.
LineColor	Brush	FaceplateFg	Line color of the rectangle.
Text	String	"Text"	Text to appear for the indicator.
TextColor	Brush	FaceplateFg	Color of the text.

IndicatorBool

IndicatorBool element is a boolean indicator with a text. The indicator is controlled by the property **Value**.

Table 157 shows symbols for IndicatorBool.

Table 157. Symbols Showing IndicatorBool

Symbol	IndicatorBool Information
□ Text	Value = False
■ Text	Value = True

Table 158. IndicatorBool Properties

Name	Type	Default	Description
DataQuality	DataQuality	Good	Data quality indication. This can be **Good**, **Bad**, or **Uncertain**.
Inverted	Boolean	False	Inverts indication of Value.
OnColor	Color	Green	Color of rectangle if "On".

Table 158. IndicatorBool Properties (Continued)

Name	Type	Default	Description
Text	String	"Text"	Text to appear for the indicator.
Value	Boolean	False	Value to be presented by the indicator. Refer to Table 157.

IndicatorBoolRef

IndicatorBoolRef element is a boolean indicator with a text. The indicator is controlled by the property **ValueRef**.

Table 159 shows symbols for IndicatorBoolRef.

Table 159. Symbols Showing IndicatorBoolRef

Symbol	IndicatorBoolRef Information
☐ Text	ValueRef#Value = False
▦ Text	ValueRef#Value = True

Table 160. IndicatorBoolRef Properties

Name	Type	Default	Description
Inverted	Boolean	False	Inverts indication of ValueRef.
OnColor	Color	Green	Color of rectangle if "On".
SignalStatus	Integer	192	Signal status.
Text	String	"Text"	Text to appear for the indicator.
ValueRef	PropertyRef	null	Value to be presented by the indicator. Refer to Table 159.

IndicatorBoolOr

IndicatorBoolOr element is a boolean indicator with text. This element is controlled by the property **ValueRef1** (PropertyRef) OR **ValueRef2** (PropertyRef).

Table 161 shows symbols for IndicatorBoolOr.

Table 161. Symbols Showing IndicatorBoolOr

Symbol	IndicatorBoolOr Information
□ Text	ValueRef1#Value and ValueRef2#Value = False
■ Text	ValueRef1#Value or ValueRef2#Value = True

Table 162. IndicatorBoolOr Properties

Name	Type	Default	Description
Inverted	Boolean	False	Inverts indication of the result of ValueRef1 and ValueRef2.
OnColor	Color	Green	Color of rectangle if "On".
Text	String	"Text"	Text to appear for the indicator.
ValueRef1	PropertyRef	null	Value to present the indicator. Refer to Table 161.
ValueRef2	PropertyRef	null	Value to be presented by the indicator. Refer to Table 161.

IndicatorCheckBox

IndicatorCheckBox element is a boolean indicator combined with check box and is used in ProcessObjInsumLib for trips and warnings.

Table 163 shows symbols for IndicatorCheckBox.

Table 163. Symbols Showing IndicatorCheckBox
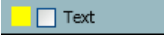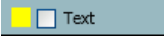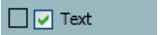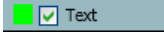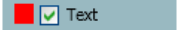
Symbol	ValueRef	Signal	SignalLatched
☐☐ Text	False	False	False
🟩☐ Text	False	False	True
🟨☐ Text	False	True	False
🟨☐ Text	False	True	True
☐☑ Text	True	False	False
☐☑ Text	True	False	True
🟩☑ Text	True	True	False
🟥☑ Text	True	True	True

Table 164. IndicatorCheckBox Properties

Name	Type	Default	Description
DataQuality	DataQuality	Good	Data quality indication. This can be **Good**, **Bad**, or **Uncertain**.
Signal	Boolean	False	Signal value. See Table 163.
SignalLatched	Boolean	False	Signal latched value. See Table 163.
Text	String	"Text"	Text to appear for the check box.

Table 164. IndicatorCheckBox Properties (Continued)

Name	Type	Default	Description
Tooltip	String	""	Tooltip to be displayed for the check box.
ValueRef	PropertyRef	null	Check box value.

IndicatorInputValue

IndicatorInputValue element is a boolean indicator combined with input value field.

Table 165 shows symbols for IndicatorInputValue.

Table 165. Symbols Showing IndicatorInputValue

Symbol	IndicatorInputValue Information
☐ Text 77.80 %	BoolRef = False ValueRef = 77.8
■ Text 77.80 %	BoolRef = True ValueRef = 77.8

Table 166. IndicatorInputValue Properties

Name	Type	Default	Description
BoolRef	PropertyRef	null	Reference to an aspect object property. The value of the property will be displayed as the indicator value. See Table 165.
DataType	DataTypeEnum	Real	Data type of the input value. This can be **Boolean**, **DateTime**, **Integer**, **Real**, **String**, or **Time**.
EnableInput	Boolean	True	Enables interaction.

Table 166. IndicatorInputValue Properties (Continued)

Name	Type	Default	Description
Fraction	Integer	2	Number of decimals if the data type is *Real*.
Function	FunctionEnum	Interact	Select **Interact** when interaction to the value should be possible. Select **Show** when only the value shall be presented.
InputMax	Real	100.	Maximum input value if the data type is *Real* or *Integer*.
InputMin	Real	0.	Minimum input value if the data type is *Real* or *Integer*.
Inverted	Boolean	False	Inverts the value of BoolRef.
OnColor	Color	Green	Color of rectangle if "On".
Text	String	"Text"	Text to appear for the indicator.
ToolTip	String	"Tooltip"	Tooltip to be displayed for the indicator.
Unit	String	"%"	Unit to be presented for the indicator.
ValueRef	PropertyRef	null	Reference to an aspect object property. The value of the property will be displayed as the input value. See Table 165.

InputField

The InputField element is used to enable an input operation. The input field is controlled by the property **ValueRef** and can accept an input of different data types controlled by the property **DataType**.

Table 167 shows symbols for InputField.

Table 167. Symbols Showing InputField
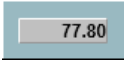
Symbol	Symbol Information
77.80	ValueRef = 77.8

Table 168. InputField Properties

Name	Type	Default	Description
DataType	DataTypeEnum	Real	Data type of ValueRef. This can be **Boolean**, **DateTime**, **Integer**, **Real**, **String**, or **Time**.
EnableInput	Boolean	True	Enables interaction.
Fraction	Integer	2	Number of decimals if data type is *Real*.
Function	FunctionEnum	Interact	Select **Interact** when interaction to the value should be possible. Select **Show** when the value only shall be presented.
InputMax	Real	100.	Maximum input value if data type is *Real* or *Integer*.
InputMin	Real	0.	Minimum input value if data type is *Real* or *Integer*.
SignalStatus	Integer	192	Signal status.
StepSize	Real	1.0	Size of change when changing the value by stepping.

Table 168. InputField Properties (Continued)

Name	Type	Default	Description
ToolTip	String	"Tooltip"	Input value tooltip.
ValueRef	PropertyRef	null	Reference to an aspect object property. The value of the property will be displayed as the input value. See Table 167.

InputValue

InputValue element is an input field combined with text (left) and unit (right). The input value field is controlled by the property **ValueRef**.

Table 169 shows symbols for InputValue.

Table 169. Symbols Showing InputField

Symbol	Symbol Information
Text 77.80 %	Function = Interact EnableInput = True
Text 77.80 %	Function = Interact EnableInput = False
Text 77.80 %	Function = Show
Text 77.80 %	ValueBgColorEnable = True ValueBgColor = Red

Table 170. InputValue Properties

Name	Type	Default	Description
DataType	DataTypeEnum	Real	Data type of ValueRef. This can be **Real**, **Integer**, **String**, **Time**, **DateTime**, or **Boolean**.
EnableInput	Boolean	True	Enables interaction.
Fraction	Integer	2	Number of decimals if data type real.
Function	FunctionEnum	Interact	Select **Interact** when interaction to the value should be possible. Select **Show** when the value only shall be presented.
InputMax	Real	100.	Maximum input value if data type is *Real* or *Integer*.
InputMin	Real	0.	Minimum input value if data type is *Real* or *Integer*.
SignalStatus	Integer	192	Signal status.
StepSize	Real	1.0	Size of change when changing the value by stepping.
Text	String	"Text"	Text to appear for the input value.
ToolTip	String	"Tooltip"	Tooltip to be displayed for the input value.
Unit	String	"%"	Unit to be presented for the input value.
ValueBgColor	Color	Red	Background color when ValueBgColorEnable = True
ValueBgColorEnable	Boolean	False	Controls the background color.
ValueRef	PropertyRef	null	Reference to an aspect object property. The value of the property will be displayed as the input value. See Table 169.

IOSignalBool

IOSignalBool element displays IO signal status (BoolIO), and buttons for forcing IO value and resetting of latches.

Table 171 shows symbols for IOSignalBool.

Table 171. Symbols Showing IOSignalBool

Table 172. IOSignalBool Properties

Name	Type	Default	Description
Caption	String	""	Group box header text.
ForcedEnabled	Boolean	True	Enables forced buttons.
ForcedRef	PropertyRef	null	Signal Forced target.
IOValueRef	PropertyRef	null	Signal IOValue target.
ResetEnabled	Boolean	False	Enables input of reset button (right).
ResetRef	PropertyRef	null	Reset button target.
ResetType	ResetTypeEnum	None	Determines icon of reset button. This can be **None**, **Override**, **Vote**, **Quality**, or **Latch**.
ResetVisible	Boolean	True	Controls visibility of Reset button.
SignalName	String	""	Name of signal.
SignalType	SignalTypeEnum	In	Input or Output signal. This can be **In** or **Out**.

Table 172. IOSignalBool Properties (Continued)

Name	Type	Default	Description
StatusRef	PropertyRef	null	Signal Status target.
ValueRef	PropertyRef	null	Signal Value target.

IOSignalReal

IOSignalReal element displays IO signal status (RealIO), and buttons for forcing IO value and resetting of latches.

Table 173 shows symbols for IOSignalReal.

Table 173. Symbols Showing IOSignalReal
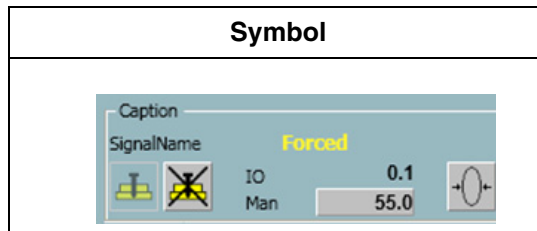
Symbol

Table 174. IOSignalReal Properties

Name	Type	Default	Description
Caption	String	""	Group box header text.
ForcedEnabled	Boolean	True	Enables forced buttons.
ForcedRef	PropertyRef	null	Signal Forced target.
Fraction	Integer	2	Number of decimals
IOValueRef	PropertyRef	null	Signal IOValue target.
ResetEnabled	Boolean	False	Enables input of reset button (right).
ResetRef	PropertyRef	null	Reset button target.

Table 174. IOSignalReal Properties (Continued)

Name	Type	Default	Description
ResetType	ResetTypeEnum	None	Determines icon of reset button. This can be **None**, **Override**, **Vote**, **Quality**, or **Latch**.
ResetVisible	Boolean	True	Controls visibility of Reset button.
SignalName	String	""	Name of signal.
SignalType	SignalTypeEnum	In	Input or Output signal. This can be **In** or **Out**.
StatusRef	PropertyRef	null	Signal Status target.
ValueRef	PropertyRef	null	Signal Value target.

IOStatusMessage

IOStatusMessage element displays the status of IO signal. For redundant IO, primary and secondary modules are represented in the form of two rectangles (in the picture, the left rectangle is the primary module).

Table 175 shows symbols for IOStatusMessage.

Table 175. Symbols Showing IOStatusMessage
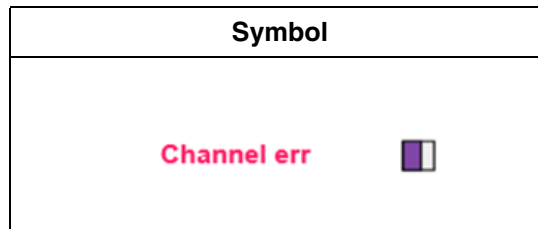
Table 176. IOStatusMessage Properties

Name	Type	Default	Description
Status	PropertyRef	null	I/O status message.

PictureAspectLink

PictureAspectLink is an aspect link button element with an image on top. The button is dimmed if the property **EnableInput**=false .

Table 177 shows symbols for PictureAspectLink.

Table 177. Symbols Showing PictureAspectLink

Symbol
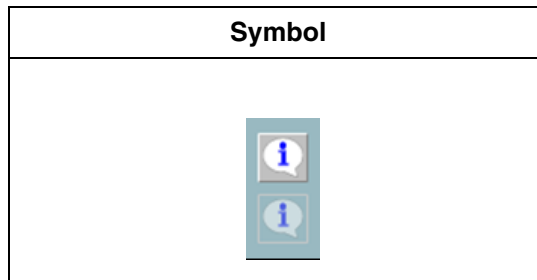
Table 178. PictureAspectLink Properties

Name	Type	Default	Description
AspectView	ViewReference	Null	Aspect view reference.
EnableInput	Boolean	True	Enables interaction.
Image	Image	AC 800M Images:Empty	Image presented.
ToolTip	String	"PictureAspect Link"	Tooltip to be displayed for the symbol.

PicturePushButton1

PicturePushButton1 is a Push button element for one target with image on top. The button is dimmed if property **EnableInput**=false.

Table 179 shows symbols for PicturePushButton1.

Table 179. Symbols Showing PicturePushButton1

Symbol
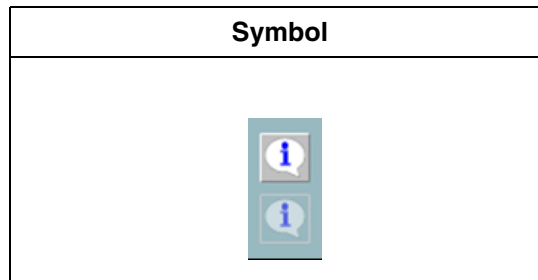
Table 180. PicturePushButton1 Properties

Name	Type	Default	Description
EnableInput	Boolean	True	Enables interaction.
Image	Image	AC 800M Images:Empty	Image presented.
Target	PropertyRef	null	Reference to an aspect object property. The *Value* will be written to this property.
Tooltip	String	"PicturePushButton"	Tooltip to be displayed for the symbol.
Value	Variant	Empty	Value to be written to *Target*.

PicturePushButton2

PicturePushButton2 is a Push button element for two targets with image on top. The button is dimmed if property **EnableInput**=false.

Table 181 shows symbols for PicturePushButton2.

Table 181. Symbols Showing PicturePushButton2

Symbol
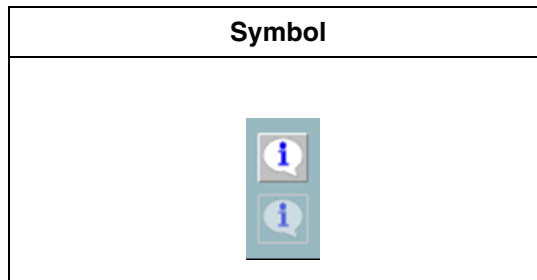
Table 182. PicturePushButton2 Properties

Name	Type	Default	Description
EnableInput	Boolean	True	Enables interaction
Image	Image	AC 800M Images:Empty	Image presented.
Target1	PropertyRef	null	Reference to an aspect object property. The *Value1* will be written to this property.
Target2	PropertyRef	null	Reference to an aspect object property. The *Value2* will be written to this property.
ToolTip	String	"PicturePush Button2"	Tooltip to be displayed for the symbol.
Value1	Variant	Empty	Value to be written to *Target1*.
Value2	Variant	Empty	Value to be written to *Target2*.

RadioButton

RadioButton element is a radio button with a text. The radio button is controlled by the property **Target**. Table 183 shows symbols for RadioButton.

Table 183. Symbols Showing RadioButton

Symbol	Symbol Information
○ Text	Target < Value or Target > Value
⊙ Text	Target = Value

Table 184. RadioButton Properties

Name	Type	Default	Description
EnableInput	Boolean	True	Enables interaction.
Target	PropertyRef	null	Reference to an aspect object property. The *Value* will be written to this property.
Text	String	"Text"	Text to appear for the symbol.
ToolTip	String	""	Tooltip to be displayed for the symbol.
Value	Variant	Empty	Value to be written to *Target*.

TextAspectLink

TextAspectLink is an aspect link button element with a text. The button does not allow interaction if property **EnableInput**=false.

Table 185 shows symbols for TextAspectLink.

Table 185. Symbols Showing TextAspectLink
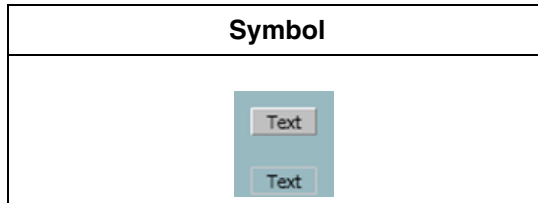
Symbol

Table 186. TextAspectLink Properties

Name	Type	Default	Description
AspectView	ViewReference	null	Aspect view reference.
EnableInput	Boolean	True	Enables interaction.
Text	String	""	Text to appear in the symbol.
ToolTip	String	"TextAspectLink"	Tooltip to be displayed for the symbol.

TextLabel

TextLabel is a text to represent the labels in faceplate elements.

Figure 186 shows symbols for TextLabel.

Table 187. Symbols Showing TextLabel

Symbol
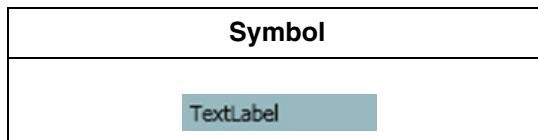
Table 188. TextLabel Properties

Name	Type	Default	Description
DataQuality	DataQuality	Good	Data quality indication. This can be **Good**, **Bad**, or **Uncertain**.
HAlign	HorizontalAlignment	Near	Horizontal alignment of text.
Text	String	"TextLabel"	Text to appear in the symbol.

TextPushButton1

TextPushButton1 is a push button element for one target with a text. The button is transparent if property **EnableInput**=false.

Table 189 shows symbols for TextPushButton1.

Table 189. Symbols Showing TextPushButton1

Symbol
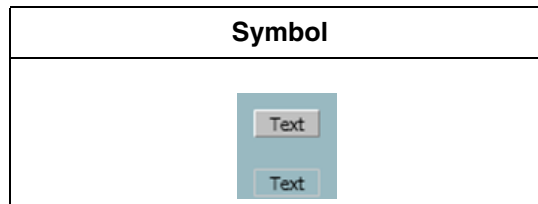
Table 190. TextPushButton1 Properties

Name	Type	Default	Description
EnableInput	Boolean	True	Enables interaction.
Target	PropertyRef	null	Reference to an aspect object property. The *Value* will be written to this property.
Text	String	""	Text to appear in the button.
ToolTip	String	"TextPushButton1"	Tooltip to be displayed for the button.
Value	Variant	Empty	Value to be written to *Target*.

TextPushButton2

TextPushButton2 is a push button element for two targets with a text. The button is transparent if property **EnableInput** = false (see lower button in Table 191).

Table 191 shows symbols for TextPushButton2.

Table 191. Symbols Showing TextPushButton2

Symbol
Text Text
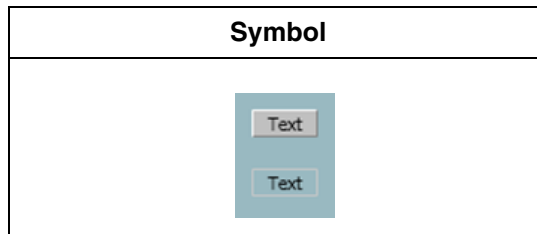
Table 192. TextPushButton2 Properties

Name	Type	Default	Description
EnableInput	Boolean	True	Enables interaction.
Target1	PropertyRef	null	Reference to an aspect object property. The *Value1* will be written to this property.
Target2	PropertyRef	null	Reference to an aspect object property. The *Value 2* will be written to this property.
Text	String	""	Text to appear in the button.
ToolTip	String	"TextPushButton2"	Tooltip to be displayed for the button.
Value1	Variant	Empty	Value to be written to *Target1*.
Value2	Variant	Empty	Value to be written to *Target2*.

TrimCurveBool

TrimCurveBool element is a trim curve created for two boolean values, two vertical scales, one for each boolean curve. Each curve can be enabled/disabled through the T1-T2 check boxes.

The background grid pattern can be enabled/disabled through the Grid check box.

The time scale range can be modified through the lower left input field.

Table 193 shows symbols for TrimCurveBool.

Table 193. Symbols Showing TrimCurveBool
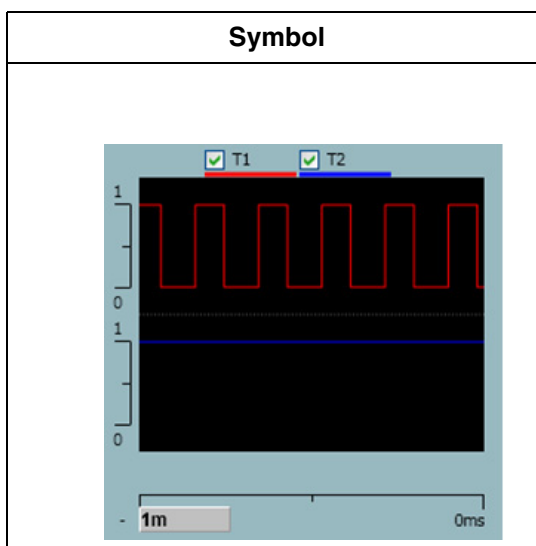
Table 194. TrimCurveBool Properties

Name	Type	Default	Description
In1Color	Color	Red	Color of curve 1.
In1MaxValueText	String	"1"	Maximum value text of curve 1.
In1MaxValueText	String	"0"	Minimum value text of curve 1.
In1Text	String	"T1"	Name of curve 1.

Table 194. TrimCurveBool Properties (Continued)

Name	Type	Default	Description
In1Value	HistoryReference	null	Value for curve 1.
In1Visible	Boolean	True	Show curve 1.
In2Color	Color	Blue	Color of curve 2.
In2MaxValueText	String	"1"	Maximum value text of curve 2.
In2MinValueText	String	"0"	Minimum value text of curve 2.
In2Text	String	"T2"	Name of curve 2.
In2Value	HistoryReference	null	Value for curve 2.
In2Visible	Boolean	True	Show curve 2.
TimeScale	PropertyRef	null	Time scale duration.

TrimCurveReal1

TrimCurveReal1 element is a trim curve created for four real values with one vertical scale to the left. Each curve can be enabled/disabled through the T1-T4 check boxes.

The background grid pattern can be enabled/disabled through the Grid check box.

The time scale range can be modified through the lower left input field.

Table 195 shows symbols for TrimCurveReal1.

Table 195. Symbols Showing TrimCurveReal1

Symbol

Table 196. TrimCurveReal1 Properties

Name	Type	Default	Description
InXColor	Color	When X=1, Red When X=2, Blue When X=3, Yellow When X=4, Green	Color of curve, where X can be a number from 1 to 4.
InXMax	Real	100	Maximum value of the curve, where X can be a number from 1 to 4.
InXMin	Real	0	Minimum value of the curve, where X can be a number from 1 to 4
InXText	String	When X=1, "T1" When X=2, "T2" When X=3. "T3" When X=4, "T4"	Name of curve, where X can be a number from 1 to 4.

Table 196. TrimCurveReal1 Properties (Continued)

Name	Type	Default	Description
InXValue	HistoryReference	null	Value for curve, where X can be a number from 1 to 4.
InXVisible	Boolean	True	Controls the visibility of the curve, where X can be a number from 1 to 4.
LeftScaleFraction	Integer	1	Number of decimals on the left scale.
LeftScaleUnit	String	""	Unit on left scale.
TimeScale	PropertyRef	null	Time scale duration.

TrimCurveReal2

TrimCurveReal2 element is a trim curve created for four real values with two vertical scales one each on the left and right side. Each curve can be enabled/disabled through the T1-T4 check boxes.

The background grid pattern can be enabled/disabled through the Grid check box.

The time scale range can be modified through the lower left input field.

Table 197 shows symbols for TrimCurveReal2.

Table 197. Symbols Showing TrimCurveReal2

Symbol

Table 198. TrimCurveReal2 Properties

Name	Type	Default	Description
InXColor	Color	When X=1, Red When X=2, Blue When X=3, Yellow When X=4, Green	Color of curve, where X can be a number from 1 to 4.
InXMax	Real	100	Maximum value of the curve, where X can be a number from 1 to 4.
InXMin	Real	0	Minimum value of the curve, where X can be a number from 1 to 4.
InXText	String	When X=1, "T1" When X=2, "T2" When X=3. "T3" When X=4, "T4"	Name of curve, where X can be a number from 1 to 4.

Table 198. TrimCurveReal2 Properties (Continued)

Name	Type	Default	Description
InXUseLeftScale	Boolean	When X=1, True When X=2, True When X=3, False When X=4, False	Assign curve to the left scale, where X can be a number from 1 to 4.
InXValue	HistoryReference	null	Value for curve, where X can be a number from 1 to 4.
InXVisible	Boolean	True	Controls the visibility of the curve, where X can be a number from 1 to 4.
LeftScaleFraction	Integer	1	Number of decimals on the left scale.
LeftScaleUnit	String	""	Unit on left scale.
RightScaleFraction	Integer	1	Number of decimals on the right scale.
RightScaleUnit	String	""	Unit on the right scale.
TimeScale	PropertyRef	null	Time scale duration.

TrimCurveRealBool

TrimCurveRealBool element is a trim curve created for two real and one boolean values, one vertical scale for the real curve and one vertical scale for the boolean curve. Each curve can be enabled/disabled through the T1-T3 check boxes.

The background grid pattern can be enabled/disabled through the Grid check box.

The time scale range can be modified through the lower left input field.

Table 199 shows symbols for TrimCurveRealBool.

Table 199. Symbols Showing TrimCurveRealBool

Symbol
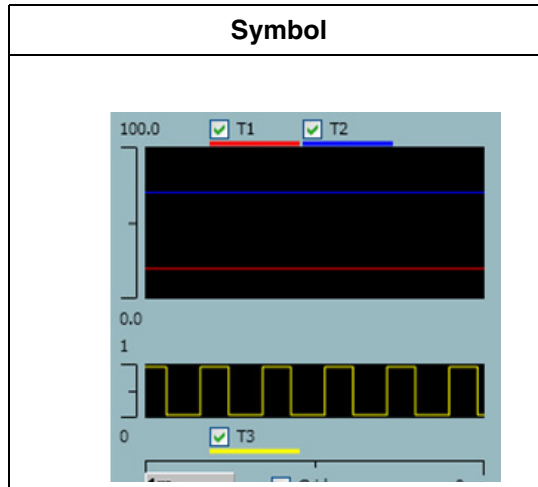
Table 200. TrimCurveRealBool Properties

Name	Type	Default	Description
InXRealColor	Color	When X=1, Red When X=2, Blue When X=3, Yellow	Color of curve, where X can be a number from 1 to 3.
InXRealMax	Real	100	Maximum value of curve, where X can be a number from 1 to 3.
InXRealMin	Real	0	Minimum value of curve, where X can be a number from 1 to 3.
InXRealText	String	When X=1, "T1" When X=2, "T2" When X=3. "T3"	Name of curve, where X can be a number from 1 to 3.
InXRealValue	HistoryReference	null	Value for curve, where X can be a number from 1 to 3.

Table 200. TrimCurveRealBool Properties (Continued)

Name	Type	Default	Description
InXRealVisible	Boolean	True	Controls the visibility of the curve, where X can be a number from 1 to 3.
LeftScaleFraction	Integer	1	Number of decimals on the left scale.
LeftScaleUnit	String	""	Unit on left scale.
TimeScale	PropertyRef	null	Time scale duration.

AC 800M Symbols

The Display Elements, Icon, and Reduced Icon contain a symbol which represents the functionality of the object. Some of the symbols are static, and some are dynamic, that is, they have connections to the controller variables.

The same symbol is used in both Display Element Icon and Display Element Reduced Icon for an object.

The symbols use a logical color definition table named **AC 800M Symbol Colors** (in the **Workplace Structure**).

The AC 800M symbols are selected from **Toolboxes > AC 800M Symbols** in the **View** menu of the Graphics Builder.

Table 201 specifies the common properties of the symbols.

Table 201. Symbol Properties

Property	Value
BackColor	Transparent
PresentationMode	FreeResize
Height	72
Width	72

The AC 800M Symbols also include a number of sub symbols which are used as building blocks in the symbols. The sub types have the prefix *Base*.

AC 800M symbols in the toolbox contains the following:

- DecoupleFilter
- Delay
- EquipProceduemplate
- ForcedSignals
- GroupStartAnd
- GroupStartHead

- GroupStartObjectTemplate

- GroupStartOr

- GroupStartStandBy

- GroupStartStep

- InsumBreaker

- LeadLag

- Level

- ManualAuto

- Max

- Max4

- Mimo

- Min

- Min4

- MotorBi

- MotorInsumBasic

- MotorInsumExtended

- MotorUni

- MotorValve

- Pid

- PidCascadeLoop

- PidForwardLoop

- PidMidrangeLoop

- PidOverrideLoop

- PidSingleLoop

- PulseWidth

- SDLevel

- Select

- Select4

- SFC2D

- SignalBool

- SignalReal

- SignalSupervision

- SystemDiagnostics

- ThreePos

- Uni

- ValveUni

- Vote

- Base Generic Elements

DecoupleFilter

This is a static symbol describing the functionality of DecoupleFilter objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 202 shows symbols used for indication of DecoupleFilter.

Table 202. Symbols Showing DecoupleFilter

Symbol

Table 203. DecoupleFilter Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

Delay

This is a static symbol describing the functionality of Delay objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 204 shows symbols used for indication of Delay.
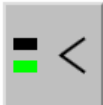
Table 204. Symbols Showing Delay
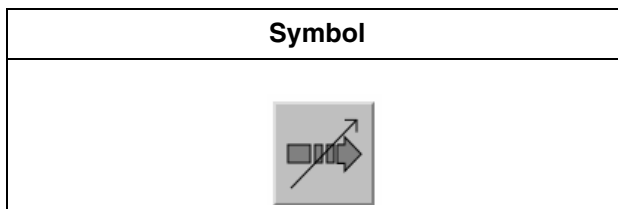
Symbol

Table 205. Delay Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

EquipProceduemplate

This is a static symbol describing the functionality of EquipProceduemplate objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 206 shows symbols used for indication of EquipProceduemplate.

Table 206. Symbols Showing EquipProceduemplate
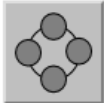
Symbol

Table 207. EquipProceduemplate Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

ForcedSignals

This is a static symbol describing the functionality of ForcedSignals objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 208 shows symbols used for indication of ForcedSignals.

Table 208. Symbols Showing ForcedSignals
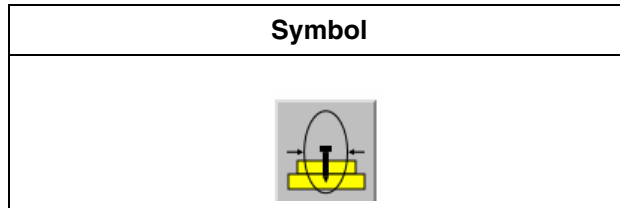
Symbol

Table 209. ForcedSignals Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

GroupStartAnd

This is a symbol describing the functionality of GroupStartAnd objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 210 shows symbols used for indication of GroupStartAnd.

Table 210. Symbols Showing ForcedSignals
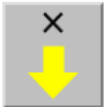
Symbol	Symbol Information
	GroupStartMode=0
	GroupStartMode=1
	GroupStartMode=2

Table 211. ForcedSignals Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
GroupStartMode	Integer	0	Size and color of symbol. Refer Table 210.

GroupStartHead

This is a symbol describing the functionality of GroupStartHead objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 212 shows symbols used for indication of GroupStartHead.

Table 212. Symbols Showing GroupStartHead
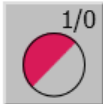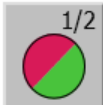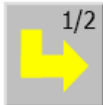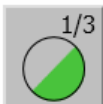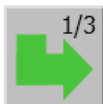
CfgType=Object	CfgType=Arrow	Symbol Information
1/0	1/0	GroupStartMode=0 TotalNoOfSteps=0
1/2	1/2	GroupStartMode=1 TotalNoOfSteps=2
1/3	1/3	GroupStartMode=2 TotalNoOfSteps=3

Table 213. GroupStartHead Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
CfgType	SymbolTypeEnum	Object	Type of symbol. This can be **Object** or **Arrow**.
GroupStartMode	Integer	0	Size and color of symbol.Refer Table 212.
TotalNoOfSteps	Integer	0	Total number of steps. Refer Table 212.

GroupStartObjectTemplate

This is a symbol describing the functionality of GroupStartObjectTemplate objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 214 shows symbols used for indication of GroupStartObjectTemplate.

Table 214. Symbols Showing GroupStartObjectTemplate
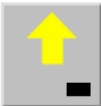
Symbol	Symbol Information
	GroupStartMode=0 Started=false
	GroupStartMode=1 Started=false
	GroupStartMode=2 Started=false
	GroupStartMode=0 Started=true

Table 215. GroupStartObjectTemplate Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
GroupStartMode	Integer	0	Size and color of symbol. Refer Table 214.
Started	Boolean	False	Controls the color of the lower right rectangle. Refer Table 214.

GroupStartOr

This is a symbol describing the functionality of GroupStartOr objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 216 shows symbols used for indication of GroupStartOr.

Table 216. Symbols Showing GroupStartOr
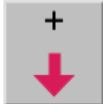
Symbol	Symbol Information
	GroupStartMode=0

Table 216. Symbols Showing GroupStartOr (Continued)
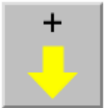
Symbol	Symbol Information
	GroupStartMode=1
	GroupStartMode=2

Table 217. GroupStartOr Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
GroupStartMode	Integer	0	Size and color of symbol. Refer Table 216.

GroupStartStandBy

This is a symbol describing the functionality of GroupStartStandBy objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 218 shows symbols used for indication of GroupStartStandBy.

Table 218. Symbols Showing GroupStartStandBy

Symbol	Symbol Information
	GroupStartMode=0
	GroupStartMode=1
	GroupStartMode=2

Table 219. GroupStartStandBy Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
GroupStartMode	Integer	0	Size and color of symbol. Refer Table 218.

GroupStartStep

This is a symbol describing the functionality of GroupStartStep objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 220 shows symbols used for indication of GroupStartStep.

Table 220. Symbols Showing GroupStartStep
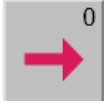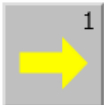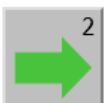
Symbol	Symbol Information
0	GroupStartMode=0 MaxNoOfSteps=0
1	GroupStartMode=1 MaxNoOfSteps=1
2	GroupStartMode=2 MaxNoOfSteps=2

Table 221. GroupStartStep Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
GroupStartMode	Integer	0	Size and color of symbol. Refer Table 220.
MaxNoOfSteps	Integer	0	Maximum number of steps. Refer Table 220.

InsumBreaker

This is a symbol describing the functionality of InsumBreaker objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 222 shows symbols used for indication of InsumBreaker.

Table 222. Symbols Showing InsumBreaker

Symbol	Symbol Information
	Open=False, Closed=False Open=True, Closed=False Open=True, Closed=True
	Open=false Closed=true

Table 223. InsumBreaker Properties

Name	Type	Default	Description
Open	Boolean	False	InsumBreaker Open, refer to Table 222.
Closed	Boolean	False	InsumBreaker Closed, refer to Table 222.

LeadLag

This is a static symbol describing the functionality of LeadLag objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 224 shows symbols used for indication of LeadLag.

Table 224. Symbols Showing LeadLag
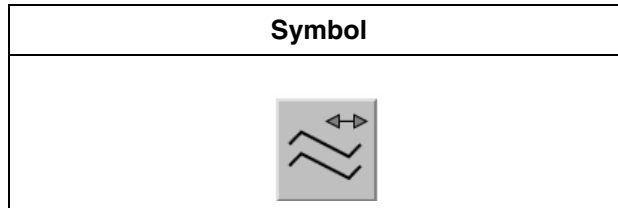
Symbol

Table 225. LeadLag Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

Level

This is a static symbol describing the functionality of Level objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 226 shows symbols used for indication of Level.

Table 226. Symbols Showing Level
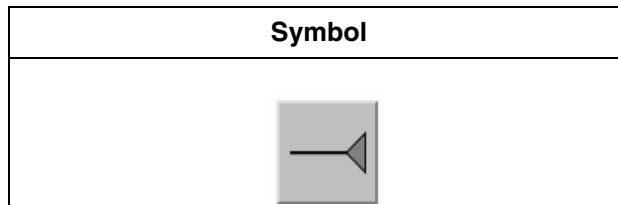
Symbol

Table 227. Level Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

ManualAuto

This is a symbol describing the functionality of ManualAuto objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 228 shows symbols used for indication of ManualAuto.

Table 228. Symbols Showing ManualAuto

Symbol	Symbol Information
	Auto=false
	Auto=true

Table 229. ManualAuto Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.

Table 229. ManualAuto Properties (Continued)

CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
Auto	Boolean	False	Specifies if the symbol is in Manual or auto mode, refer to Table 228.

Max

This is a symbol describing the functionality of Max objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 230 shows symbols used for indication of Max.

Table 230. Symbols Showing Max

Symbol	Symbol Information
	In2Selected=false
	In2Selected=true

Table 231. Max Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.

Table 231. Max Properties (Continued)

CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
In2Selected	Boolean	False	Box 2 selected. Refer to Table 230.

Max4

This is a symbol describing the functionality of Max4 objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 232 shows symbols used for indication of Max4.

Table 232. Symbols Showing Max4

Symbol	Symbol Information
	Selected (1-4) controls the color of the 4 boxes Selected=2

Table 233. Max4 Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
Selected	Integer	0	Selects the specific box, refer to Table 232.

Mimo

This is a static symbol describing the functionality of Mimo objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 234 shows symbols used for indication of Mimo22CC, Mimo41CC and Mimo44CC.

Table 234. Symbols Showing Mimo
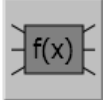
Symbol

Table 235. Mimo Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

Min

This is a symbol describing the functionality of Min objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 236 shows symbols used for indication of Min.

Table 236. Symbols Showing Min

Symbol	Symbol Information
	In2Selected=false
	In2Selected=true

Table 237. Min Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
In2Selected	Boolean	False	Box 2 selected. Refer to Table 236.

Min4

This is a symbol describing the functionality of Min4 objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

Table 238 shows symbols used for indication of Min4.

Table 238. Symbols Showing Min4

| Symbol	Symbol Information
	Selected (1-4) controls the color of the 4 boxes Selected=2

Table 239. Min4 Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
Selected	Integer	0	Selects the specific box, refer to Table 238.

MotorBi

This is a symbol describing the functionality of MotorBi objects with two feedbacks. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

The **CfgType** property controls the appearance of the symbol. Figure 247 shows symbols used for indication of MotorBi.
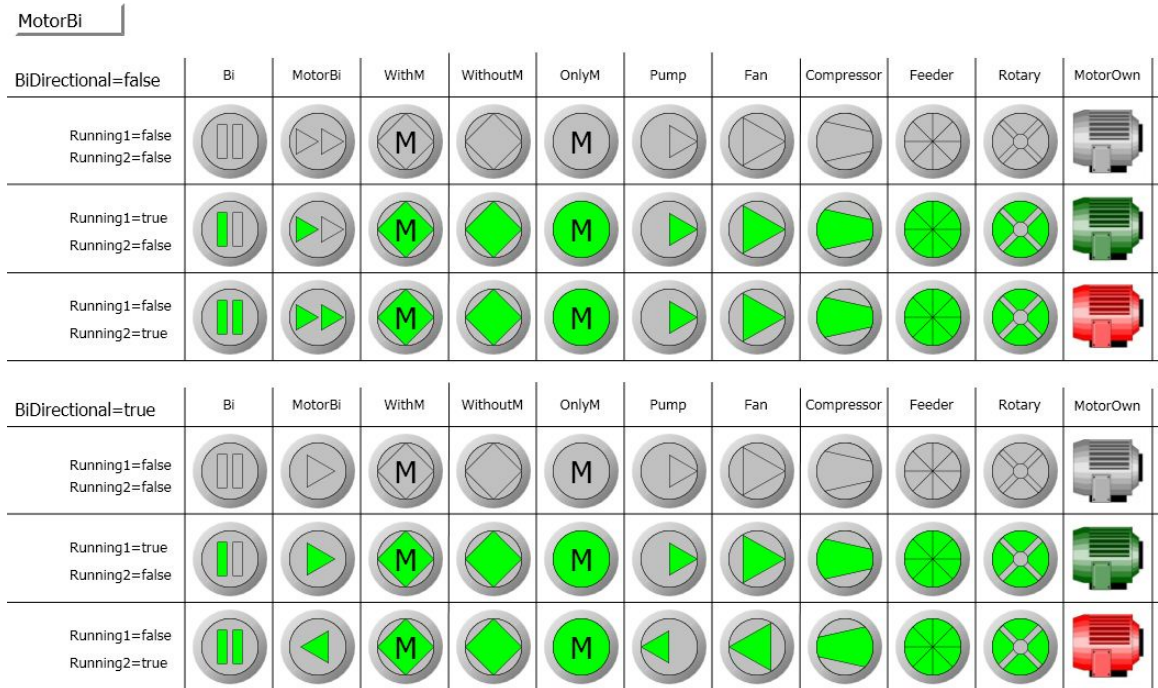
Figure 247. MotorBi Symbols

Table 240. MotorBi Properties

Name	Type	Default	Description
BiDirectional	Boolean	False	Specifies if the motor is bidirectional or not.

Table 240. MotorBi Properties (Continued)

CfgDirection	Direction	Right	Controls the direction of the symbol. Used by MotorBi, Pump, Fan and Compressor. This can be **Up**, **Down**, **Left**, or **Right**.
CfgFrameWidth	Integer	8	Frame width of the symbol.
CfgRunning1Image	Image	-	Own image showed when Running1= True. Used when **CfgType**=MotorOwn.
CfgRunning2Image	Image	-	Own image showed when Running2= True. Used when **CfgType**=MotorOwn.
CfgStopImage	Image	-	Own image showed for stopped motor. Used when **CfgType**=MotorOwn.
CfgType	SymbolTypeEnum	Bi	Type of symbol. The types can be **Bi, Compressor, Fan, Feeder, Motor_WithM**, **Motor_WithoutM**, **Motor_OnlyM**, **MotorBi**, **MotorOwn**, **Pump**, or **Rotary**. Refer to Figure 247.
Running1	Boolean	False	Running first position, motor running with low speed or forward. Refer to Figure 247.
Running2	Boolean	False	Running second position, motor running with high speed or backward. Refer to Figure 247.

MotorInsumBasic

This is a symbol describing the functionality of MotorInsumBasic objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

The **CfgType** property controls the appearance of the symbol. Figure 248 shows symbols used for indication of MotorInsumBasic.

Figure 248. MotorInsumBasic Symbols

Table 241. MotorInsumBasic Properties

Name	Type	Default	Description
CfgDirection	Direction	Right	Controls the direction of the symbol. Used by MotorInsum, Pump, Fan and Compressor. This can be **Up**, **Down**, **Left**, or **Right**.
CfgFrameWidth	Integer	8	Framewidth of the symbol.
CfgType	SymbolTypeEnum	MotorInsum	Type of symbol. The types can be **MotorInsum**, **Motor_WithM**, **Motor_WithoutM**, **Motor_OnlyM**, **Pump**, **Fan**, **Compressor**, **Feeder**, or **Rotary**. Refer to Figure 248.
Runs1	Boolean	False	Running first position, motor running with low speed or forward. Refer to Figure 248.
Runs2	Boolean	False	Running second position, motor running with high speed or backward. Refer to Figure 248.
StarterType	Integer	0	MotorInsumBasic StarterType. Refer to Figure 248.

MotorInsumExtended

This is a symbol describing the functionality of MotorInsumExtended objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

The **CfgType** property controls the appearance of the symbol. Figure 249 shows symbols used for indication of MotorInsumExtended.

Figure 249. MotorInsumExtended Symbols

Table 242. MotorInsumExtended Properties

Name	Type	Default	Description
CfgDirection	Direction	Right	Controls the direction of the symbol. Used by MotorInsum, Pump, Fan and Compressor. This can be **Up**, **Down**, **Left**, or **Right**.
CfgFrameWidth	Integer	8	Frame width of the symbol.
CfgType	SymbolTypeEnum	MotorInsum	Type of symbol. The types can be **MotorInsum**, **Motor_WithM**, **Motor_WithoutM**, **Motor_OnlyM**, **Pump**, **Fan**, **Compressor**, **Feeder**, or **Rotary**.
Delta	Boolean	False	MotorInsumExtended Delta. Refer to Figure 249.
Runs1	Boolean	False	Running first position, motor running with low speed or forward. Refer to Figure 249.
Runs2	Boolean	False	Running second position, motor running with high speed or backward. Refer to Figure 249.
Star	Boolean	False	MotorInsumExtended Star. Refer to Figure 249.
StarterType	Integer	0	MotorInsumExtended StarterType. Refer to Figure 249.

MotorUni

This is a symbol describing the functionality of MotorUni objects with one feedback. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

The **CfgType** property controls the appearance of the symbol. Figure 250 shows symbols used for indication of MotorUni.

Figure 250. MotorUni Symbols

Table 243. MotorUni Properties

Name	Type	Default	Description
CfgDirection	Direction	Right	Controls the direction of the symbol. Used by MotorUni, Pump, Fan and Compressor. The direction can be **Up**, **Down**, **Left**, or **Right**.
CfgFrameWidth	Integer	8	Frame width of the symbol.
CfgRunningImage	Image	-	Own image showed for running motor. Used when **CfgType**=MotorOwn.

Table 243. MotorUni Properties (Continued)

Name	Type	Default	Description
CfgStopImage	Image	-	Own image showed for stopped motor. Used when **CfgType**=MotorOwn.
CfgType	SymbolTypeEnum (Uni,)	Uni	Type of symbol. The types can be **MotorUni**, **Motor_WithM**, **Motor_WithoutM**, **Motor_OnlyM**, **Pump**, **Fan**, **Compressor**, **Feeder**, **Rotary**, or **MotorOwn**. Refer to Figure 250.
Running	Boolean	False	Specifies whether the motor is running or not. Refer to Figure 250.

MotorValve

This is a symbol describing the functionality of MotorValve objects with two feedbacks. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

The **CfgType** property controls the appearance of the symbol. Table 244 shows symbols used for indication of MotorValve.

Table 244. Symbols Showing MotorValve
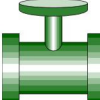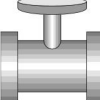
CfgType Valve	CfgType ValveMotor	CfgType ValveOwn	
			Open=False Close=False
			Open=True Close=False
			Open=False Close=True
			Open=True Close=True

Table 245. MotorValve Properties

Name	Type	Default	Description
CfgCloseImage	Image	-	Own image showed for closed valve. Used when **CfgType**=ValveOwn.
CfgFrameWidth	Integer	2	Frame width of the symbol.

Table 245. MotorValve Properties (Continued)

Name	Type	Default	Description
CfgIntermediateImage	Image	-	Own image showed for intermediate positioned valve. Used when **CfgType**=ValveOwn.
CfgOpenImage	Image	-	Own image showed for opened valve. Used when **CfgType**=ValveOwn.
CfgOrientation	Orientation	Horizontal	Controls the direction of the symbol. Used by Valve and ValveMotor. This can be **Vertical** or **Horizontal**.
CfgType	SymbolTypeEnum	Arrow	Type of symbol. The types can be **Valve**, **ValveMotor**, or **ValveOwn**. Refer to Table 244.
Close	Boolean	False	Valve closed. Refer to Table 244.
Open	Boolean	False	Valve opened. Refer to Table 244.

Pid

This is a symbol describing the functionality of Pid objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

The **CfgType** property controls the appearance of the symbol. Table 246 shows symbols used for indication of Pid.

Table 246. Symbols Showing Pid

CfgType Arrow	CfgType Arrow Advanced	CfgType ArrowFuzzy	CfgType ArrowSimple	CfgType ISA	CfgType ValveMotor	
						ISAName=Name Value=25
						ISAName=XXX Value=50

Table 247. Pid Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
CfgOrientation	Orientation	Horizontal	Controls the direction of the symbol. Only used by ValveMotor. This can be **Vertical** or **Horizontal**.

Table 247. Pid Properties (Continued)

Name	Type	Default	Description
CfgType	SymbolTypeEnum ( )	Arrow	Type of symbol. The types can be **Arrow**, **ArrowAdvanced**, **ArrowFuzzy**, **ArrowSimple**, **ISA**, or **ValveMotor**. Refer to Table 246.
ISAName	String	"Name"	Text presented in symbol. Used by ISA.
RangeMax	Real	100.0	Maximum range for analog presentation. Used by ValveMotor.
RangeMin	Real	0.0	Minimum range for analog presentation. Used by ValveMotor.
Value	Real	50.0	Actual value. Used by ValveMotor.

PidCascadeLoop

This is a static symbol describing the functionality of PidCascadeLoop objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 248 shows symbols used for indication of PidCascadeLoop.

Table 248. Symbols Showing PidCascadeLoop
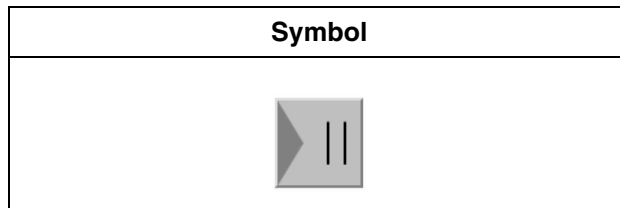
Symbol

Table 249. PidCascadeLoop Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

PidForwardLoop

This is a static symbol describing the functionality of PidForwardLoop objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 250 shows symbols used for indication of PidForwardLoop.

Table 250. Symbols Showing PidForwardLoop
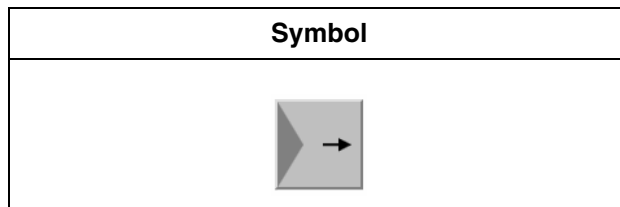
Symbol

Table 251. PidForwardLoop Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

PidMidrangeLoop

This is a static symbol describing the functionality of PidMidrangeLoop objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 252 shows symbols used for indication of PidMidrangeLoop.

Table 252. Symbols Showing PidMidrangeLoop
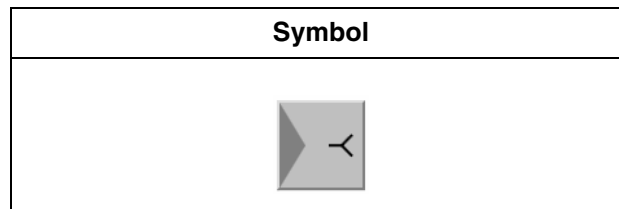
Symbol

Table 253. PidMidrangeLoop Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

PidOverrideLoop

This is a static symbol describing the functionality of PidOverrideLoop objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 254 shows symbols used for indication of PidOverrideLoop.

Table 254. Symbols Showing PidOverrideLoop
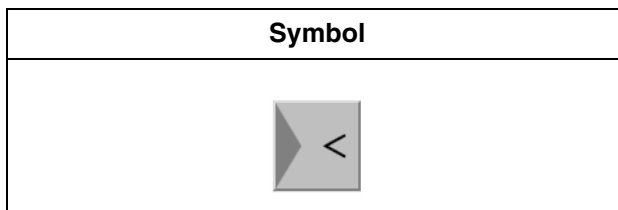
Symbol

Table 255. PidOverrideLoop Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

PidSingleLoop

This is a static symbol describing the functionality of PidSingleLoop objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 256 shows symbols used for indication of PidSingleLoop.

Table 256. Symbols Showing PidSingleLoop
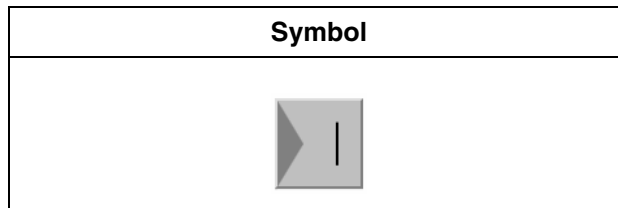
Symbol

Table 257. PidSingleLoop Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

PulseWidth

This is a static symbol describing the functionality of PulseWidth objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 258 shows symbols used for indication of PulseWidth.

Table 258. Symbols Showing PulseWidth
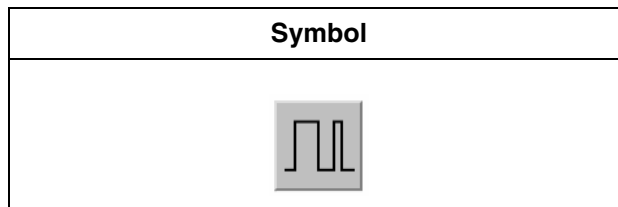
Symbol

Table 259. PulseWidth Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

SDLevel

This is a symbol describing the functionality of SDLevel objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

The **CfgType** property controls the appearance of the symbol. Table 260 shows symbols used for indication of SDLevel.

Table 260. Symbols Showing SDLevel

Symbol	Symbol Information
	ShowLevelNo=false LatchedLevel=false
	LevelNo=1 ShowLevelNo=true LatchedLevel=false
	ShowLevelNo=false LatchedLevel=true

Table 261. SDLevel Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgType	SymbolTypeEnum	Level	Type of symbol. The types can be **Level** or **Own**. Refer to Table 260.
CfgImage	Image	-	Own image showed for SDLevel. Used when **CfgType**=Own.
CfgFrameWidth	Integer	2	Frame width of the symbol.
LatchedLevel	Boolean	False	Output activated. Refer to Table 260.
LevelNo	Integer	0	Specifies the hierarchy level. Refer to Table 260.
ShowLevelNo	Boolean	False	Displays the level number in the symbol. Refer to Table 260.

Select

This is a symbol describing the functionality of Select objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 262 shows symbols used for indication of Select.

Table 262. Symbols Showing Select

Symbol	Symbol Information
	In2Selected=false
	In2Selected=true

Table 263. Select Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
In2Selected	Boolean	False	Box 2 selected. Refer to Table 262.

Select4

This is a symbol describing the functionality of Select4 objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 264 shows symbols used for indication of Select4.

Table 264. Symbols Showing Select4

Symbol	Symbol Information
	Selected (1-4) controls the color of the 4 boxes Selected=2

Table 265. Select4 Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
Selected	Integer	0	Specifies the box selected. Refer to Table 264.

SFC2D

This is a static symbol describing the functionality of SFC2D objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 266 shows symbols used for indication of SFC2D.

Table 266. Symbols Showing SFC2D

Symbol

Table 267. SFC2D Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

SignalBool

This is a symbol describing the functionality of SignalBool objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

The **CfgType** property controls the appearance of the symbol. Table 268 shows symbols used for indication of SignalBool.

Table 268. Symbols Showing SignalBool

CfgType=Circle	CfgType=Square	CfgType=Switch	CfgType=Own	
				Value = false
				Value = true

Table 269. SignalBool Properties

Name	Type	Default	Description
CfgDirection	Direction	Up	Controls the position of the Circle and the Square.
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**. Only used for Switch.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**. Only used for Switch.
CfgFrameWidth	Integer	2	Frame width of the symbol. Only used for Switch.
CfgOffImage	Image	-	Own image showed for off signal. Used when **CfgType**=Own.

Table 269. SignalBool Properties (Continued)

Name	Type	Default	Description
CfgOnImage	Image	-	Own image showed for on signal. Used when **CfgType**=Own.
CfgType	SymbolTypeEnum (Circle, Square, Switch, Own)	Circle	Type of symbol. The types can be **Circle**, **Square**, **Switch**, or **Own**. Refer to Table 268.
Value	Boolean	False	Boolean signal value. Refer to Table 268.

SignalReal

This is a symbol describing the functionality of SignalReal objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

The **CfgType** property controls the appearance of the symbol. Table 270 shows symbols used for indication of SignalReal.

Table 270. Symbols Showing SignalReal

CfgType Actuator	CfgType ValveMotor	CfgType ValveMotor	
			CfgDirection=Up CfgOrientation=Horizontal Value=50.0
			CfgDirection=Down CfgOrientation=Vertical Value=50.0

Table 270. Symbols Showing SignalReal (Continued)

CfgType Actuator	CfgType ValveMotor	CfgType ValveMotor	
			CfgDirection=Left
			CfgDirection=Right

Table 271. SignalReal Properties

Name	Type	Default	Description
CfgDirection	Direction	Down	Controls the direction of the symbol. Only used by Actuator. This can be **Up**, **Down**, **Left**, or **Right**.
CfgFrameWidth	Integer	4	Frame width of the symbol. Used by ValveMotor.
CfgImage	Image	-	Own image showed for real signal. Used when **CfgType**=Own.
CfgOrientation	Orientation	Horizontal	Controls the direction of the symbol. Used by ValveMotor. This can be **Vertical** or **Horizontal**.
CfgText	String	"AI"	Text displayed inside circle. Used by Actuator.

Table 271. SignalReal Properties (Continued)

Name	Type	Default	Description
CfgType	SymbolTypeEnum	Actuator	Type of symbol. The types can be **Actuator**, **ValveMotor**, or **Own**. Refer to Table 270.
RangeMax	Real	100.0	Maximum range for analog presentation. Only used by ValveMotor.
RangeMin	Real	0.0	Minimum range for analog presentation. Only used by ValveMotor.
Value	Real	50.0	Actual value. Only used by ValveMotor.

SignalSupervision

This is a static symbol describing the functionality of SignalSupervision objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 272 shows symbols used for indication of SignalSupervision.

Table 272. Symbols Showing SignalSupervision

Symbol

Table 273. SignalSupervision Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

SystemDiagnostics

This is a symbol describing the functionality of SystemDiagnostics objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 274 shows symbols used for indication of SystemDiagnostics.

Table 274. Symbols Showing SystemDiagnostics

Symbol	Symbol Information
50 %	TotalSystemLoadPerCent=50

Table 275. SystemDiagnostics Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.

Table 275. SystemDiagnostics Properties

CfgFrameWidth	Integer	2	Frame width of the symbol.
TotalSystemLoadPerCent	Integer	0	System Diagnostics Total System Load PerCent. Refer to Table 274.

ThreePos

This is a symbol describing the functionality of ThreePos objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 276 shows symbols used for indication of ThreePos.

Table 276. Symbols Showing ThreePos

Symbol	Symbol Information
	Increase=true
	Decrease=true
	MaxPosReached =true
	MinPosReached =true

Table 277. ThreePos Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.
Decrease	Boolean	False	Indicates decreasing output signal. Refer to Table 276.
Increase	Boolean	False	Indicates increasing output signal. Refer to Table 276.
MaxPosReached	Boolean	False	Actuator reached the maximum value. Refer to Table 276.
MinPosReached	Boolean	False	Actuator reached the minimum value. Refer to Table 276.

Uni

This is a combined motor/valve symbol describing the functionality of Uni objects with one feedback. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

The **CfgType** property controls the appearance of the symbol. Table 278 shows symbols used for indication of Uni.

Table 278. Uni Symbols

Symbol										
Uni	Uni	MotorUni	WithM	WithoutM	OnlyM	Pump	Fan	Compressor	Feeder	Rotary
Activate=false	⬤	▷	M	⬡	M	▷	◁	◁	✳	✳
Activate=true	⬤	▶	M	◆	M	▶	▶	◀	✳	✳
	Own	Valve	Hand	Actuator	Motor	Diaphragm				
Activate=false	🏭	⋈	⋈	⋈	Ⓜ⋈	⋈				
Activate=true	🏭	⋈	⋈	⋈	Ⓜ⋈	⋈				

Table 279. Uni Properties

Name	Type	Default	Description
Activate	Boolean	False	Motor/Valve activated. Refer to Table 278.
CfgActiveImage	Image	-	Own image showed for active motor/valve. Used when **CfgType**=Own.
CfgDeactiveImage	Image	-	Own image showed for deactive motor/valve. Used when **CfgType**=Own.
CfgDirection	Direction	Right	Controls the direction of the motor symbol. Used by MotorUni, Pump, Fan and Compressor. This can be **Up**, **Down**, **Left**, or **Right**.
CfgFrameWidth	Integer	8	Frame width of the symbol.
CfgOrientation	Orientation	Horizontal	Controls the direction of the valve symbols. This can be **Vertical** or **Horizontal**.
CfgType	SymbolTypeEnum	Uni	Type of symbol. The types can be **Uni**, **MotorUni**, **Motor_WithM**, **Motor_WithoutM**, **Motor_OnlyM**, **Pump**, **Fan**, **Compressor**, **Feeder**, **Rotary**, **Own**, **Valve**, **ValveHand**, **ValveActuator**, **ValveMotor**, or **ValveDiaphragm**. Refer to Table 278.

ValveUni

This is a symbol describing the functionality of ValveUni objects with one feedback. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element.

The **CfgType** property controls the appearance of the symbol. Table 280 shows symbols used for indication of ValveUni.

Table 280. ValveUni Symbols

Table 281. ValveUni Properties

Name	Type	Default	Description
CfgFrameWidth	Integer	4	Frame width of the symbol.
CfgOpenImage	Image	-	Own image showed for opened valve. Used when **CfgType**=Own.
CfgCloseImage	Image	-	Own image showed for closed valve. Used when **CfgType**=Own.
CfgOrientation	Orientation	Horizontal	Controls the direction of the symbol (not used when **CfgType**=ValveOwn). This can be **Vertical** or **Horizontal**. Refer to Table 280.
CfgType	SymbolTypeEnum	Valve	Type of symbol. The types can be **Valve**, **ValveHand**, **ValveAcuator**, **ValveMotor**, **ValveDiaphragm**, or **ValveOwn**. Refer to Table 280.
Open	Boolean	False	Specifies whether the valve is opened or not. Refer to Table 280.

Vote

This is a static symbol describing the functionality of Vote objects. The symbol can be used together with Display Element Icon or Display Element Reduced Icon core element. Table 282 shows symbols used for indication of Vote.

Table 282. Symbols Showing Vote
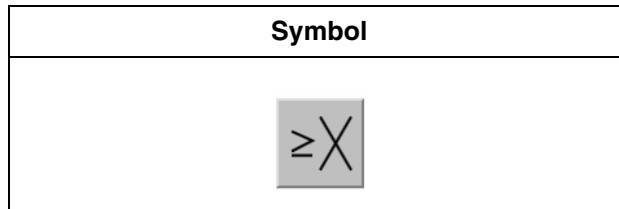
Symbol

Table 283. Vote Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

Base Generic Elements

The base generic elements are generic elements used as building blocks in other generic element. These base symbols are used in Motor symbols, Valve symbols, and as background symbols.

For information on the running states, refer to the symbols MotorUni and MotorBi. For information on the valve states, refer to the symbol ValveUni.

The following are the base generic elements:

• BaseBi

• BaseMotorBi

• BaseMotorCompressor

- BaseMotorFan

- BaseMotorFeeder

- BaseMotorM

- BaseMotorPump

- BaseMotorRotary

- BaseMotorUni

- BaseRound

- BaseSquare

- BaseUni

- BaseValve

- BaseValveActuator

BaseBi

Table 284 shows symbols used for indication of BaseBi.

Table 284. Symbols Showing BaseBi

Symbol

Table 285. BaseBi Properties

Name	Type	Default	Description
Running1	Boolean	False	Running first position, motor running with low speed or forward.
Running2	Boolean	False	Running second position, motor running with high speed or backward.

BaseMotorBi

Table 286 shows symbols used for indication of BaseMotorBi.

Table 286. Symbols Showing BaseMotorBi

Symbol

Table 287. BaseMotorBi Properties

Name	Type	Default	Description
CfgDirection	Direction	Right	Direction of the symbol. This can be **Up**, **Down**, **Left**, or **Right**.
Running1	Boolean	False	Running first position, motor running with low speed or forward.
Running2	Boolean	False	Running second position, motor running with high speed or backward.

BaseMotorCompressor

Table 288 shows symbols used for indication of BaseMotorCompressor.

Table 288. Symbols Showing BaseMotorCompressor

Symbol

Table 289. BaseMotorCompressor Properties

Name	Type	Default	Description
BiDirectional	Boolean	False	Bidirectional or not.
CfgDirection	Direction	Right	Direction of the symbol. This can be **Up**, **Down**, **Left**, or **Right**.
Running1	Boolean	False	Running first position, motor running with low speed or forward.
Running2	Boolean	False	Running second position, motor running with high speed or backward.

BaseMotorFan

Table 290 shows symbols used for indication of BaseMotorFan.

Table 290. Symbols Showing BaseMotorFan

Symbol

Table 291. BaseMotorFan Properties

Name	Type	Default	Description
BiDirectional	Boolean	False	Bidirectional or not.
CfgDirection	Direction	Right	Direction of the symbol. This can be **Up**, **Down**, **Left**, or **Right**.
Running1	Boolean	False	Running first position, motor running with low speed or forward.
Running2	Boolean	False	Running second position, motor running with high speed or backward.

BaseMotorFeeder

Table 292 shows symbols used for indication of BaseMotorFeeder.

Table 292. Symbols Showing BaseMotorFeeder
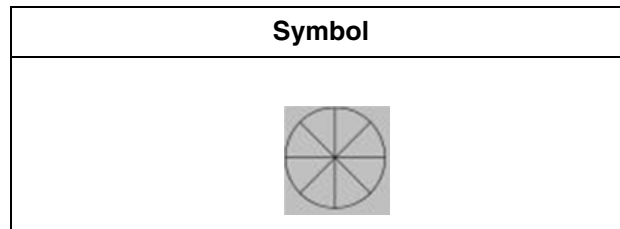
Symbol

Table 293. BaseMotorFeeder Properties

Name	Type	Default	Description
Running	Boolean	False	Specifies whether the motor is running or not.

BaseMotorM

Table 294 shows symbols used for indication of BaseMotorM.

Table 294. Symbols Showing BaseMotorM
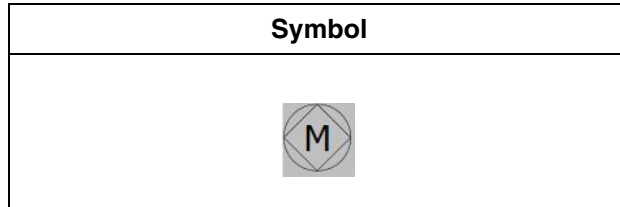
Symbol

Table 295. BaseMotorM Properties

Name	Type	Default	Description
CfgEnableM	Boolean	True	Enables "M".
CfgEnableRectangle	Boolean	True	Enables rectangle.
CfgFrameWidth	Integer	8	Frame width of the symbol.
Running	Boolean	False	Specifies whether the motor is running or not.

BaseMotorPump

Table 296 shows symbols used for indication of BaseMotorPump.

Table 296. Symbols Showing BaseMotorPump
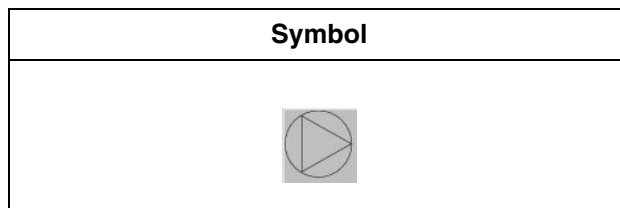
Symbol

Table 297. BaseMotorPump Properties

Name	Type	Default	Description
BiDirectional	Boolean	False	Bidirectional or not.
CfgDirection	Direction	Right	Direction of the symbol. This can be **Up**, **Down**, **Left**, or **Right**.
Running1	Boolean	False	Running first position, motor running with low speed or forward.
Running2	Boolean	False	Running second position, motor running with high speed or backward.

BaseMotorRotary

Table 298 shows symbols used for indication of BaseMotorRotary.

Table 298. Symbols Showing BaseMotorRotary

Symbol

Table 299. BaseMotorRotary Properties

Name	Type	Default	Description
Running	Boolean	False	Specifies whether the motor is running or not.

BaseMotorUni

Table 300 shows symbols used for indication of BaseMotorUni.

Table 300. Symbols Showing BaseMotorUni

Symbol

Table 301. BaseMotorUni Properties

Name	Type	Default	Description
BiDirectional	Boolean	False	Specifies if the motor is Bidirectional or not.
CfgDirection	Direction	Right	Direction of the symbol. This can be **Up**, **Down**, **Left**, or **Right**.
Running1	Boolean	False	Running first position, motor running with low speed or forward.
Running2	Boolean	False	Running second position, motor running with high speed or backward.

BaseRound

Table 302 shows symbols used for indication of BaseRound.

Table 302. Symbols Showing BaseRound
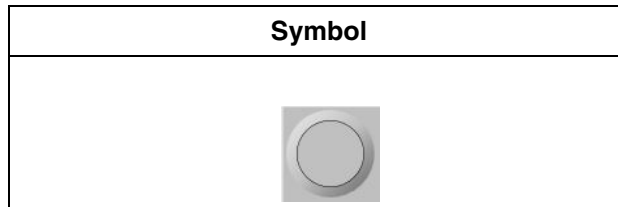
Symbol

Table 303. BaseRound Properties

Name	Type	Default	Description
CfgFrameWidth	Integer	8	Frame width of the symbol.

BaseSquare

Table 304 shows symbols used for indication of BaseSquare.

Table 304. Symbols Showing BaseSquare
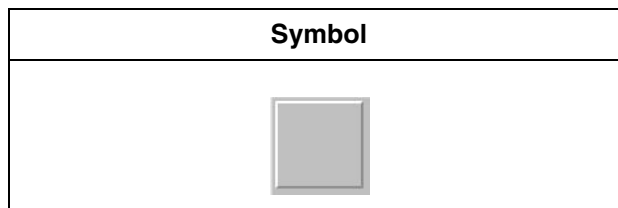
Symbol

Table 305. BaseSquare Properties

Name	Type	Default	Description
CfgFillStyle	FillStyleEnum (Solid, Transparent)	Solid	Fill style for the symbol. This can be **Solid** or **Transparent**.
CfgFrame3DEffect	Frame3DEffect (Raised, Flat, Sunken)	Raised	3D effect for the frame. This can be **Raised**, **Flat** or **Sunken**.
CfgFrameWidth	Integer	2	Frame width of the symbol.

BaseUni

Table 306 shows symbols used for indication of BaseUni.

Table 306. Symbols Showing BaseUni

Symbol

Table 307. BaseUni Properties

Name	Type	Default	Description
Running	Boolean	False	Specifies if the motor is running or not.

BaseValve

Table 308 shows symbols used for indication of BaseValve.

Table 308. Symbols Showing BaseValve

Symbol

Table 309. BaseValve Properties

Name	Type	Default	Description
CfgFrameWidth	Integer	2	Frame width of the valve.
CfgOrientation	Orientation	Horizontal	Direction of the valve symbol. This can be **Vertical** or **Horizontal**.
FillColor	Brush	Transparent()	Fill color of valve symbol.

BaseValveActuator

Table 310 shows symbols used for indication of BaseValveActuator.

Table 310. Symbols Showing BaseValveActuator

Symbol

Table 311. BaseValveActuator Properties

Name	Type	Default	Description
CfgOrientation	Orientation	Horizontal	Direction of the actuator symbol. This can be **Vertical** or **Horizontal**.
CfgType	SymbolTypeEnum	Hand	Type of actuator. The types can be **Hand**, **Actuator**, **Motor**, or **Diaphragm**.

Appendix C User-Defined Aspect Categories

User-defined aspect categories can be created in the workplace. The user can create a template aspect for the user-defined aspect category.

If the user requires a default configuration, for example, size of the graphic element or background of the graphic element, a template should be added to the aspect category. Otherwise, the template existing on the aspect type will be used while creating an instance of the aspect category.

Execute the following steps to create an aspect category.

1. In the **Aspect System Structure**, click **Process Graphics 2**. This includes **Generic Element PG2**, **Graphic Display PG2**, **Graphic Element PG2**, and **Solution Library PG2** aspect types.
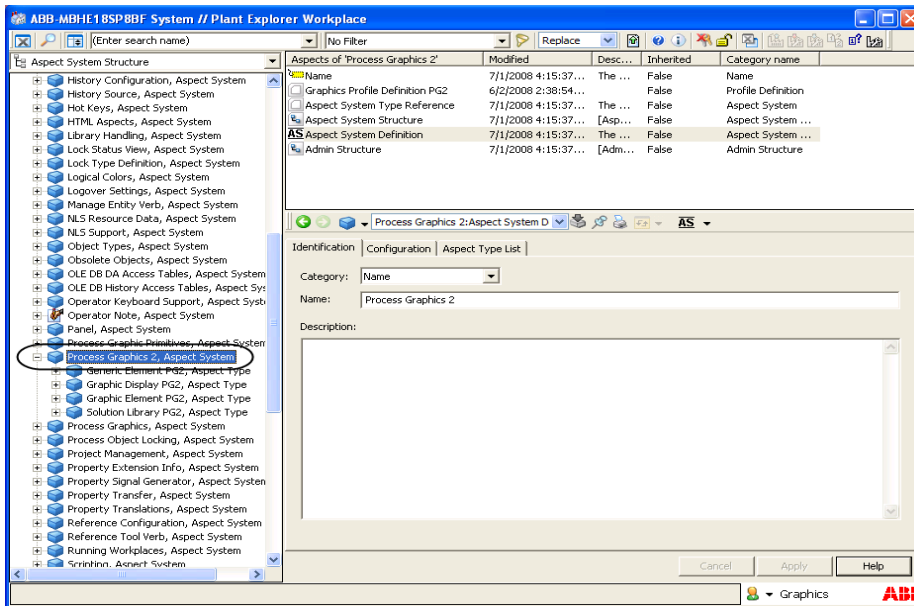
Figure 251. Aspect System Structure

2. Right-click on one of the aspect types. For example, right-click on **Graphic Display PG2** and select **Aspect Type Definition** from the context menu. The **Aspect Type Definition** dialog appears as shown in Figure 253.
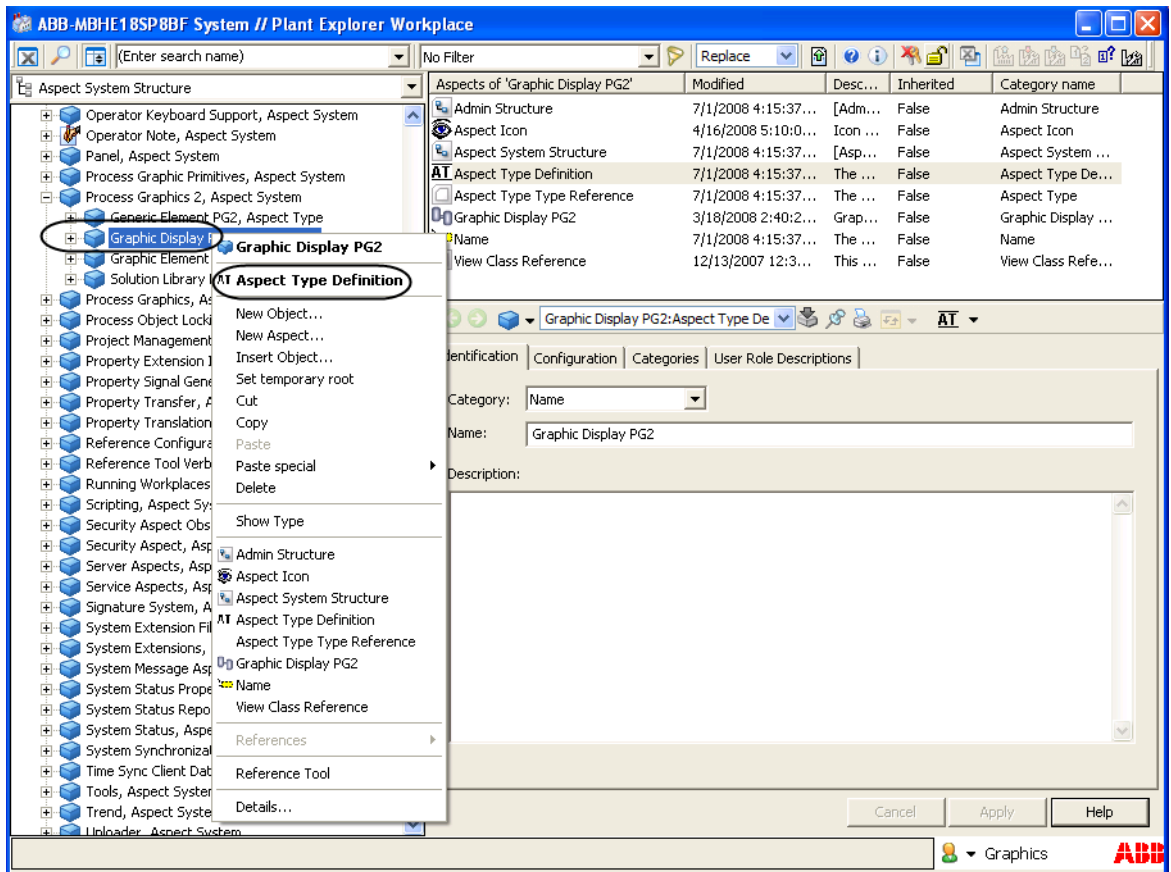
Figure 252. Context menu of the aspect type
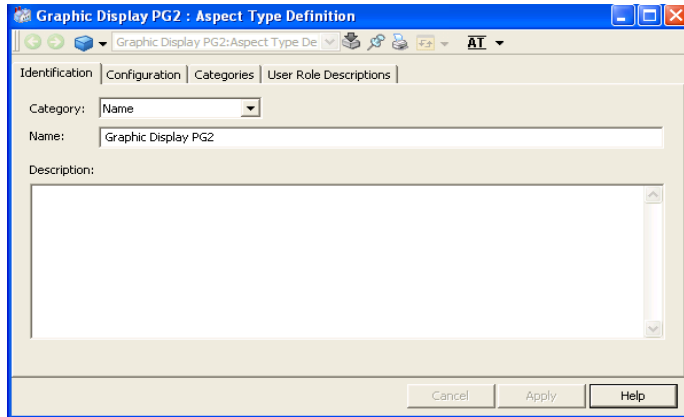
Figure 253. Aspect Type Definition

3. Click **Categories** tab and then click **Add**. The **New Category** dialog appears as shown in Figure 255.
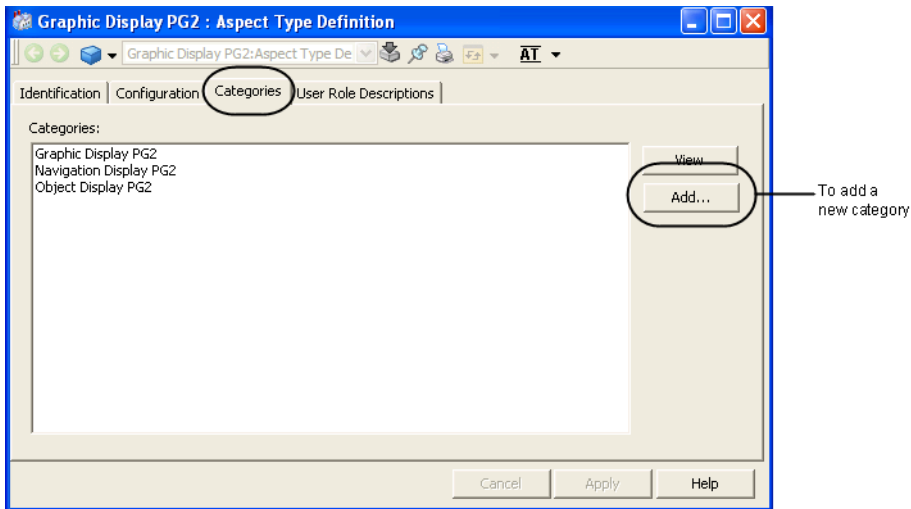
Figure 254. Categories tab
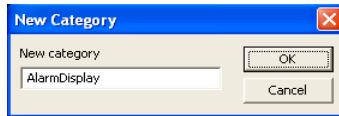
Figure 255. New Category

4.   Enter **Alarm Display** as the name for the category and click **OK**.

5.   Click **Apply** in **Aspect Type Definition** dialog to save the changes.

6.   Close the dialog. The aspect type **Graphic Display PG2** contains the newly created aspect category **Alarm Display**.
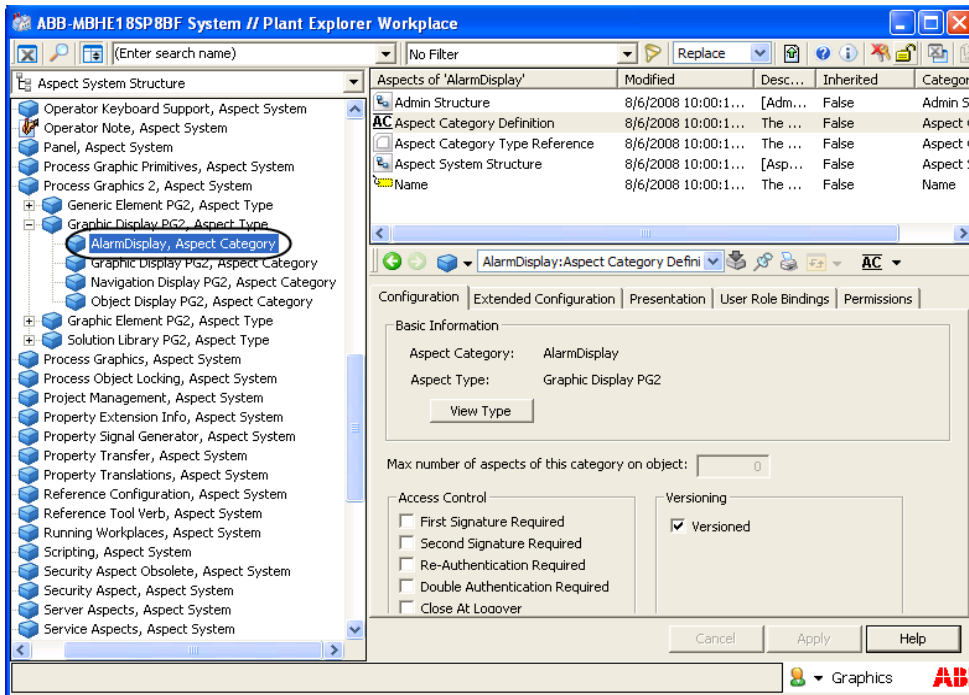
Figure 256. Aspect type with the new aspect category

7.   Right-click on **Alarm Display**, and select **New Aspect** from the context menu. The **New Aspect** dialog appears as shown in Figure 258.
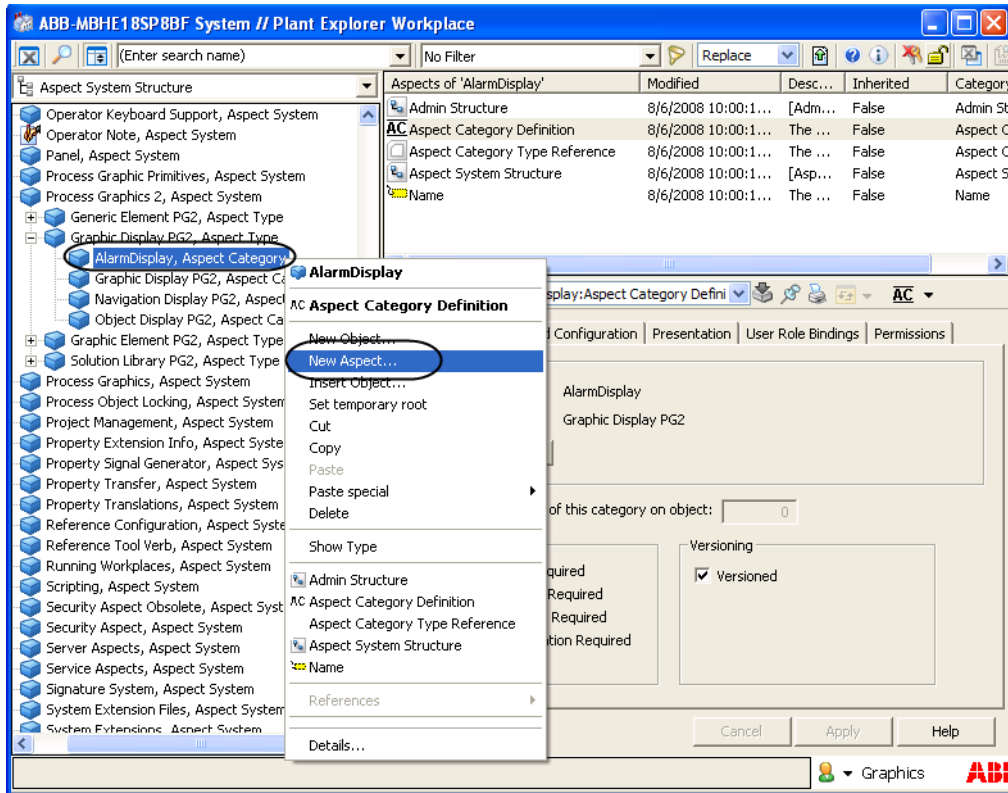
Figure 257. Context menu of the new aspect category
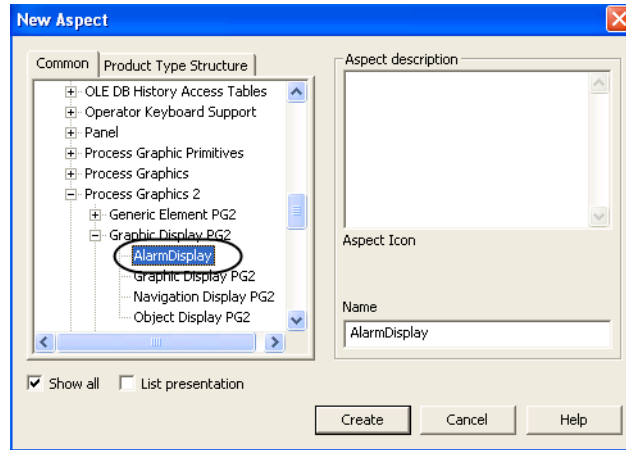
Figure 258. New Aspect

8. In the aspect list, select **Alarm Display** from **Process Graphics 2 > Graphic Display PG2**. Enter a name for the aspect and click **Create**. Refer Figure 258.

The templates from aspect type (**Graphic Display PG2** in this example) is copied to the new aspect category **Alarm Display**.

9. Right-click the newly created aspect and select **Edit** from the context menu. This opens the Graphics Builder.
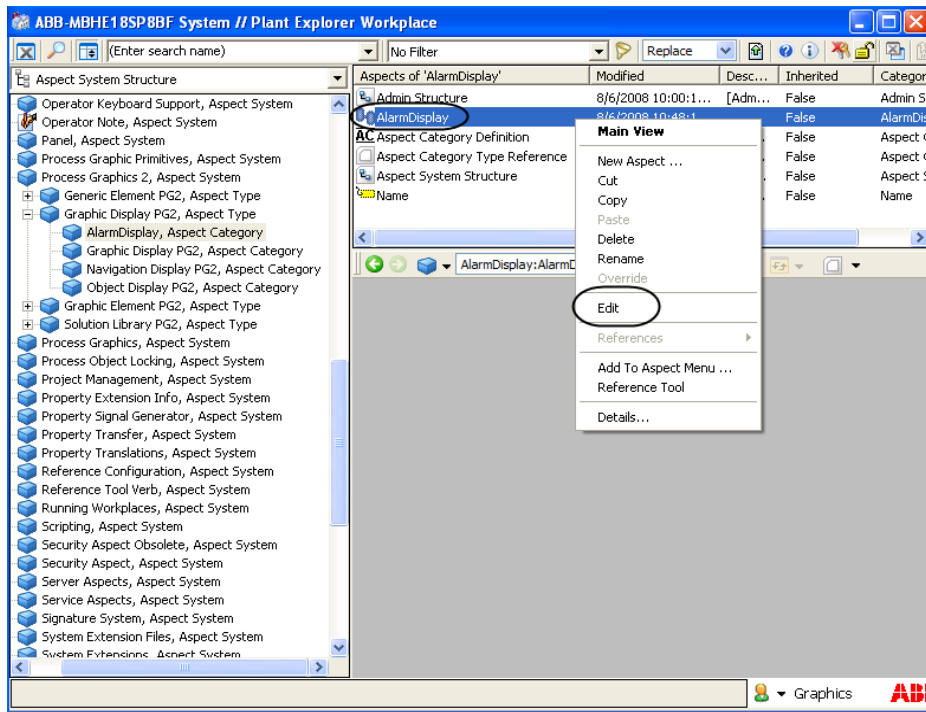
Figure 259. Context menu of the new aspect

10. In the Graphics Builder, configure the template by changing background color or by adding graphic primitives.

> The user can add references to global properties (a kind of data-entity reference).

11. Select **File > Save** to save the configuration. Close the Graphics Builder.

12. Right-click the newly created aspect and select **Details** from the context menu as shown in Figure 259. The **Details** dialog appears.
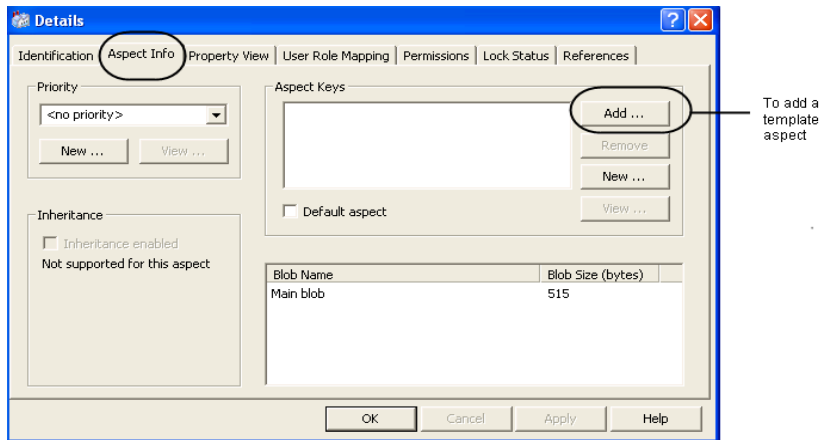
Figure 260. Aspect Details

13. To add the template aspect, click **Aspect Info** tab and click **Add**.
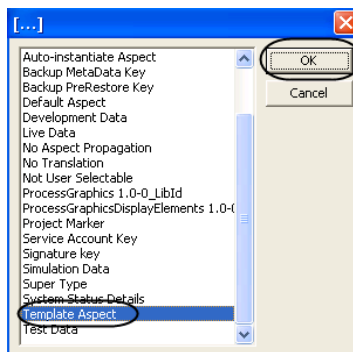
Figure 261. Selecting Template Aspect

14. Select **Template Aspect** and click **OK**.

15. Click **Apply** and then click **OK** in **Details** dialog to save the changes.

> The user should save all the changes as a **.afw** file as a backup.

Configuring user roles and permission

The user settings and permission should be configured after creating a user-defined aspect category. User setting specifies the user roles to perform read and modify actions, and permission specifies the permission to read and modify instances of the aspect.

Configuration of user roles and permission is done in the **Aspect Category Definition** aspect of the newly created aspect category.

Right-click on the newly created aspect category and select **Aspect Category Definition** from the context menu.
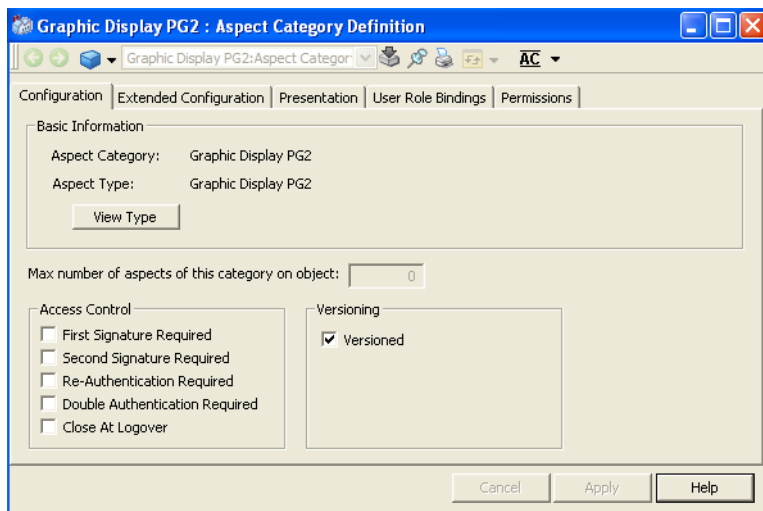
Figure 262. Aspect Category Definition

Click **User Role Bindings** tab to set the user roles.
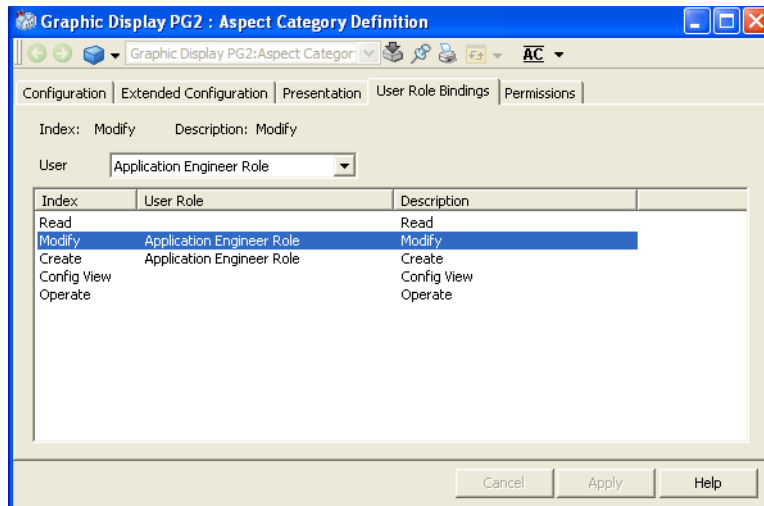
Figure 263. User Role Bindings

Execute the following steps to set a user role.

1.   Select an index for which the user role should be set (for example, Read).

2.   Select the user role in **User**. All users belonging to the selected user role will
      have access to the selected index.

Do not select a user role if all users are allowed to perform the selected operation.

Click **Permissions** tab to set the user permission.
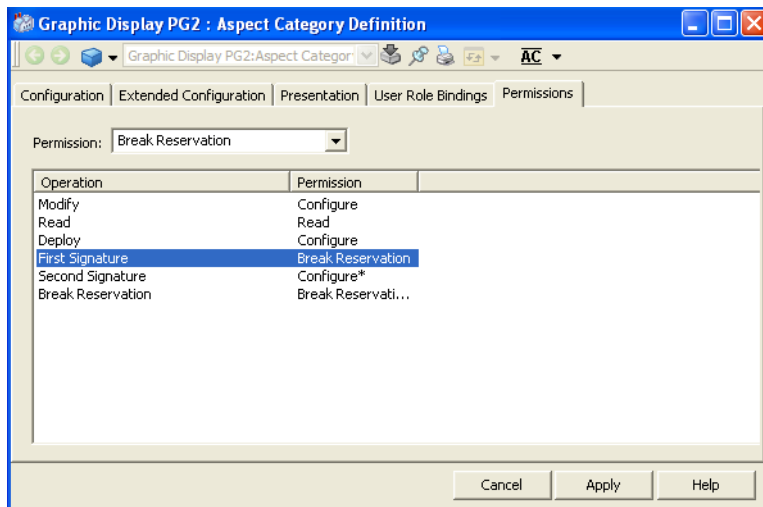
Figure 264. Permissions

Execute the following steps to set a permission.

1.   Select an operation for which the permission should be set (for example, Modify).

2.   Select the required permission in **Permission**.

Appendix D Generic Elements and Object Types

Following is an example that helps the user to create an object type. This also explains how to create an instance of object aware graphic aspect on the object type and how to create an instance of the object type.

1. Create an object *Tank* of **Object Type** in the **Object Type Structure**.
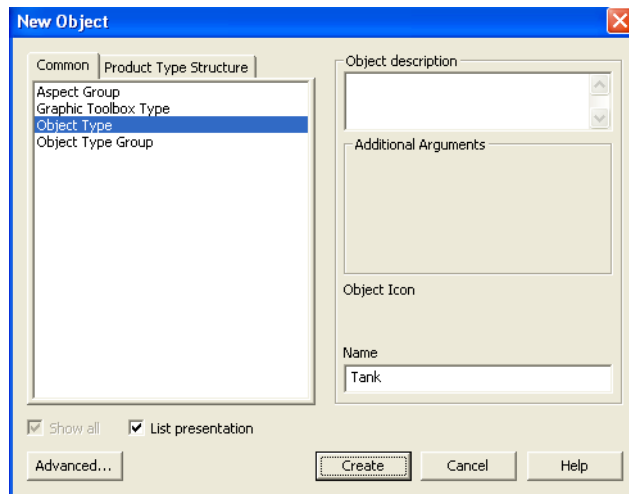
Figure 265. New Object

2. Create a graphic aspect of **Graphic Element PG2** in the *Tank* object type. For more information on creating graphic aspects, refer to Creating a New Graphic Aspect on page 31.

3. Right-click on the aspect and select **Details** from the context menu. Set **Inheritance enabled** for the aspect.
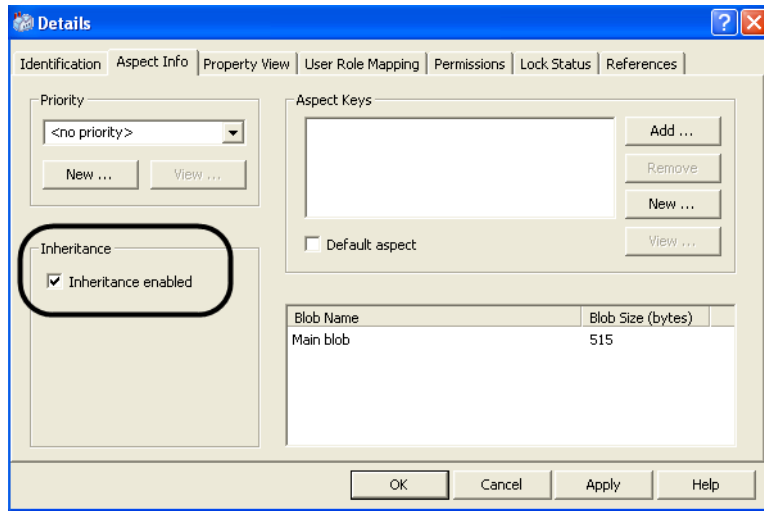
Figure 266. Enabling Inheritance

4. Edit the graphic aspect using the Graphics Builder. For more information on editing a graphic aspect, refer to Launching the Graphics Builder on page 32.

5. Go to the **Functional Structure**.

6. Create a new object *Tank1* for the category *Tank*. The object *Tank1* is an instance of object *Tank*.

   All graphic aspects having inheritance enabled in *Tank* object is available in *Tank1* object.
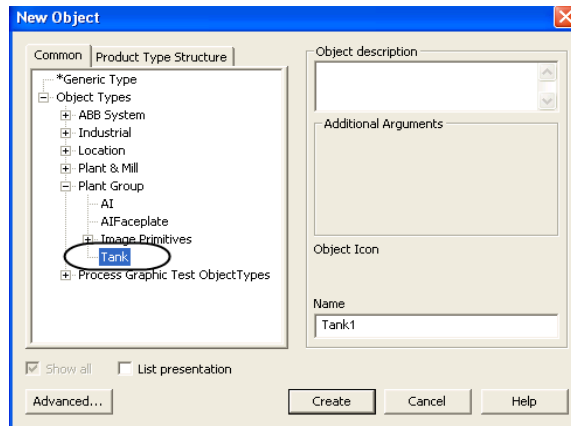
Figure 267. New Object Instance

Creating Generic Elements

This section helps the user to create generic elements directly in the **Graphics Structure** or in a library. Create objects from the following object types for creating toolboxes and generic elements.

- **Graphics Toolbox Typ**e - Instances of this object type represents toolboxes in a library.

- **Graphics Toolbox Item Type** - Instances of this object type represents generic elements in a library.

- **Graphics Toolbox** - Instances of this object type represents toolboxes in the **Graphics Structure**, irrespective of whether the elements are library elements. They are used in the **Graphics Structure.**

- **Graphics Toolbox Item** - Instances of this object type represents generic elements created directly in the **Graphics Structure** without having an element in a library.

Creating Toolbox and Generic Elements in Graphics Structure

The object layout in **Graphics Structure > Graphics Tools** determines the toolboxes which should be displayed in the Graphics Builder. It also contains the generic elements that should be displayed in each toolbox.
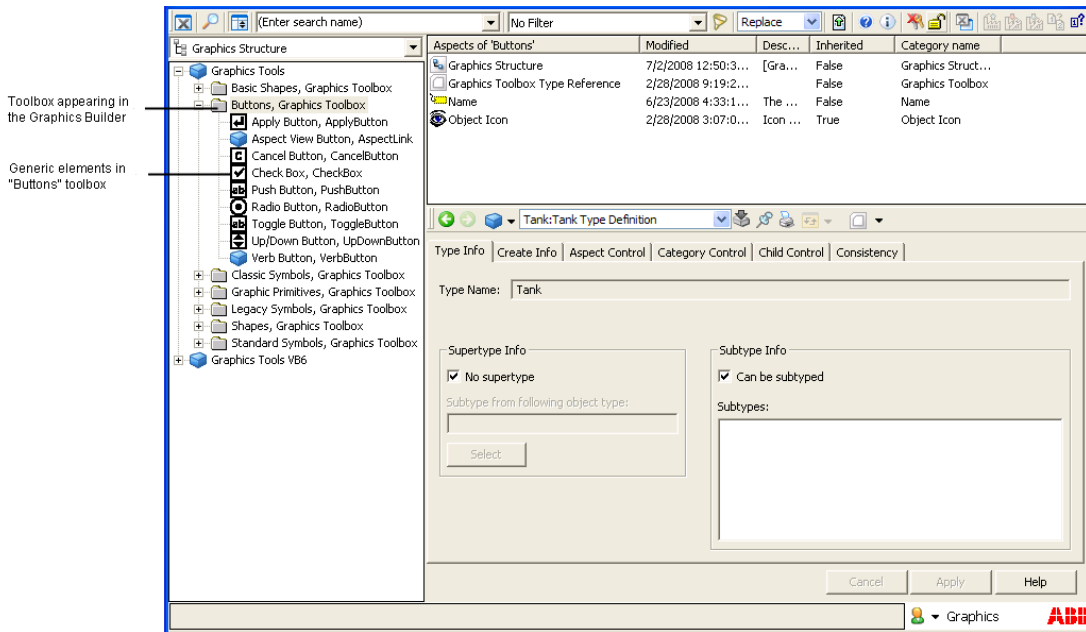
Figure 268. Object Layout of Graphics Structure

Creating a toolbox

Execute the following steps to create a toolbox in the **Graphics Structure.**

1. Create an object of **Graphics Toolbox** type in **Graphic Tools.** For example, *Basic Shapes*.
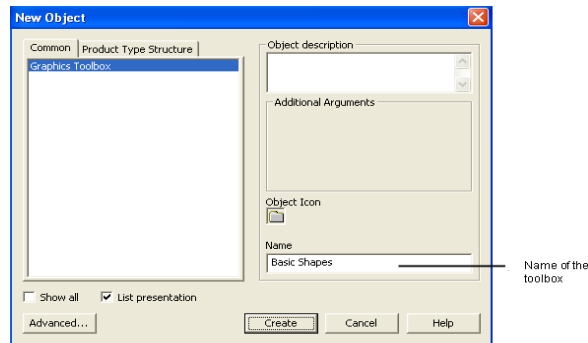
Figure 269. New Toolbox object

Creating generic elements

Execute the following steps to create generic elements in the **Graphics Structure**.

1. Create the *Basic Shapes* toolbox as specified in Creating a toolbox.

2. Create another object *Circle* of **Graphics Toolbox Item** type within the *Basic Shapes* object as mentioned in step 1.
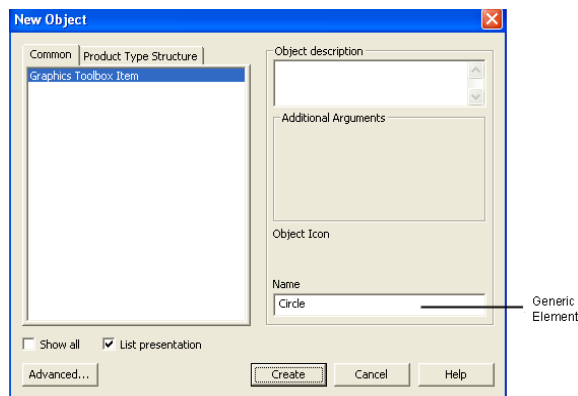
Figure 270. New ToolBox Item object

Generic Element appears for the object in the aspect list as shown in Figure 271.
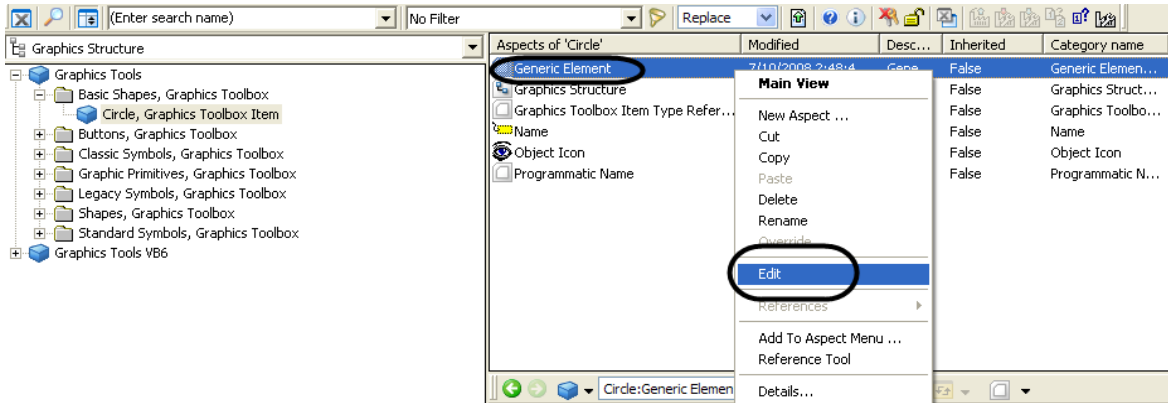
Figure 271. Editing Generic Element and Programmatic Name

3. Select the **Programmatic Name** aspect of the toolbox item object as shown in Figure 271.

4. Type a name for this aspect and click **Apply**.
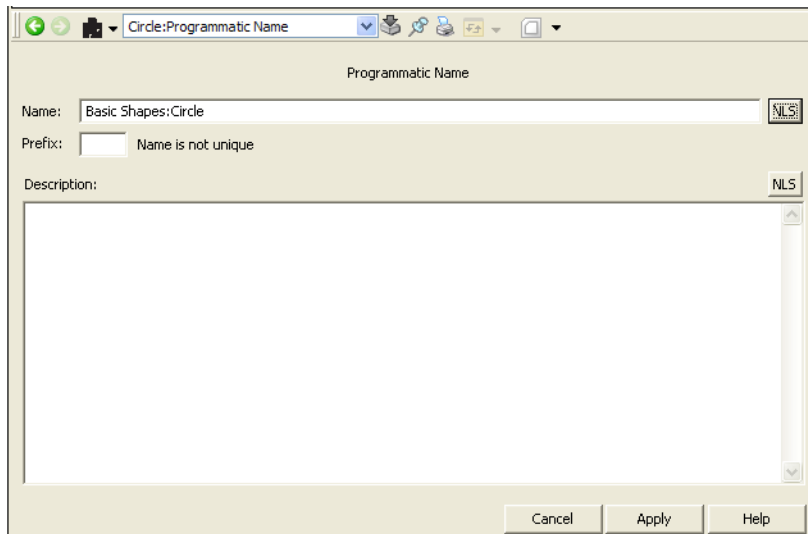
Figure 272. Programmatic Name settings

5. Edit the generic element using the Graphics Builder. For more information on editing a graphic aspect, refer to Launching the Graphics Builder on page 32.

Create a graphic aspect in the **Functional Structure** and edit it in the Graphics Builder. The newly created object appears as a toolbox in the Graphics Builder.

Creating a Generic Element in library

Generic Elements are created in libraries to prevent the users from modifying them.

Creating generic elements in a library includes three major steps, which are:

1. Create library. For more information on creating libraries, refer to *System 800xA, Control, AC 800M Configuration (3BSE035980*).*

2. Create toolbox.

    –   Creating a **Graphics Toolbox Type** object in the library.

     –   Creating a corresponding **Graphics Toolbox** object in the **Graphics Structure**.

3.   Create generic element.

     –   Create a **Graphics Toolbox Item Type** object in the library (in the **Object Type Structure**).

     –   Add the necessary configurations.

     –   Create an instance of the newly created library object in the **Graphics Structure**.

Creating generic element

To create a generic element in the library:

1.   Create an object *Basic Shapes* of **Graphics Toolbox Type** in the library that is created in **Object Type Structure.**
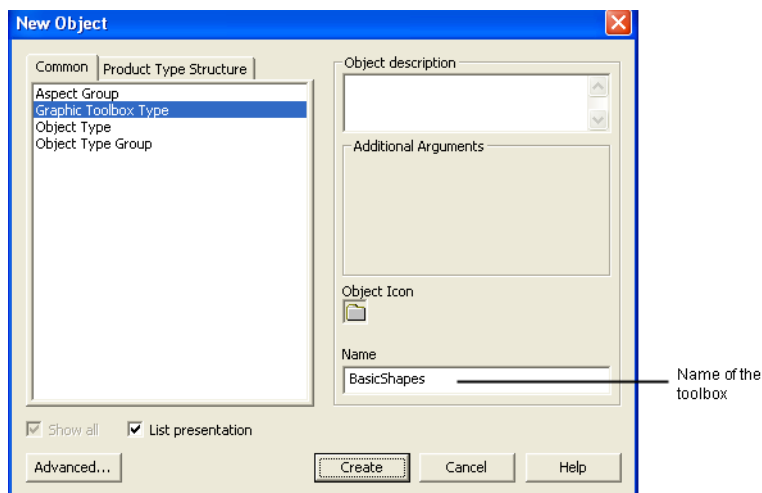
Figure 273. New Toolbox object

2.   Create an object *Images* of **Graphics Toolbox Item Type** within the *Basic Shapes* object as mentioned in step 1.
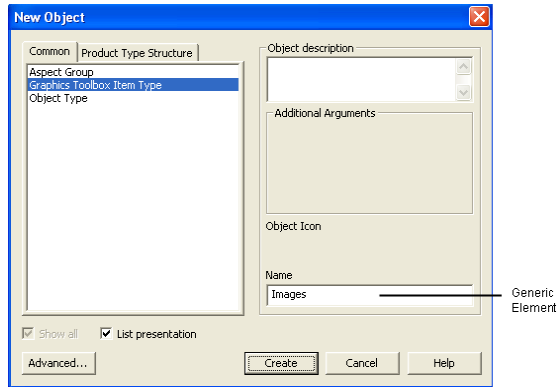
Figure 274. New ToolBox Item object

3.   Select the **Type Definition** aspect of the toolbox item type object.
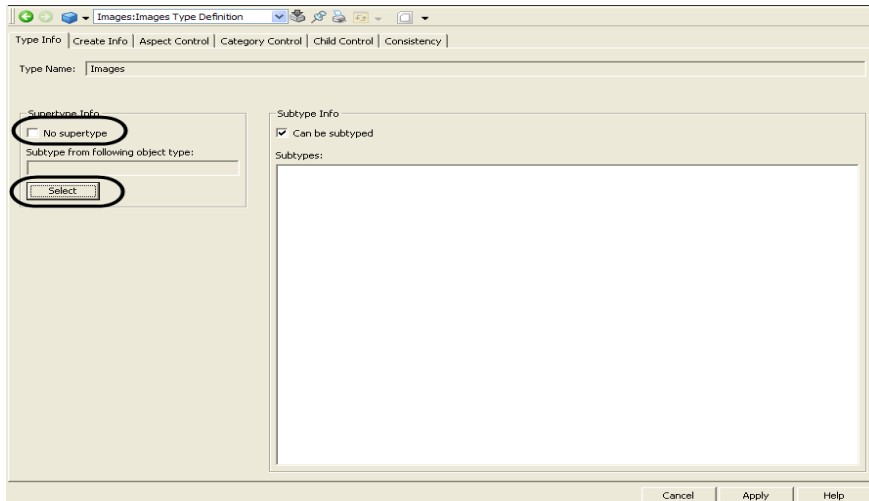
Figure 275. Type Definition settings

4. Clear the **No supertype** check box and click **Select** as shown in Figure 275. The **Select Object Type** dialog appears.
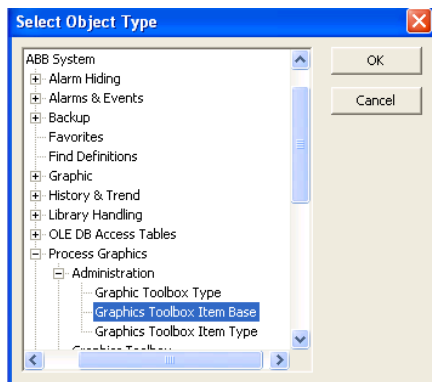
Figure 276. Select Object Type

5. Select the object type as **Graphics Toolbox Item Base** from **Object Types > ABB System > Process Graphics > Administration** and click **OK**.

6. Select the **Generic Element** aspect of the toolbox item object as shown in Figure 277.
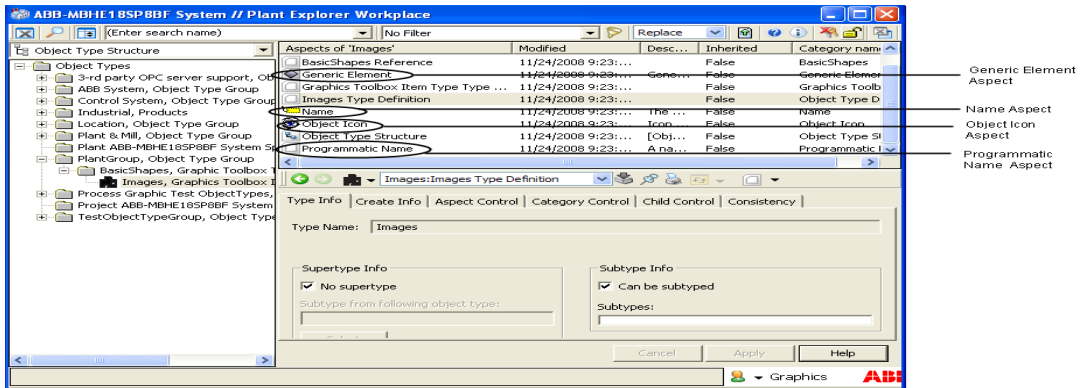
Figure 277. Aspects of Toolbox Item object

7.   Click the **Aspect Info** tab and select **Inheritance Enabled**.
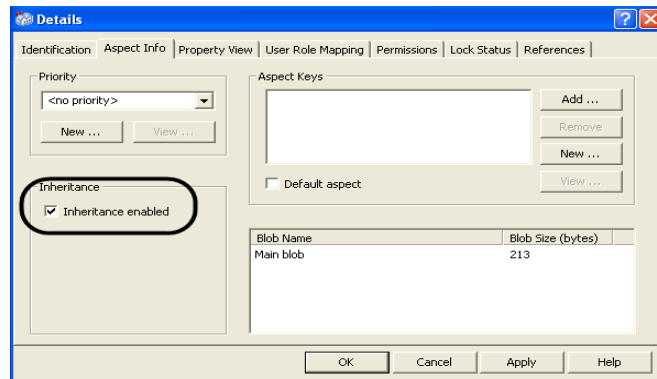
Figure 278. Generic Element settings

8.   Select the **Name** aspect of the toolbox item object as shown in Figure 277.
     Type a name for the aspect in the preview area.
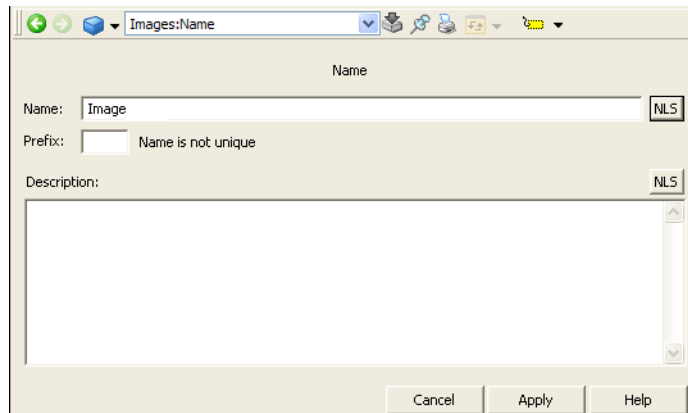
Figure 279. Name settings

9.   Select the **Programmatic Name** aspect of the toolbox item object as shown in Figure 277.

10. Type a name for this aspect and click **Apply**.
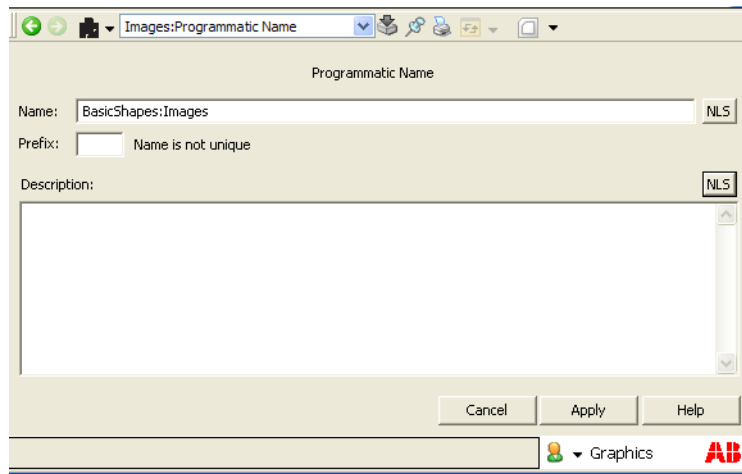
Figure 280. Programmatic Name settings

11. Right-click on **Programmatic Name** aspect and select **Details** from the context menu. Repeat step 7.

12. Right-click on **Object Icon** aspect (refer Figure 277) and select **Details** from the context menu. Repeat step 7. This aspect is used for defining icon for the element in the toolbox.

13. Edit the **Generic Element** aspect using the Graphics Builder. For more information on editing a graphic aspect, refer to Launching the Graphics Builder on page 32.

Creating instances of generic element

To create an instance of the generic element in **Graphics Structure**:

1. Create an object of **Graphics Toolbox** type in **Graphic Tools.** In this example, the toolbox *Basic Shapes* is created as shown in Figure 269.

2. Create an object *Image* of the *Images* object type within *Basic Shapes* toolbox object.

Select **Show Al**l to view all object types as shown in Figure 281.
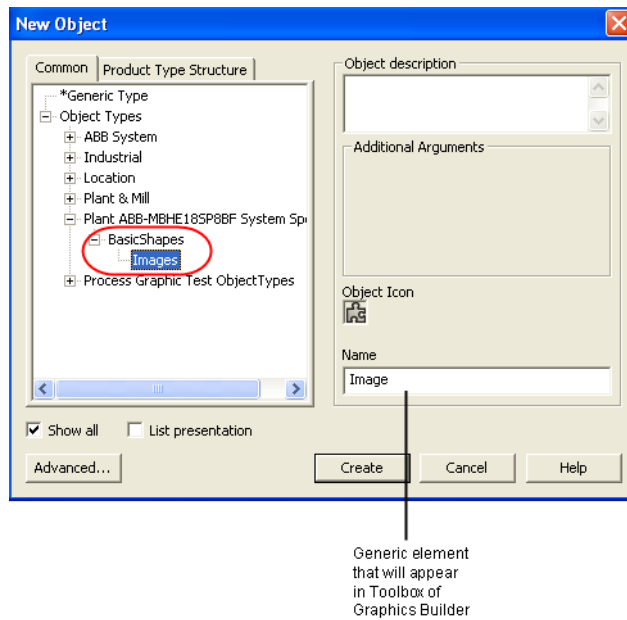
Figure 281. Instance of a Generic Element

3.   Create a graphic aspect in the **Functional Structure** and edit it in the Graphics Builder.

The newly created object appears as a toolbox in the Graphics Builder.

Appendix E Sizes of Faceplates

To group the faceplates created in different versions of the System 800xA, it is recommended that these guidelines are followed.

This section guides the Faceplate Element builders to configure a faceplate element such that the faceplate works as expected. All sizes are measured in pixel units.

Default Faceplate Element

To fit into a normal faceplate a Faceplate Element should have default size according to the table below.

Table 312. Faceplate Element Size

Unit	Height	Width
Pixel	220	250
Twip	3300	3750

The ratio between Height and Width will then be 0.88.

Default Faceplate

A faceplate should have default size according to the table below:.

Table 313. Faceplate Size

Unit	Height	Width
Pixel	400	250

In later versions of the product, when faceplates can be grouped together, faceplates must have the same size to fit in the group display.

Sizes of Each Default Component in a Faceplate

A normal faceplate has a default configuration, which holds:

- A default sized header

- A status and navigation bar with 1 row and 6 positions

- A faceplate element area

- A button area with 2 rows and 6 buttons per row

The sizes of these are added in the table below. Note that the width is not added since all components reside in the same space width.

Table 314. Sizes of Default Component in a Faceplate

Component	Height	Width
Header	46	250
Status and Navigation	46	249
Element	220	250
Buttons	87	249
Sum	399	250

Non-default Faceplate

Any view of the faceplate can be of any size. (The normal faceplate view should have a height of 400 pixels and a width of 250 pixels.) There is a minimum size allowed in a faceplate, that is, a single header with the width of three buttons. This gives a minimum height of 46 pixels and a minimum width of 126 pixels.

Size of Each Part of the Components

Each part of a component (for example, an aspect link in the Status and Navigation bar) has a fixed size. To best fit in the Faceplate Element, calculate the size it should have according to the rules below.

Table 315. Component Size

Component	Part No	Height	Width
Header		46	-
Status and Navigation	First	46	44
	Following	41	41
Element	Tab row	24	-
	Element	-	-
Buttons	First	46	44
	Following	41	41

Header

The header has a fixed height of 46 pixels and a non-fixed width, that must be at least 88 pixels wide.

Status, Navigation Bar and Buttons

Indicators, Aspect links and buttons have the same size, that is, 38x38 pixels. The space between the "buttons" is 3 pixels wide and high and the bottom horizontal line is 2 pixels high. So, the first "button" is 3+38+3 (=44) pixels wide and 3+38+3+2 (=46) pixels high and the following are 3+38 (=41) pixels wide and high.

The sum of 6 indicators (and/or Aspect links) on one row is 44+5*41 (=249) pixels wide and 46 pixels high.

The sum of 6 buttons (on one row) times 2 rows is 44+5*41 (=249) pixels wide and 46+41 (=87) pixels high.

Element

The Faceplate Element varies in size. The faceplate element uses the left over space in the faceplate. To preserve the aspect ratio and to keep the font size, it is recommended to calculate the faceplate element height and width as specified in this section.

Tab Rows

If the tabs are used in any form (happens if more than one faceplate element is configured in a faceplate view) each tab row takes 24 pixels in height. More than one tab group does not take any more space (except if they are configured vertically, and adding more pixels on tab heights).

Example

Assume that the user requires to design a reduced faceplate and a normal faceplate. The different views should contain these component parts:

Table 316. Example Faceplate Content

Component	Reduced	Normal
Header	1	1
MaxNoOfIndicatorRows	0	2
MaxNoOfIndicatorsPerRow	0	5
MaxNoOfButtonRows	1	2
MaxNoOfButtonPerRow	4	5
Elements	1	1

The views to have these sizes:

Table 317. Size of Views

View	Height	Width
Reduced	150	To fit 4 buttons
Normal	400	250

Calculation gives these sizes per component in each view:

Table 318. Sizes per Component

Component	Reduced		Normal	
	Height	Width	Height	Width
Header	46	-	46	-
Status and Navigation	0	0	87	208

Table 318. Sizes per Component (Continued)

Component	Reduced		Normal	
	Height	**Width**	**Height**	**Width**
Buttons	46	167	87	249
Sum	92	167	220	250
Element	58	167	180	250

The element size available is calculated as the space left between the sum of all other components and the total space in the view.

Appendix F Upgrading Faceplates

The Faceplate aspects created in all versions prior to 800xA system version 5.1 uses symbolic names to identify the references to faceplate elements or other properties.

The Faceplate aspects in 800xA system version 5.1 use object IDs and aspect IDs to identify the references. This is used by default for the newly created faceplate aspects. The faceplate aspects created in older versions must be upgraded to use the new reference handling.

Upgrading a faceplate is optional. It is strongly recommended to upgrade the faceplates because of the following reasons:

- Better runtime performance; no name server access required when faceplate is opened.

- More reliable because the references continue to work even if target object is renamed.

- Upgraded faceplates function efficiently with the Consistency Checker and Reference Tool. This behaves similar to a PG2 graphic aspect.

- The upgrade to new reference handling can also repair some broken references.

Upgrading a faceplate aspect

Execute the following steps to upgrade a faceplate aspect:

1. Right-click the faceplate aspect and select **Config View** from the context menu.

2. In the **Layout** tab, select the **New Reference Handling** check box. This check box is enabled for the faceplates that exist in the system.
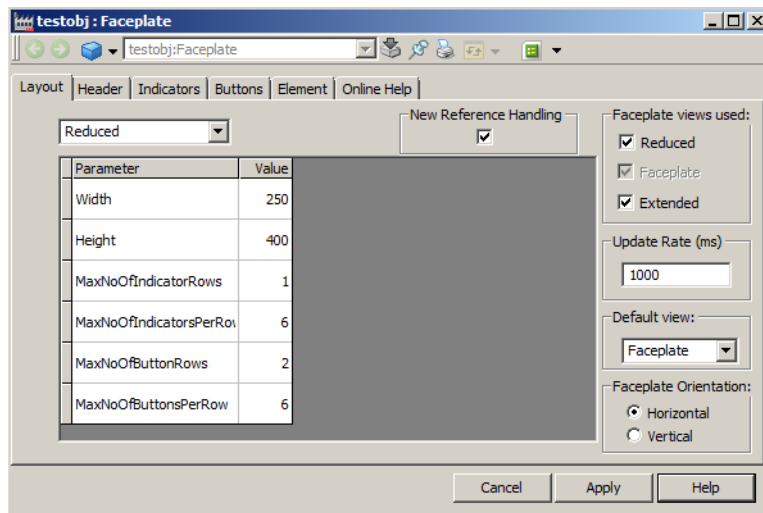
Figure 282. Faceplate Config View

3. Click **Apply** to save the changes.

New Reference Handling will be the default functionality for newly created faceplates.

The user is not allowed to revert back after upgrading a faceplate.

Use the **Consistency Check** tool to upgrade several faceplate aspects at the same time (for example, to upgrade a complete library). Aspects that are not upgraded be displayed as an information. These aspects can be upgraded using the **Repair** option. Execute the following steps to upgrade two or more faceplate aspects at the same time:

1. In the workplace toolbar, select ![icon] icon to launch the **Consistency Check** tool.

2. Drag and drop the library containing faceplate aspects to be upgraded, into the **Consistency Check** dialog.

Ensure that the library is open if the faceplates are in a library.

Figure 283. Consistency Check Tool

3.   Click **Options** and the **Options** dialog appears.
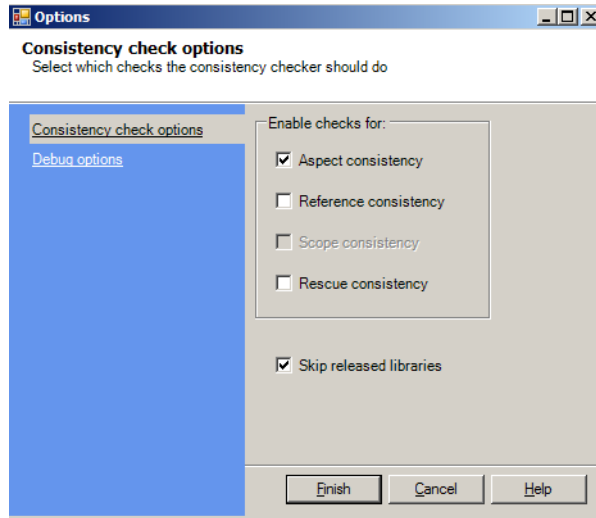
Figure 284. Consistency Check Options

4. Only select the **Aspect consistency** check box and click **Finish**.

5. In the **Consistency Check** dialog, click all the three tabs (**Errors**, **Warnings**, and **Information**) to enable just the messages in **Information** tab.

6. Click **Check** to begin the consistency check.

    Any faceplate aspect that is not upgraded is displayed in the **Information** tab.
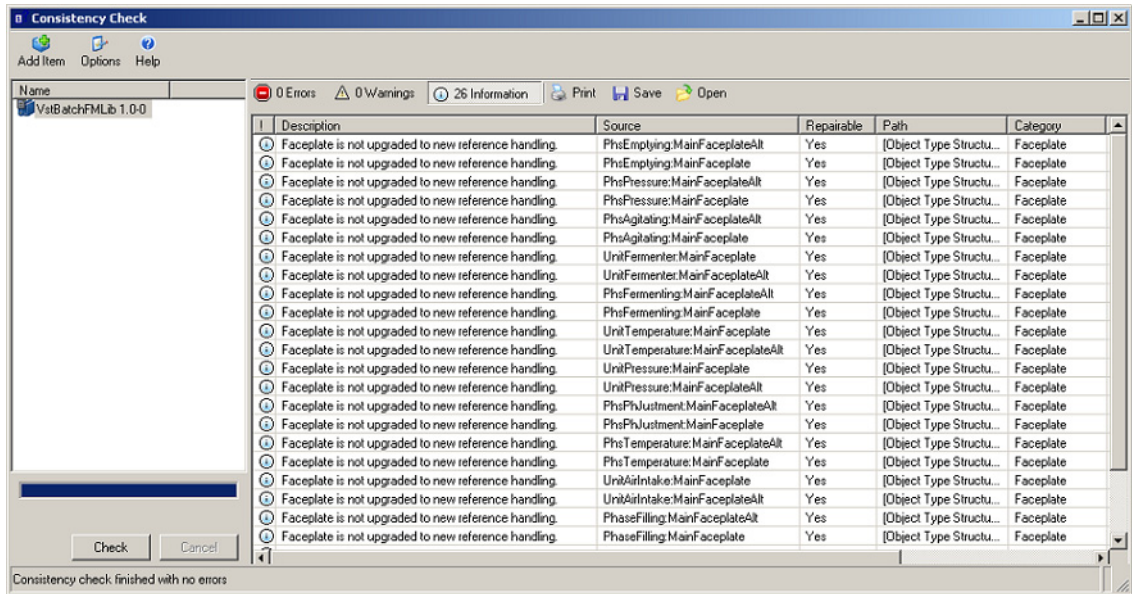
Figure 285. Consistency Check - Information tab

7. To upgrade the faceplate aspects to the new reference handling, select the
    faceplate aspects which display the message *Faceplate is not upgraded to new
    reference handling* in the **Information** tab.

8. Right-click the selected faceplate aspects and select **Repair Aspect** from the
    context menu.
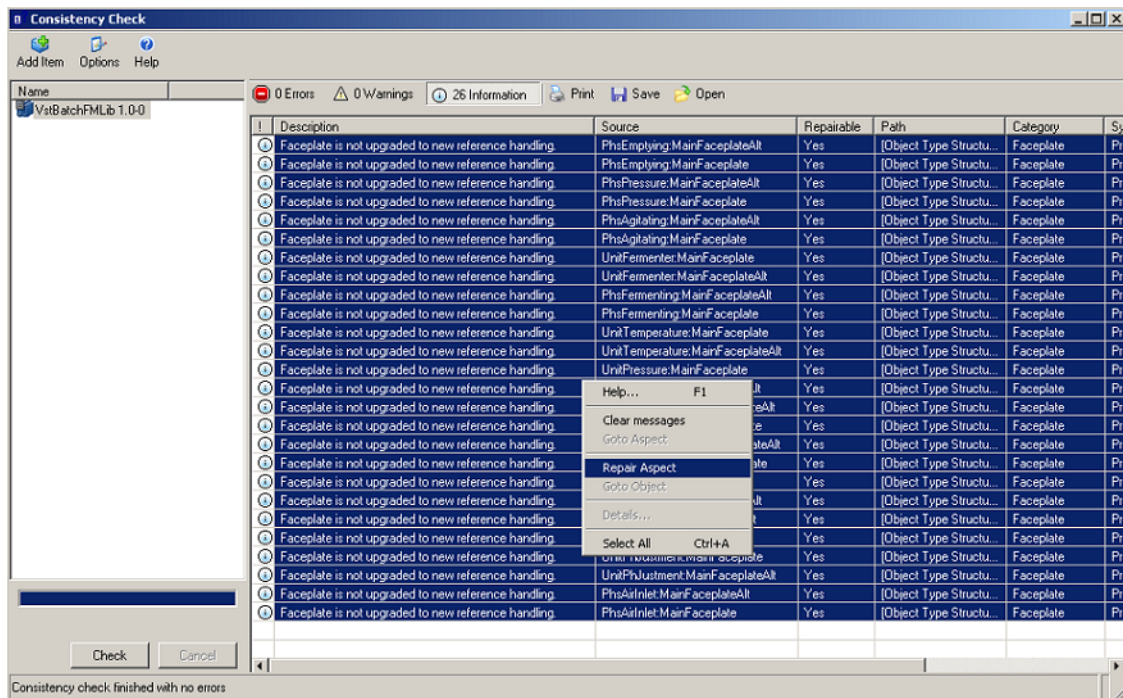
Figure 286. Consistency Check Tool

This prompts for a confirmation. Click **Yes** to upgrade.

This displays a message *Upgrade to new reference handling succeeded* for all upgraded faceplates.

9. Verify that the new references are functioning.

10. In **Options** dialog box, select the **Reference Consistency** check box and click **Finish**.

11. Verify if there exist any errors or warnings in the **Errors** tab or **Warnings** tab.

12. Manually correct the reference errors in the **Config View** of the faceplate aspect.

Appendix G Graphics Builder Shortcut Keys

Table 319 lists the keyboard shortcuts used in the Graphics Builder.

Table 319. Shortcut Keys in Graphics Builder

Menu	Menubar Item	Shortcut key
File	Save	CTRL + S
	Show Migration Errors	CTRL + E
	Diagnostics	CTRL + D
	Print	CTRL + P
	Offline Mode	CTRL + W
	Exit	ALT + F4

Table 319. Shortcut Keys in Graphics Builder (Continued)

Menu	Menubar Item	Shortcut key
Edit	Undo	CTRL + Z
	Redo	CTRL + Y
	Cut	CTRL + X
	Copy	CTRL + C
	Paste	CTRL + V
	Delete	Delete
	Delete Current View	CTRL + Delete
	Select All	CTRL + A
	Select > Right	ALT + right
	Select > Left	ALT + left
	Select > Up	ALT + up
	Select > Down	ALT + down
	Save as Solution	CTRL + SHIFT + N

Table 319. Shortcut Keys in Graphics Builder (Continued)

Menu	Menubar Item	Shortcut key
View	Toolboxes	CTRL + SHIFT + T
	Element Explorer	CTRL + SHIFT + E
	Graphic Items	CTRL + SHIFT + G
	Properties	CTRL + SHIFT + P
	Data References	CTRL + SHIFT + D
	Resource References	CTRL + SHIFT + R
	Input Properties	CTRL + SHIFT + I
	Expression Variables	CTRL + SHIFT + V
	User Enumerations	CTRL + SHIFT + U
	Solution Libraries	CTRL + SHIFT + S
	Test Data	CTRL + SHIFT + A
	Modes > Edit	F5
	Modes > Live	F6
	Zoom > Zoom In	CTRL + +
	Zoom > Zoom Out	CTRL + -
	Zoom > Fit to window	CTRL + 0
	Zoom > Home view	CTRL + 1

Table 319. Shortcut Keys in Graphics Builder (Continued)

Menu	Menubar Item	Shortcut key
Format	Group	CTRL + G
	Ungroup	CTRL + U
	Rotate > RotateBy	CTRL + R
	Rotate > Left	CTRL + ALT + left
	Rotate > Right	CTRL + ALT + right
	Flip > Flip Horizontal	CTRL + SHIFT + 6
	Flip > Flip Vertical	CTRL + SHIFT + 7
	Order > Bring To Front	CTRL + SHIFT + F
	Order > Bring Forward	CTRL + F
	Order > Send Backward	CTRL + B
	Order > Send to Back	CTRL + SHIFT + B
	Align > Align Lefts	SHIFT + left
	Align > Align Centers	CTRL + SHIFT + up
	Align > Align Rights	SHIFT + right
	Align > Align Tops	SHIFT + up
	Align > Align Middles	CTRL + SHIFT + down
	Align > Align Bottoms	SHIFT + down
	Distribute > Distribute Horizontally	CTRL + SHIFT + 8
	Distribute > Distribute Vertically	CTRL + SHIFT + 9
	Horizontal Spacing > Horizontal Spacing Make Equal[*]	F9
	Horizontal Spacing > Increase Horizontal Spacing[*]	F10

Table 319. Shortcut Keys in Graphics Builder (Continued)

Menu	Menubar Item	Shortcut key
Format	Horizontal Spacing > Decrease Horizontal Spacing*	F11
	Horizontal Spacing > Remove Horizontal Spacing*	F12
	Vertical Spacing > Vertical Spacing Make Equal*	CTRL + F9
	Vertical Spacing > Increase Vertical Spacing*	CTRL + F10
	Vertical Spacing > Decrease Vertical Spacing*	CTRL + F11
	Vertical Spacing > Remove Vertical Spacing*	CTRL + F12
	Grid > Snap to Grid	F7
	Grid > Size to Grid	F8
Tools	Options	CTRL + SHIFT + O
	Display Documentation Tool	CTRL + SHIFT + Y
	Reference Documentation	CTRL + SHIFT + J
Window	Reset Docking Layout	CTRL + SHIFT + L
Help	Contents	F1

*   Feature Pack Functionality

Index

Revision History

Introduction

This section provides information on the revision history of this User Manual.

The revision index of this User Manual is not related to the 800xA 5.1 System Revision.

Revision History

The following table lists the revision history of this User Manual.

Revision Index	Description	Date
-	First version published for 800xA 5.1	June 2010
A	Second version published for 800xA 5.1 Rev A	May 2011
B	Third version published for 800xA 5.1 Feature Pack 1	August 2011
C	Updated for 800xA 5.1 Rev B	June 2012
D	Updated for 800xA 5.1 Feature Pack 3	August 2012
E	Updated for 800xA 5.1 Feature Pack 4	February 2013

Updates in Revision Index A

The following table shows the updates made in this User Manual for 800xA 5.1 Rev A.

Updated Section/Sub-section	Description of Update
Section 3, Graphic Aspects	Added a section *Printing a Graphic Aspect*.
Appendix A, Standard Building Blocks	Added an information on the usage of the primitive in the *Elevator* and *Conveyor* sub-sections. Added a sub-section *Tooltip* in the section *Input Items*.

Updates in Revision Index B

The following table shows the updates made in this User Manual for 800xA 5.1 Feature Pack 1.

Updated Section/Sub-section	Description of Update
About this User Manual	Added a section *Feature Pack* describing the user manual conventions used for indicating the Feature Pack content.
Section 4, Expressions	Added the functions *Limit* and *LimitArray* to the *Miscellaneous Functions* and *Array Function*s tables respectively in the *Expression Functions* section.
Section 7, Faceplate Framework	Added the details of hiding the faceplate header and hiding the faceplate footer in the *Header Area* and *View Selection Buttons* sections respectively.
Appendix A, Standard Building Blocks	Added the details of High Performance Primitives in a section *High Performance* with the sub-sections *High Performance Pipe, High Performance Trend, TwoScreenNavigate, ThreeScreenNavigate*, and *FourScreenNavigate*.

Updates in Revision Index C

The following table shows the updates made in this User Manual for 800xA 5.1 Revision B.

Updated Section/Sub-section	Description of Update
Section 3, Graphic Aspects	Added sub section *Indication of Broken Graphic Element References.*
Section 4, Expressions	Update on ViewReference expression for Aspect View Wrapper.
Appendix B, Standard Building Blocks for AC800M	Following topics have been added: • *AC 800M Display Elements* section has been updated for graphic element aspects. • *AC 800M Faceplate Elements* section has been updated for generic elements. • *AC 800M Symbols* section has been updated for symbols.

Updates in Revision Index D

The following table shows the updates made in this User Manual for 800xA 5.1 Feature Pack 3.

Updated Section/Sub-section	Description of Update
Section 3, Graphic Aspects	Updated the section *Invoke Object Context Meny* with the information on restricting access to Acknowledge All Visible Alarms in the context menu.

Updated Section/Sub-section	Description of Update
Appendix A, Standard Building Blocks	1. Added the details of out terminal in the section *Aspect View Button*. 2. Added a new section *Aspect View Button with History*.
Appendix B, Standard Building Blocks for AC 800M	Property tables updated for the sections *AC 800M Display Elements*, *AC 800M Faceplate Elements*, and *AC 800M Symbols*.

Updates in Revision Index E

The following table shows the updates made in this User Manual for 800xA 5.1 Feature Pack 4.

Updated Section/Sub-section	Description of Update
Section 2, Graphics Builder	1. Added an information with respect to the change in status of the Graphics Builder in the section *Status Bar*.
	2. Added the section *Horizontal or Vertical Spacing of Graphic Items*.
	3. Added a tip with respect to replacing a graphic element in the section *Element Explorer*.
	4. Added a few features of Properties Window in the section *Properties Window*.
	5. Added a tip describing an icon for the Expression Editing area in the section *Expression Editing Area*.
	6. Added information about Modifying and Renaming an expression variable in the section *Expression Variables*.
	7. Added a sub section *Hotspots in a solution* in the section *Solution Library > Adding a solution to the edit panel*.
	8. Added *Set Hotspot* information in the table *Context menu of a solution* of the section *Context Menu and Toolbar*.
	8. Added the details of Enum data type for input properties in the table *Test Data Columns* of the section *Test Data*.
Section 3, Graphic Aspects	Added a new section *ain Object Awareness* in the section *Handling Mouse Events*.

Updated Section/Sub-section	Description of Update
Appendix A, Standard Building Blocks	1. Updated the section *Aspect View Button*. 2. Removed the section *Aspect View Button with History*. 3. Added the sections *High Performance Trend 2, High Performance Bar, High Performance FP Bar, High Performance Alarm Indication, High Performance Voltmeter, High Performance Interlock, High Performance Radar 3 Spokes, High Performance Radar 4 Spokes, High Performance Radar 5 Spokes, High Performance Profile Indication, High Performance Profile Indication Map* 3. Moved the sections *TwoScreenNavigate, ThreeScreenNavigate*, and *FourScreenNavigate* to the section *Navigation*. 4. Added information on *ainObjectAwareness* in the section *Input Items*.
Appendix G, Graphics Builder Shortcut Keys	Added the shortcut keys for the options available in *Format > Horizontal Spacing* and *Format > Vertical Spacing*.

Contact us

ABB AB
Control Technologies
Västerås, Sweden
Phone: +46 (0) 21 32 50 00
e-mail: processautomation@se.abb.com
www.abb.com/controlsystems

ABB Automation GmbH
Control Technologies
Mannheim, Germany
Phone: +49 1805 26 67 76
e-mail: marketing.control-products@de.abb.com
www.abb.de/controlsystems

ABB S.P.A.
Control Technologies
Sesto San Giovanni (MI), Italy
Phone: +39 02 24147 555
e-mail: controlsystems@it.abb.com
www.abb.it/controlsystems

ABB Inc.
Control Technologies
Wickliffe, Ohio, USA
Phone: +1 440 585 8500
e-mail: industrialitsolutions@us.abb.com
www.abb.com/controlsystems

ABB Pte Ltd
Control Technologies
Singapore
Phone: +65 6776 5711
e-mail: processautomation@sg.abb.com
www.abb.com/controlsystems

ABB Automation LLC
Control Technologies
Abu Dhabi, United Arab Emirates
Phone: +971 (0) 2 4938 000
e-mail: processautomation@ae.abb.com
www.abb.com/controlsystems

ABB China Ltd
Control Technologies
Beijing, China
Phone: +86 (0) 10 84566688-2193
www.abb.com/controlsystems

3BSE049230-510 E

Power and productivity
for a better world™

ABB