# GraphWorX64 – Smart Symbol Binding with Real Time Data

APPLICATION NOTE

September 2013

**Description:** A quick example of using advanced properties of smart binding and smart symbols.
**OS Requirement:** Vista x64/ Windows 7 x64/ Windows 8 x64/ Windows Server 2008 x64/ Windows Server 2008 R2 x64/ Windows Server 2012
**General Requirement:** GENESIS64 10.8 and familiarity with smart binding in GraphWorX64.

## Introduction

Smart Symbols are an enhanced version of object groups. They greatly simplify the configuration and save engineering time by allowing a configuration of properties of inner objects without a need to drill-into the symbol.

This document focuses on a couple of new smart properties which provides engineers with an option to use resolved data source values in their binding logic. It is very convenient to be able to use OPC values as parameters inside other data sources instead of using scripts in displays. With this feature data sources can also feed properties of 3rd party WPF controls directly without the need for retrieving the values with a script.

For further information on Smart Symbols and the Smart Symbol Property Binding Editor see the GraphWorX64 application notes on Smart symbols or consult the help files:

## Runtime Value Properties

In this version of GraphWorX64 there are two new smart property types:

- StringDataSourceAndRuntimeValue
- StringDataSourceWithTokensAndRuntimeValue

These new types have the same characteristics as these preexisting types:

- StringDataSource
- StringDataSourceWithTokens

However, these new types also expose the runtime value of the data source as a bind able property instead of just the data source name.

Because the data type of the runtime value is not known until runtime, in the Smart Binding Editor dialog you will need to uncheck the "Show Compatible Properties Only" checkbox in order to see the properties for the runtime values.

For example, if you define a smart property of type "StringDataSourceAndRuntimeValue" named "MyProperty", in the binding editor you will see two properties: one named "(MyProperty)" which is the data source name, and another named "(MyPropertyRuntimeValue)" which is the runtime value of the data source.

In configuration mode, the runtime value of these smart properties is null. In runtime mode, the runtime value will automatically be converted to the data type of the target (bound) property, if possible (this is usually possible if the value can be converted to and from a string).

## Data Source Definition based on other Data Source Output Value

This example will show you how to use output of one OPC data source as part of another data source definition.

1. Open a new GraphWorX64 display by going to Start → All Programs → ICONICS → GENESIS64 → GraphWorX64 → GraphWorX64.
2. Create two process points representing local simulation variables such as "**localsim:sine**" and "**localsim:ramp**".
3. Create a third process point which should be **Data Entry**. For data source enter a String local user variable with no initial value, similar to the variable in Figure 1.

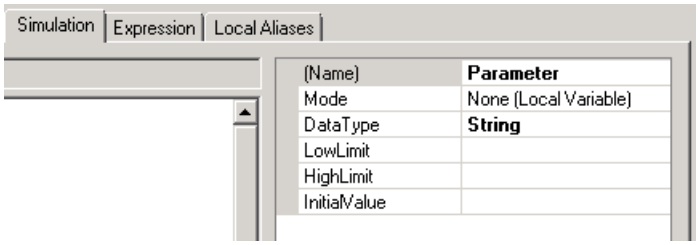


**Figure 1 - Creating Empty Local Variable**

4. **Duplicate** this process point.
5. Select the duplicated process point, right-click on it and select **Group**, then right-click on it again and select **Convert to Smart Symbol**.

   **NOTE:** If you don't know how to use or create Smart Symbols see application note entitled *GraphWorX64 - Smart Symbols*.

6. In Automatic Smart Binding Options select to **Expose data sources as Smart Properties**


**Figure 2 – Smart Binding Option**

7. Give it a name of your choice and change the property type to **StringDataSourceAndRuntimeValue**
8. If you check the properties of the smart symbol you should see one smart property which should contain the original data source

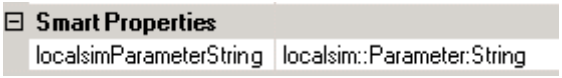


**Figure 3 - Smart Property**

9. Expand the Smart Symbol in object Explorer, select the label and look at the Dynamics panel. You should see the output property of the smart property similar to Figure .

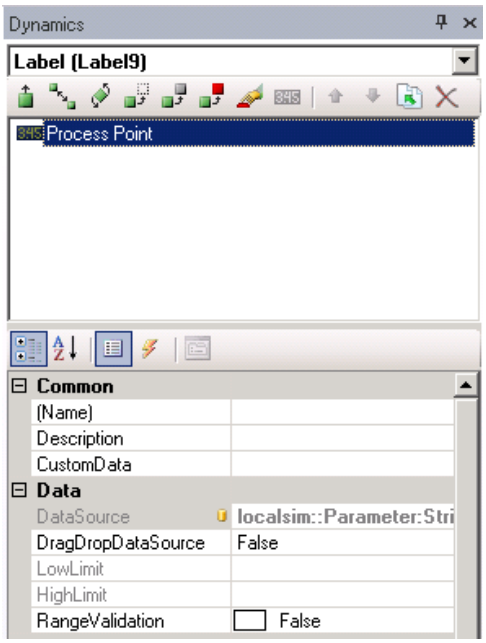


**Figure 4 - Smart Data Source Property**

10. Right-click on **DataSource** and select **Smart Binding Editor**. You should see a similar window to Figure .

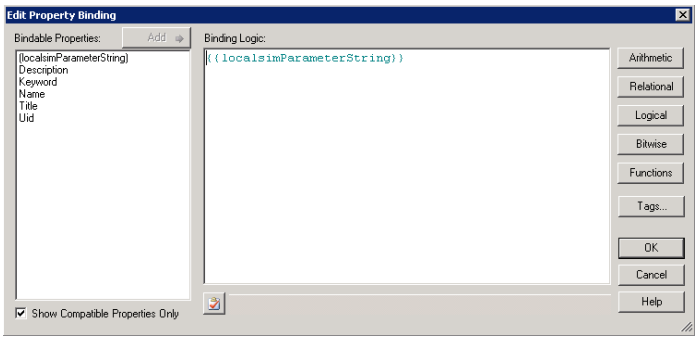


**Figure 5 - Smart Symbol Binding Editor**

11. In the bottom left, uncheck Show Compatible Properties Only.
12. Delete the binding logic and using the editor create the binding logic below:

"localsim:" + tostring({{localsimParameterStringRuntimeValue}})

13. Now if you go to runtime you should see zero in the Smart symbol, nothing in its predecessor and running values in the other two process points.
14. Type in "**sine**" or "**ramp**" (without the quotes) into the empty process point and you should see a value identical to the respective process point.


**Figure 2 - Data Source Changing by Parameter**

### Animating vertices of a polyline using local variables

1. Create a new GraphWorX display.
2. Insert a **Canvas** panel from Home ribbon

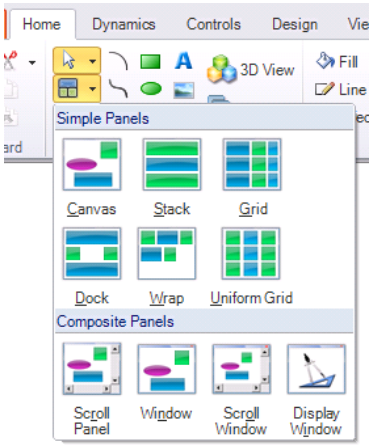


**Figure 3 - Panel Selection**

3. Insert a polyline into the canvas so as it stretches from the top right corner to the bottom left corner.
4. Group the polyline and convert the group to a smart symbol.
5. In the Edit Smart Properties dialog, add 10 properties with a type of **StringDataSourceAndRuntimeValue** and name them V01, V02, etc. up to V10.
6. Find the **Points** property of the polyline, right click on it and choose **Smart Binding Editor**.
7. Create the following Binding Logic:

```
"10," + tostring({{V01RuntimeValue}}+10) +
" 20," + tostring({{V02RuntimeValue}}+10) +
" 30," + tostring({{V03RuntimeValue}}+10) +
" 40," + tostring({{V04RuntimeValue}}+10) +
" 50," + tostring({{V05RuntimeValue}}+10) +
" 60," + tostring({{V06RuntimeValue}}+10) +
" 70," + tostring({{V07RuntimeValue}}+10) +
" 80," + tostring({{V08RuntimeValue}}+10) +
" 90," + tostring({{V09RuntimeValue}}+10) +
" 100," + tostring({{V10RuntimeValue}}+10)
```

8. Next fill in the Smart properties similar to Figure 4.

| | |
|---|---|
| **Measurements** | |
| Angle | **0** |
| DescendantBounds | **0,0,0,0** |
| **Smart Properties** | |
| V01 | localsim:sine |
| V02 | localsim:sine:::::500 |
| V03 | localsim:sine:::::1000 |
| V04 | localsim:sine:::::1500 |
| V05 | localsim:sine:::::2000 |
| V06 | localsim:sine:::::2500 |
| V07 | localsim:sine:::::3000 |
| V08 | localsim:sine:::::3500 |
| V09 | localsim:sine:::::4000 |
| V10 | localsim:sine:::::4500 |

**Figure 4 - Smart Properties Constituting Vertices**

9. Enter Runtime and see the polyline being drawn.

**Connecting WPF controls**

1. Create a new GraphWorX display.
2. Open the **Toolbox** panel in GraphWorX64 and pick **WPF Controls** from the dropdown.
3. Insert a **ProgressBar** control
4. Group it and convert it to a Smart Symbol
5. Drill into the symbol, find **Value** property of the Progress Bar control, and add it to **Smart Properties**
6. Select the symbol, right-click on it, select **Edit Smart Properties**, and change the type of Value from double to **StringDataSourceAndRuntimeValue**
7. Drill into the symbol and select the Progress Bar Control again. Right-click on Value, select the **Smart Binding Editor**, and insert the "**(ValueRuntimeValue)**" bindable property, like in Figure 5.

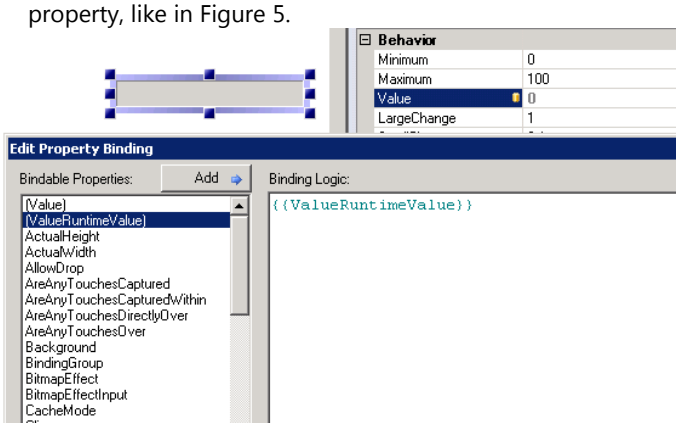


**Figure 5 - Creating Binding to WPF Progress Bar**

8. Select the symbol again and connect the smart property to an input variable like "localsim:ramp".
9. Enter runtime to see the progress bar working.

## More Efficient Binding Logic

Using the StringDataSourceAndRuntimeValue property type you can more easily create your calculations, though this way is less efficient in runtime.

The following two binding logics will give you the same results. The first one is more difficult to configure for the binding expression, but more efficient at runtime. The second one is easier to configure, but less efficient in runtime.

"x=-{{" + {{DataSource}} + "}}" - {{DataSourceRuntimeValue}}