GraphWorX64 Scripting
Shape Manipulation

APPLICATION NOTE

September 2013

Description: Guide to runtime shape manipulation using GraphWorX64 Scripting
OS Requirement: Vista x64/ Windows 7 x64/ Windows 8 x64/ Windows Server 2008 x64/ Windows Server 2008 R2 x64/ Windows Server 2012
General Requirement: Installation of GENESIS64. Basic GraphWorX64 Scripting knowledge

Introduction

Scripting allows you full access to the GENESIS64 API. In this document, we are going to see how we can use GraphWorX64 API to alter a shape's appearance on the display while in Runtime mode.

Display Setup

To become familiar with shape manipulation we are going to create an example based on a display containing a rectangle, which is the shape we want to manipulate, as well as a few buttons to run some scripts.

The buttons we want to create will let us do the following:

- Alter the rectangle's position
- Rotate the rectangle
- Change the rectangle's color

We will now create a display containing a rectangle, and nine buttons - four of which will be used to move the rectangle up, down, left, and right. Two of the buttons will be used to rotate the rectangle clockwise and counter-clockwise. The three remaining buttons will be used to set the color to red, green and blue.

Feel free to arrange these elements as you wish. **Figure 1** shows the setup we created for this example.

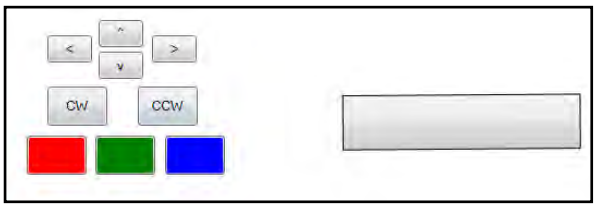

Figure 1 - The Display with a Rectangle and Nine Buttons

Now we can go on and create the script code needed to perform the nine different tasks.

Script Definition

The nine script functions we are going to create are very similar to each other. In all of them we get a reference to the rectangle object we have on the screen (therefore it is important that the rectangle has been given a name, "Rect1" in our case), and then we call an API function with some parameters which let us perform the desired action.

The first function we will analyze is called MoveUp, which will move the rectangle up by 10 pixels in our application
In order to create the script module and bind it to our button;

1) Select the *Up* button to display the properties of the button and go to the Dynamics Properties.
2) Click the Command field and select "Run Script" from the dropdown menu.
3) Click the "Events" button , enter "MoveUp" in the RunScript field and press Enter. In the Script Options window, leave the "Add a new Form" checkbox unchecked and click on OK.

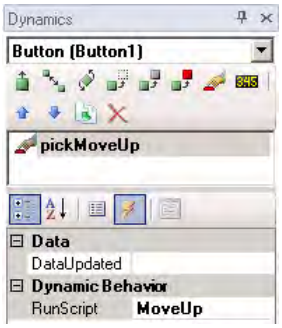

Figure 2 - Assigning the Script Function to a Pick Action

The basic structure of the function is created with no functionality. In order to move *Rect1* 10 pixels up, add the following code between the curly brackets of the function. Your function should look similar to the one below,

```
function MoveUp(sender : System.Object, cmdArgs :
Ico.Gwx.CommandExecutionEventArgs)
{
    var rect : GwxElement =
GwxElement(ThisConfiguration.GetObjectByName("Rect1
"));
    rect.MoveBy(new Vector(0,-10), null);
}
```

The function body contains two lines of code; the first one simply gets a reference to our rectangle and place it in a variable called "rect". It will be replicated among all the other functions.

In the second line we use one of the API functions called **MoveBy** to relocate the rectangle up by 10 pixels, specifying a Vector object with (x,y) coordinates relative to the current rectangle position. Since we have selected (0, -10) the x coordinate will not change, while the rectangle will move up 10 pixels along the y coordinate.

> **NOTE:** The coordinate system has the point (0, 0) in the top left corner of the display. The x coordinate grows going right, the y coordinate grows going down.

As a second example, we will now analyze the function used to rotate the rectangle clockwise.

```
function RotateClockWise(sender : System.Object,
cmdArgs : Ico.Gwx.CommandExecutionEventArgs)
{
    var rect : GwxElement =
GwxElement(ThisConfiguration.GetObjectByName("Rect1
"));
    rect.RotateBy(-15, null);
}
```

This function is very similar to the previous one, but the API function used here is called **RotateBy**. The first parameter is the angle in degrees, the second parameter is a Transform type object and we can ignore it for now and just specify null for it.

We then assign this function to the pick action of the button we want to use for clockwise rotation.

The last function we will look at here is the one to make the rectangle change to red.

```
function SetRedColor(sender : System.Object,
cmdArgs : Ico.Gwx.CommandExecutionEventArgs)
{
    var rect : GwxRectangle =
GwxRectangle(ThisConfiguration.GetObjectByName("Rec
t1"));
    rect.Fill = Brushes.Red;
}
```

In this case, the function is even simpler compared to the others, as we just need to alter the **Fill** property, assigning it one of the predefined brushes in the system. Again, we assign this function to the pick action for the red button we added to the display.

After completing the creation of all the script functions, we can go into Runtime mode and test our example to see the result. Here are some screenshots of the running display:

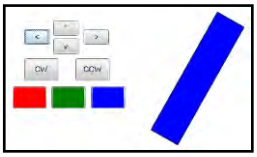

Figure 3 - The Display in Runtime

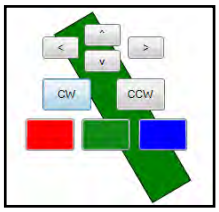

Figure 4 - The Rectangle Moved Behind the Command Buttons

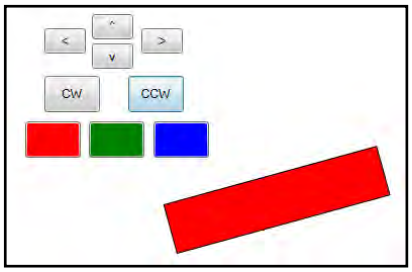

Figure 5 - The Rectangle Rotated, Moved, and Changed Color

Full Script Code

For your convenience, here is the full source code for this example. You can put all the functions inside the same module; just make sure the nine buttons are correctly set up to use these functions.

```
function RotateClockWise(sender : System.Object, cmdArgs :
Ico.Gwx.CommandExecutionEventArgs)
{
    var rect : GwxElement =
GwxElement(ThisConfiguration.GetObjectByName("Rect1"));
    rect.RotateBy(-15, null);
}

function RotateCounterClockWise(sender : System.Object,
cmdArgs : Ico.Gwx.CommandExecutionEventArgs)
{
    var rect : GwxElement =
GwxElement(ThisConfiguration.GetObjectByName("Rect1"));
    rect.RotateBy(15, null);
}

function MoveRight(sender : System.Object, cmdArgs :
Ico.Gwx.CommandExecutionEventArgs)
{
    var rect : GwxElement =
GwxElement(ThisConfiguration.GetObjectByName("Rect1"));
    rect.MoveBy(new Vector(10,0), null);
}
function MoveLeft(sender : System.Object, cmdArgs :
Ico.Gwx.CommandExecutionEventArgs)
```

```
{
    var rect : GwxElement =
GwxElement(ThisConfiguration.GetObjectByName("Rect1"));
    rect.MoveBy(new Vector(-10,0), null);
}
function MoveUp(sender : System.Object, cmdArgs :
Ico.Gwx.CommandExecutionEventArgs)
{
    var rect : GwxElement =
GwxElement(ThisConfiguration.GetObjectByName("Rect1"));
    rect.MoveBy(new Vector(0,-10), null);
}
function MoveDown(sender : System.Object, cmdArgs :
Ico.Gwx.CommandExecutionEventArgs)
{
    var rect : GwxElement =
GwxElement(ThisConfiguration.GetObjectByName("Rect1"));
    rect.MoveBy(new Vector(0,10), null);
}

function SetRedColor(sender : System.Object, cmdArgs :
Ico.Gwx.CommandExecutionEventArgs)
{
    var rect : GwxRectangle =
GwxRectangle(ThisConfiguration.GetObjectByName("Rect1"));
    rect.Fill = Brushes.Red;
}

function SetGreenColor(sender : System.Object, cmdArgs :
Ico.Gwx.CommandExecutionEventArgs)
{
    var rect : GwxRectangle =
GwxRectangle(ThisConfiguration.GetObjectByName("Rect1"));
    rect.Fill = Brushes.Green;
}

function SetBlueColor(sender : System.Object, cmdArgs :
Ico.Gwx.CommandExecutionEventArgs)
{
    var rect : GwxRectangle =
GwxRectangle(ThisConfiguration.GetObjectByName("Rect1"));
    rect.Fill = Brushes.Blue;
}
```

Conclusions

This was just a simple example, using a single shape. You can easily create more complex examples with multiple shapes on the display.

You can also make these functions capable of receiving some parameters so that you don't have to create one of them for each single shape, and you can even collapse the functions above to just three of them. This can be done by using the CustomData property of the pick actions. Consider for example the revised version of the Rotate function, defined in this way:

```
function Rotate(sender : System.Object, cmdArgs :
Ico.Gwx.CommandExecutionEventArgs)
{
    var rect : GwxElement =
GwxElement(ThisConfiguration.GetObjectByName("Rect1
"));
    rect.RotateBy(sender.CustomData, null);
}
```

As you can see, there are only two differences from the previous version: the name of the function and the first parameter of the RotateBy function. In this case, we are telling the function to use the CustomData property of the sender. The sender is the pick action who generated this event, and in Figure 6, you can see where you can edit the CustomData property for a pick action.

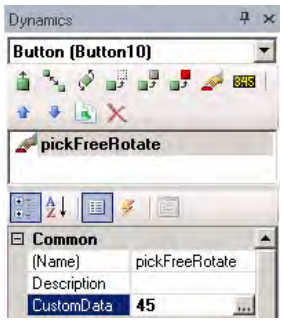

Figure 6 - The CustomData Property

In this case, when executing this pick action, we are passing the value "45" to the RotateBy function, and the rectangle will be rotated counter-clockwise by 45 degrees. To rotate clockwise, you can simply specify a negative number here.

Using these concepts, you can call many different API functions to have full control of all of your objects currently on the display. It is also possible to create new objects and place them on the screen dynamically.

You can find more information about all the available functions by reading the API documentation, which is installed with the product and is accessible from the Windows Start Menu in the ICONICS → GENESIS64 → Help group. The document is called *GENESIS64 API Reference Help*.

ICONICS Customer Connection Portal hosts a library of automation examples that are very handy when you start scripting.