

Description: Guide to create GraphWorX64 displays in an automated way using Visual Studio.

OS Requirement: Windows Server 2003 x64, Server 2008 x64, Server 2008 R2 x64, Windows Vista x64, Windows 7 x64

General Requirement: Installation of GENESIS64 and Visual Studio 2005/2008/2010. Good knowledge of a .NET language (i.e. C#, VB.NET).

Introduction

Using GraphWorX64 you can create compelling graphics to visualize your data. However that task may be repetitive and long to perform. GraphWorX64 offers many tools to speed up configuration time, but there may be cases when you want to automate the process.

There could be an instance when you have to generate several copies of a symbol but change only a few properties of it.

You might also have a set of information in a database or a file that describes the way your display should look like; automating the display creation would be a simple and efficient manner to speed up configuration time.

Two possible approaches

There are two ways to generate your display automatically. The first approach is by writing a Visual Studio application. The second approach is by using scripting code inside GraphWorX64 itself.

Although the code to write in both approaches is quite similar, Visual Studio is more oriented to generate large content for your displays, while GraphWorX64 scripting is ideal for quick object replication and style modifications.

In this document we will analyze in detail how to generate a display from Visual Studio using C#, and then introduce how the same task is performed in GraphWorX64 through scripting, using JScript.NET.

Using Visual Studio to create a display

In order to create our application, we start Visual Studio and create a new Project, with a WPF Application template in it. We will be using Visual Studio 2008 here, but you can use a different version.

While creating a new project, it is important we pick WPF Application and not Windows Forms as the Template. Give a name to your new project, and then click on OK to confirm your choice, the new application files are then created automatically.

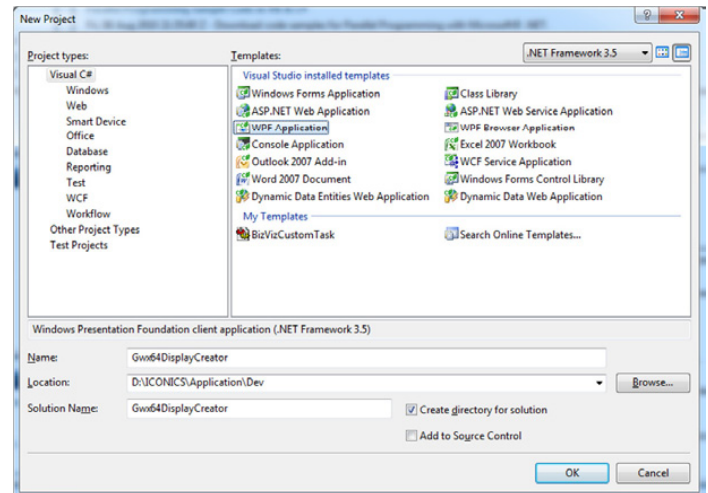


Figure 1 – Creating a new Visual Studio solution

Once the wizard completes creating the new files, you will be shown your XAML form. In this form we will add a button to trigger display creation and a GraphWorX64 Runtime View Control, which will host our generated display..

In your application you may want to set up a more complex user interface such as inserting some parameter, or selecting certain configuration files or databases. In this example, we do not create a complex user interface and only insert a simple button to trigger our code.

1. The first thing to do is to check the properties of the project and make sure the TargetFramework is set to **.NET Framework 4** instead of Client profile which is set by default.

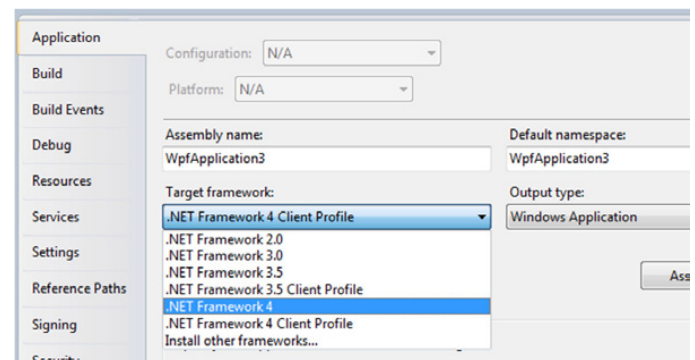


Figure 2 - Use .NET Framework 4

2. Next thing to do is to add some references to the GraphWorX64 libraries. To do this, right click on the References folder in your Solution Explorer and select "Add Reference..." (you can find the same command under the "Project" menu).

From the “Add Reference” dialog, pick the “Browse” tab and select the following files from **C:\Program Files\ICONICS\GENESIS64\Components**:

- GwxConfigCore.dll
- GwxRuntimeCore.dll
- GwxRuntimeViewControl.dll
- IcoCollections.dll
- IcoCommanding.dll
- IcoCommandManager.dll
- IcoComponentModel.dll
- IcoFwxClientWrapper.dll
- IcoFwxUaClient.dll
- IcoFwxUaCommon.dll
- IcoScriptEngine.dll
- IcoSecurityRuntime.dll
- IcoTrace.dll
- IcoWPF.dll
- IcoWPFTransitions.dll
- IcoXML.dll

Click on OK to confirm your choice.

3. Then from the same folder copy the following files into your project folder. In order to find the project folder you may right-click on **App.xaml** in your Solution Explorer and open it. Once open, right-click its tab and select Open Containing folder and copy the following files into your Project folder.

- IcoCommunication.Behaviors.config.xml
- IcoCommunication.Bindings.config.xml
- IcoCommunication.Client.config.xml
- IcoTypeDescription.xml

Add them to your project in Visual Studio. Right click on the project in Solution Explorer and select **Add -> Existing item...** Select the files that you copied and click on ADD.

Once added select them all in Solution Explorer and change their Properties. Set **Build Action** property to **None** and **Copy to Output Directory** property to **Copy Always**.

Modify your app.config file so as it contains the references to the three files we have added.

```
<?xml version="1.0"?>
<configuration>
```

```
<startup>
  <supportedRuntime version="v4.0"
sku=".NETFramework,Version=v4.0"/>
</startup>
<system.serviceModel>
  <behaviors
configSource="IcoCommunication.Behaviors.config.xml"/>
  <bindings
configSource="IcoCommunication.Bindings.config.xml" />
  <client
configSource="IcoCommunication.Client.config.xml"
/>
</system.serviceModel>
</configuration>
```

4. Now in order to insert the GraphWorX64 Runtime View Control open your Toolbox. Expand General section which is at the bottom, right click and select **Choose Items...**

Browse to the same folder as when adding references and select GwxRuntimeViewControl.dll. Click Open and OK.

Your Toolbox should now show the GwxRuntimeViewControl in the list.

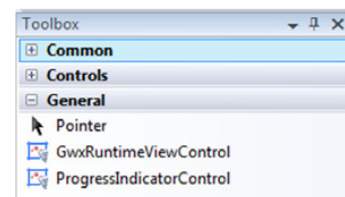


Figure 3 – Creating a new Visual Studio solution

5. Drag this control onto your form, and then, from the “Common” group in the Toolbox, click on button to add it to the application. Your application should now look more or less like this:

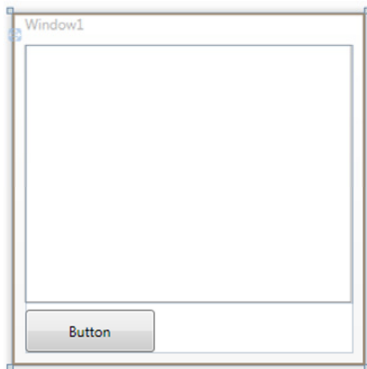


Figure 4 – The application form

6. We can now write the actual code that will populate the display. Double-click on your button to generate a handler for its Click event. This will also open the code editor for that handler.

Before adding objects to the runtime control we embedded into our application, we will define a property for our class, which will wrap the GraphWorX64 display's configuration object we will often use.

```
public GwxConfiguration Configuration
{
    get
    {
        return ctlRuntimeView.Configuration as
GwxConfiguration;
    }
}
```

```
GwxRuntimeViewControl ctlRuntimeView;
```

Where `ctlRuntimeView` is the name we gave to our runtime view control. If the `GwxConfiguration` class cannot be recognized automatically, you may need to add a "using" clause at the top of your file:

```
using Ico.Gwx;
```

Creating display

Now we are ready to start working with the display. The workflow you should follow is usually made up by three parts:

- Create a new GraphWorX64 display to work with;
- Fill it with your content;
- Save the file for later use.

1. Creating a new display file is pretty easy. All you need to do is to use the following command.

```
this.ctlRuntimeView = new GwxRuntimeViewControl();
this.ctlRuntimeView.FileNew();
```

NOTE: All the code we're writing in this section will go into the button's Click event handler we have created earlier by double-clicking on the button. "Using" clauses will go on top of the file instead.

2. After creating the new display, we will create our first object in it: a GraphWorX64 rectangle. The following code uses the `Create` method from the `GwxRectangle` class to create a new rectangle and add it to the specified configuration which belongs to our display.

```
GwxRectangle gwxRect =
GwxRectangle.Create(this.Configuration, new
Rect(10,10,80,60), this.Configuration.Root, false,
null);
```

Using the above parameters we create an 80 pixel wide and 60 pixel high rectangle, which will be placed at display coordinates (10,10).

NOTE: For further details about the other parameters, please refer to Visual Studio comprehensive Intellisense documentation or GENESIS64 API documentation.

We still have to assign a color to the rectangle, or it will not be visible. Let's paint it with red.

```
gwxRect.Fill = Brushes.Red;
```

3. We can now try to build our application and then run it. If you get any compile error, please verify you performed all the above steps correctly and then try again.

Once the build process completes successfully, you can launch your new application and click on the button to run the code we wrote to create the rectangle.

We have covered the first two steps of our workflow so far, we still have to implement the last step, to save the display somewhere, in order to use it. A quick way to do that is by using the following code.

```
this.ctlRuntimeView.SaveAs("c:\\test.gdfr");
```

If you add this code just after the creation of your rectangle the display is saved to a file of the specified name and path. Once it is saved, you will be able to open it from GraphWorX64.

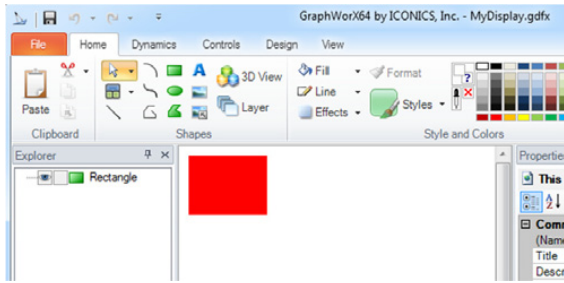


Figure 5 – The generated display opened from GraphWorX64

From GraphWorX64 you will be able to keep on editing your display, eventually adding the final touch before actually using it in your project.

At the end of this application note you can see the full code for your reference.

Generating real-life displays

The sample display we created in the previous section is just a basic example of how to generate a display through .NET code. Typically you want to use this feature when you have a large set of objects to create.

You may then take advantage of all the language constructs like “for” loops, to quickly replicate existing objects, or create new ones. It’s all a matter of writing the appropriate .NET code to gather the information needed, and then using the GraphWorX64 API methods and classes to generate the content.

Other possible improvements also include user interaction to define parameters which are typical to your application. For

example you may want your user to select a color scheme to be used to generate your objects, or to specify the number of objects to be created, or the database connection to read objects configuration (base symbol name, position, data sources ...).

Not only can you create new displays, but also open existing ones and work with them, to modify their content, or to copy that content into a new file, or another existing one.

Using GraphWorX64 to create a display

Another possibility to generate a display is by using JScript.NET code inside a GraphWorX64 display. To perform this task you will need to write code which is almost identical to the one we have seen so far (i.e. to create a rectangle) and associate it to some object which will trigger it (i.e. a button).

See our Customer Connection Portal for more examples of scripting as well as automatic generation of displays.

Creating a custom application

All we have explained in this document can also be used if you intend to embed GraphWorX64 Runtime View Control into your own application, and take full control of it, by dynamically altering its content depending on the logic of your application.

In such case, you probably won’t need to export the generated display as gdxf file, unless you need to save your application status.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Ico.Gwx;

namespace WpfApplication1
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public GwxConfiguration Configuration
        {
            get
            {
                return ctlRuntimeView.Configuration as GwxConfiguration;
            }
        }

        GwxRuntimeViewControl ctlRuntimeView;

        public MainWindow()
        {
            InitializeComponent();

            private void button1_Click(object sender, RoutedEventArgs e)
            {
                this.ctlRuntimeView = new GwxRuntimeViewControl();
                this.ctlRuntimeView.FileNew();
                GwxRectangle gwxRect = GwxRectangle.Create(this.Configuration, new Rect(10, 10, 80, 60),
this.Configuration.Root, false, null);
                gwxRect.Fill = Brushes.Red;

                this.ctlRuntimeView.SaveAs("c:\\test.gdpx");
            }
        }
    }
}
```