



GENESIS64 – Custom Server Methods in Platform Services



APPLICATION NOTE

August 2014

Description: An example of using custom server methods (commands) in Platform Services and how to call the custom commands in GraphWorX64.

General Requirement: Installation Microsoft Visual Studio 2010. General knowledge of Commanding.

Introduction

If you are new to Commanding please take a look at the following application notes:

- GENESIS64 - Commanding Quick Start
- GENESIS64 - Commanding Configuration and Runtime

This Application Note will show you how to create your own custom commands from within Platform Services. These custom commands can be executed from any ICONICS application that supports commands, such as GraphWorX64, PortalWorX-SL, and MobileHMI.

There are two examples demonstrated in this application note. The first method accepts no parameters and writes the current date and time to a text file. The second method shows you how to pass in parameters, perform the multiplication of two integers, and return their product back as an output.

Creating a Custom Command Method

Before you can call a Custom Command from Platform Services, you must create your own custom library and expose the method. We will use Microsoft Visual Studio 2010 and Visual C# for creating our examples.

To create a custom method and expose it to the Asset Catalog:

1. Open Microsoft Visual Studio 2010.
2. Create a new **C# Class Library** project.

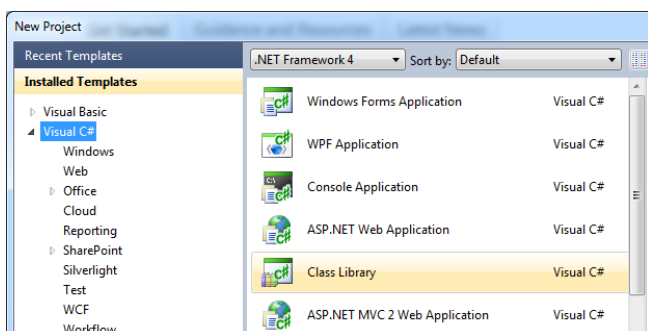


Figure 1 - C# Class Library

NOTE: Keep in mind that this is a Class Library so you will not be able to use system messages, user interface, or forms in custom command methods.

3. Reference **IcoAssetCatalogMethods.dll** which is located in C:\Program Files\ICONICS\GENESIS64\Components by default.

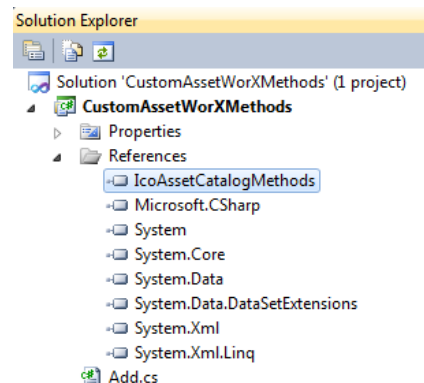


Figure 2 - Referenced IcoAssetsCatalogMethods.dll

4. Create a Custom Method code with the options below:

NOTE: Complete code can be found at the end of this Application Note.

- o Derive a new class from **Ico.AssetCatalog.Methods.CustomMethod**
- o Implement abstract methods: **GetInputArguments()**, **GetOutputArguments()**, **Execute()**

NOTE: These three methods are required, but they do not need to contain any parameters.

- o Optionally override: **BrowseMode()** and **IsBrowseAllowed()** to let the method appear in the data browser
- o Optionally override: **GetFullName()**, **GetShortName()** to modify the method names (advanced override)

5. Build the DLL.

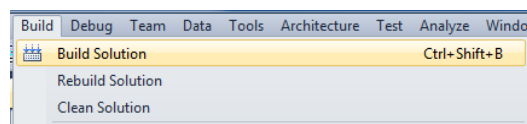


Figure 3 - Building the DLL

6. Copy it into the **Components\AssetCatalogMethods** folder along with the IcoAssetCatalogMethods.dll



Using Custom Commands in GraphWorX64

There are two ways you can call your Custom Command in GraphWorX64: as a **Pick Action** or from a **Jscript.NET Script**

NOTE: In version 10.8 and earlier, the call of custom method with parameters is available only from a Jscript.NET Script. The Pick Action currently doesn't have the option to leverage parameters, but is planned for a future release.

Pick Action

For the first example, we will use a Pick Action to call the custom method that creates a text file with the current date and time.

1. Add a button in GraphWorX64.
2. Select the **Call Method** Pick Action Command.

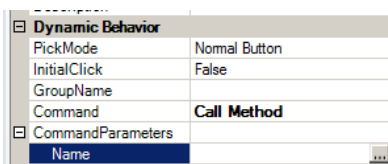


Figure 4 - Call Method Pick Action Command

3. Browse for the Custom Command.

NOTE: To see Methods in Browser, you have to enable **Expert mode** and check **Methods** option in **Filtering**

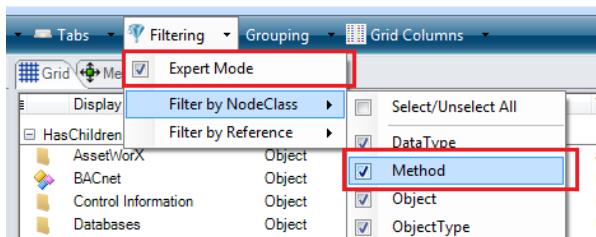


Figure 5 - Enabling support to view Methods in Data Browser

4. The Custom Method will be available under the **AssetWorX** node on the OPC UA tab of the Data Browser.

NOTE: Whether or not the Custom Method is visible is dependent on **IsBrowseAllowed()** and **BrowseMode()** definitions in the Custom Method.

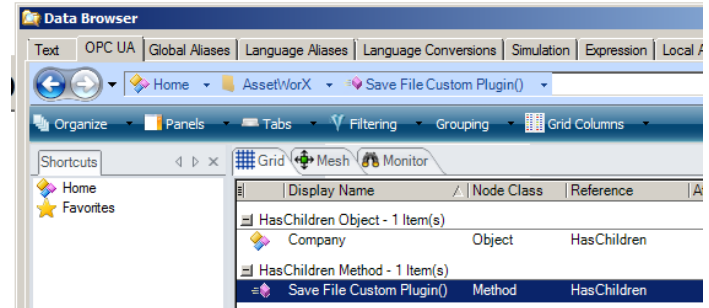


Figure 6 - Browsing for Custom Method under AssetWorX node

5. Go into runtime on your GraphWorX64 Display and click the Button to execute the Call Method Pick Action.
6. Check the result of your custom command method and verify a text file was created with the latest date and time inside.

Jscript.NET Script

In Jscript from GWX we are calling the second example with input and output parameters.

1. Add Button in GWX64.
2. As a Command for Pick Action select **Run Script**
3. Create a new script and add following code for calling MethodCallAsynch function:



```
function Pick1_CommandExecuted(sender : System.Object, cmdArgs :
Ico.Gwx.CommandExecutionEventArgs)
{
    //inputParameters
    var customObject : Object[] = [2, 5];

    //call Asynchronous Method, result from MethodCallDoneDelegate
    ThisWindow.FwxClientWrapper.MethodCallAsync("ac:",
"SampleCustomMethods.Multiply", customObject,
ThisDocument.CreateDelegate(MethodCallDoneDelegate, this,
"MethodComplete"), null);
}
```

4. Add one more function for delegate result, which provide us output parameters

```
function MethodComplete(result : MethodCallDoneResult)
{
    // outputParameter string
    var outputData : String;

    // returns all output parameters
    var i : int;
    for (i = 0; i < result.OutputArguments.Count; i++)
    {
        outputData += result.OutputArguments[i].ToString() + "\r\n";
    }
    MessageBox.Show("Result is: " + result.Status + "\r\n" + "Method
output: " + outputData , "Custom Method Called");
}
```

5. Go into runtime and run Jscript
6. Check the result of your custom command method

Example Custom Methods

Example 1

- creating a text file and insert current date and time

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CustomAssetWorXMethods
{
    public class CreateTextFileWithDateTime : Ico.AssetCatalog.Methods.CustomMethod
    {
        #region Metadata

        public override Ico.AssetCatalog.Methods.ArgumentInfo[] GetInputArguments()
        {
            return new Ico.AssetCatalog.Methods.ArgumentInfo[]
            {
            };
        }

        public override Ico.AssetCatalog.Methods.ArgumentInfo[] GetOutputArguments()
        {
            return new Ico.AssetCatalog.Methods.ArgumentInfo[]
            {
            };
        }

        #endregion

        #region Execute

        public override bool Execute(string pointName, string[] path, bool isLeaf, object[] inputArguments, object[] outputArguments)
        {
            System.IO.File.WriteAllText(@"C:\TestFile.txt", DateTime.Now.ToString());
            return true;
        }

        #endregion

        #region Name

        protected override string GetFullName()
        {
            return "Save File Custom Plugin";
        }

        protected override string GetShortName()
```



```
{
    return "Save File";
}

#endregion

#region Browsing

public override bool IsBrowseAllowed(string pointName, string[] path, bool isLeaf)
{
    // allow methods on all assets
    return true;

    // allow methods only on leaves
    //return isLeaf;
}

public override Ico.AssetCatalog.Methods.MethodBrowseMode BrowseMode
{
    get
    {
        return Ico.AssetCatalog.Methods.MethodBrowseMode.Always;
    }
}

#endregion
}
```

Example 2

- 2 inputs parameter of integer type and one output which is a multiplication of these two input numbers

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Ico.AssetCatalog.Methods;
using System.Diagnostics;

namespace SampleCustomMethods
{
    public class Multiply : CustomMethod
    {
        #region Metadata

        public override ArgumentInfo[] GetInputArguments()
        {
            return new ArgumentInfo[]
            {
                new ArgumentInfo("x", typeof(int)),
                new ArgumentInfo("y", typeof(int))
            };
        }

        public override ArgumentInfo[] GetOutputArguments()
        {
            return new ArgumentInfo[]
            {
                new ArgumentInfo("result", typeof(int))
            };
        }

        #endregion

        #region Execute

        public override bool Execute(string pointName, string[] path, bool isLeaf, object[] inputArguments, object[] outputArguments)
        {
            Debug.Assert(inputArguments.Length == 2);
            Debug.Assert(inputArguments[0] is int);
        }
    }
}
```



GENESIS64 – Custom Server Methods in Platform Services



APPLICATION NOTE

August 2014

```
        Debug.Assert(inputArguments[1] is int);

        int result = (int)inputArguments[0] * (int)inputArguments[1];

        Debug.Assert(outputArguments.Length == 1);
        outputArguments[0] = result;

        return true;
    }

#endregion

#region Browsing

public override MethodBrowseMode BrowseMode
{
    get
    {
        return MethodBrowseMode.Callback;
    }
}

public override bool IsBrowseAllowed(string pointName, string[] path, bool isLeaf)
{
    // allow methods on all assets
    return true;
}

#endregion
}
```