Tech Note 537
Creating an Application Object Script Using GRAccess

Topic#: 002274
Created: April 2008

Introduction

The GRAccess toolkit enables developers to automate activities that users normally perform manually using the Industrial Application Server Integrated Development Environment (IDE).

This *Tech Note* describes how to create and configure a DataChange Script using a C# console application.

Application Versions

To execute the GRAccess sample application that is described in this document, you will need the following prerequisites:

- Visual Studio 2005
- Industrial Application Server 2.1 or later

Using the IDE to Create and Configure a DataChange Script

Before automating IDE tasks using GRAccess, we describe the manual IDE configuration steps that are going to be automated.

Create the Derived Object Template

1. Create an object instance of the **$UserDefined** template. Give the new object the name **UserDefined_001** and open the object editor.
2. In the **UDAs** tab panel add a new Boolean User Defined Attribute called **Switch** that will be used later as

the trigger of the DataChange script.

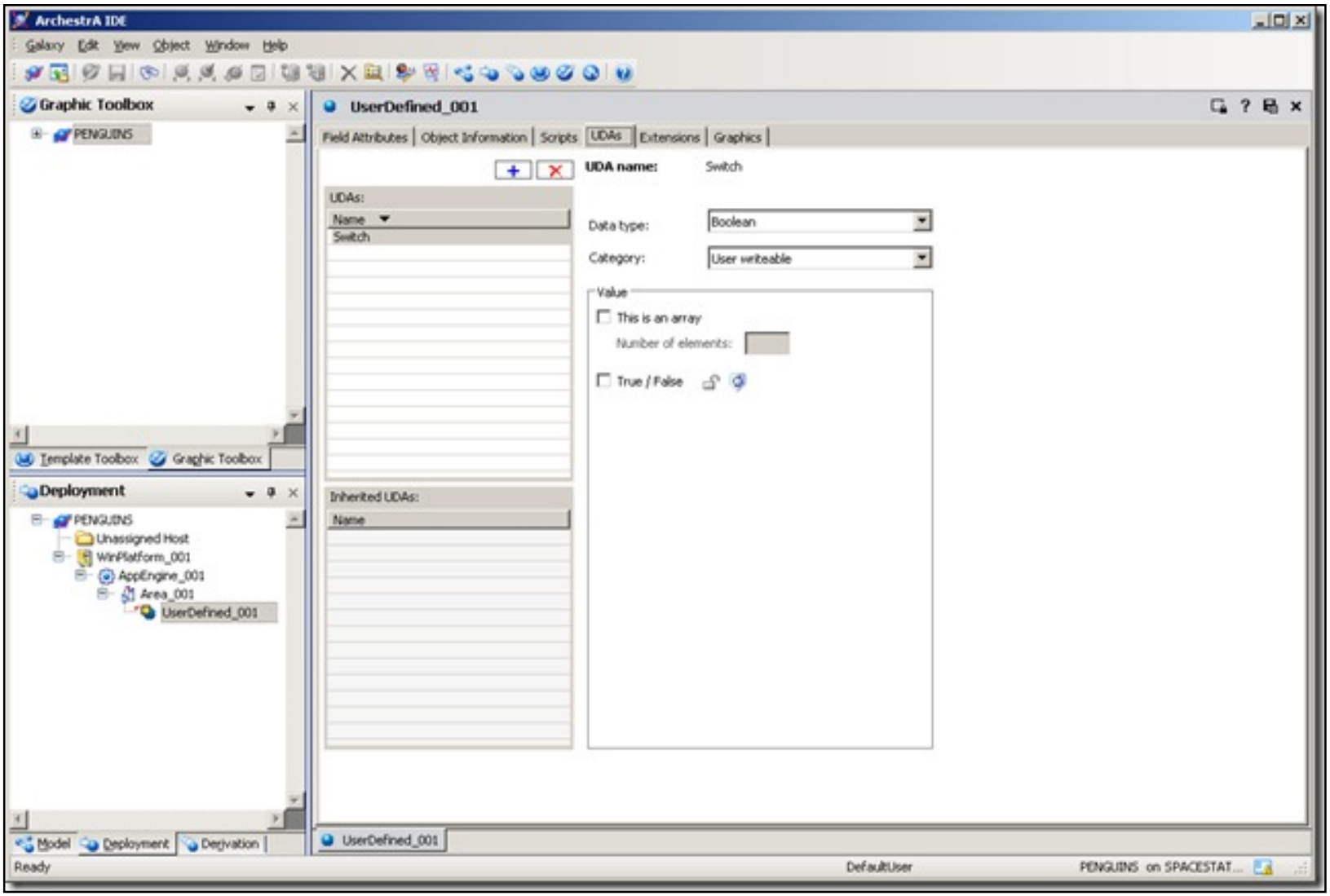

Figure 1: Add Boolean UDAs

3. Click the **Scripts** tab.

Configure the DataChange script

1. First declare a variable in the "Declarations" section. This variable will be in scope during the whole Runtime of the object.

Use the following statement for the declaration:

```
Dim Number as Integer;
```

2. In the Scripts panel, add a new script and name it **HelloWorld**.
3. In the **Expression** field type **me.Switch** and select **DataChange** as the Trigger Type.
4. Type or copy/paste the following script into the editor :

```
LogMessage("Hello World!");
LogMessage("Number of ScanState changes: " + StringFromIntg(Number, 10));
Number = Number + 1;
```

The following grahic shows the script settings in the editor:

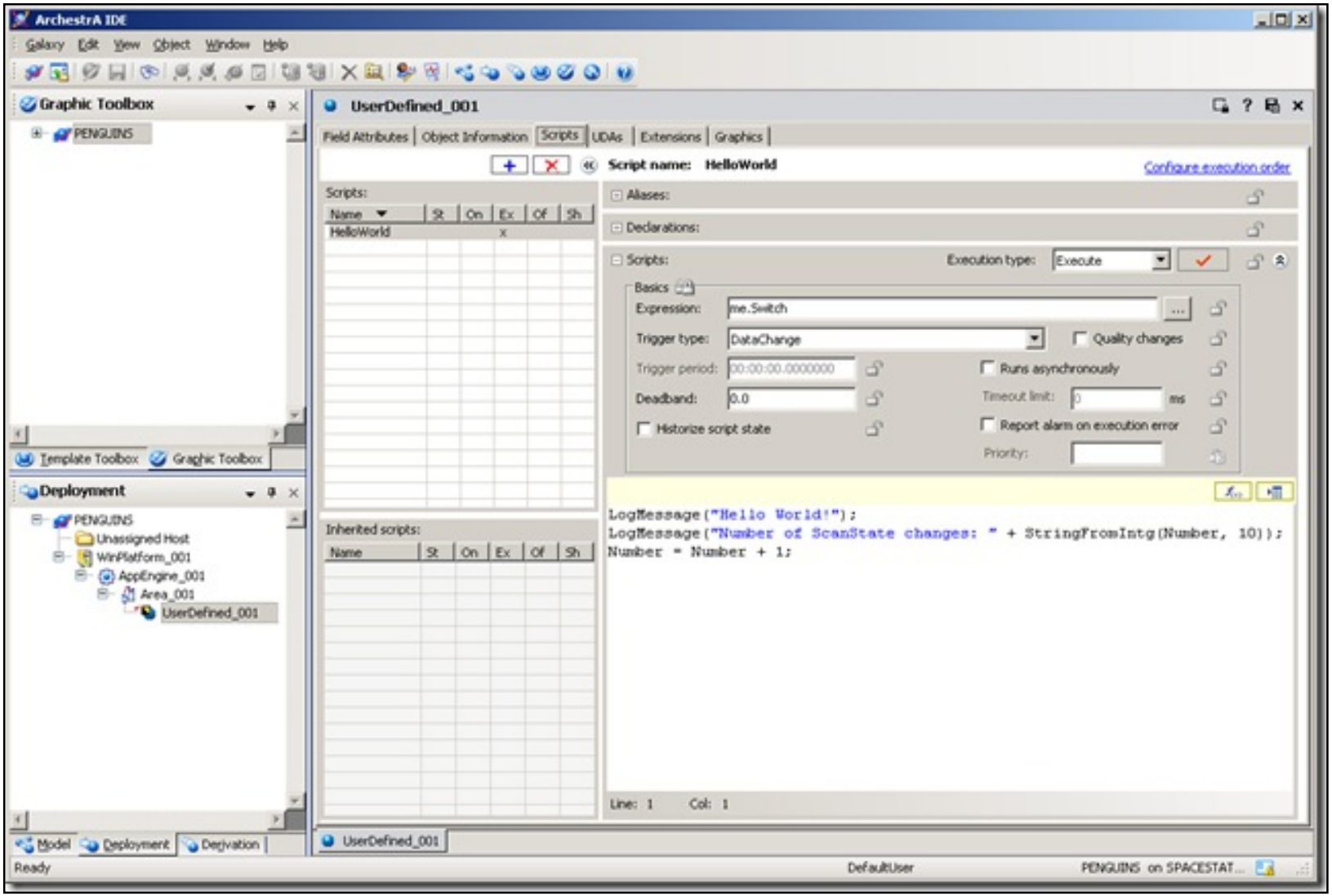

Figure 2: Script Settings for "Hello World"

What Happens at the API Level

This section describes in GRAccess API terms what happened under the hood during the configuration of the **UserDefined_001** object:

Create the Object Instance

Query the Galaxies and Login

```
Galaxies = GR.QueryGalaxies(GRMachineName);
G = Galaxies[GalaxyName];
G.Login(UserName, Password);
```

Get the $UserDefined Template and Create a New Object Instance

```
// Find $UserDefined template
Objects = G.QueryObjectsByName(EgObjectIsTemplateOrInstance.gObjectIsTemplate, ref Names);
T = (ITemplate)Objects[1];
// Create instance of $UserDefined
ObjectInstance = T.CreateInstance(ObjectName, true);
```

Checkout the Object

```
ObjectInstance.CheckOut();
```

Add New User Defined Attribute

```
// Add UDA
ObjectInstance.AddUDA(TriggerUDAName,

      MxDataType.MxBoolean,
      MxAttributeCategory.MxCategoryWriteable_USC_Lockable,
      MxSecurityClassification.MxSecurityFreeAccess, false, null);
```

Save the Object

This step is important. Saving the object at this point generates the new attribute and makes it accessible to the configuration steps that follow.

```
ObjectInstance.Save();
```

Create the DataChange script

At this point we are ready to add and configure the script (Script Extension Primitive).

First we add a new script into the list of scripts. Then we save the object. Saving the object generates all the attributes required to configure the script the conventional way. Here is the list of the configurable attributes of the ScriptExtension primitive:

```
ExecuteText
AliasReferences
Aliases
TriggerType
DataChangeDeadband
Expression
DeclarationsText
StartupText
ShutdownText
OnScanText
OffScanText
ExecutionError.Alarmed
TriggerPeriod
ScriptExecutionGroup
ScriptOrder
RunsAsync
State.Historized
ExecuteTimeout.Limit
TriggerOnQualityChange (New in 3.0)
```

Add the Script Extension Primitive

```
ObjectInstance.AddExtensionPrimitive("ScriptExtension", DataChangeScriptName, true);
```

Save the Object and Generate Script Attributes

```
ObjectInstance.Save();
```

Configure the Script Extension Attributes

```
IAttribute ScriptBodyTextAttribute = ObjectInstance.ConfigurableAttributes[DataChangeScriptName + ".
ExecuteText"];
IAttribute ScriptDeclarationsTextAttribute = ObjectInstance.ConfigurableAttributes
[DataChangeScriptName + ".DeclarationsText"];
IAttribute ScriptTriggerTypeAttribute = ObjectInstance.ConfigurableAttributes[DataChangeScriptName + ".
TriggerType"];
IAttribute ScriptExpessionAttribute = ObjectInstance.ConfigurableAttributes[DataChangeScriptName + ".
```

```
Expression"];

MxValue v = new MxValueClass();

//Setting the script delarations text
v.PutString("dim " + TriggerCounterVariableName + " as Integer;");
ScriptDeclarationsTextAttribute.SetValue(v);

//Setting the script type
v.PutString("DataChange");
ScriptTriggerTypeAttribute.SetValue(v);

//Setting the script Expression
v.PutString("me." + TriggerUDAName);
ScriptExpessionAttribute.SetValue(v);

//Setting the scripts execute text
v.PutString("LogMessage(\"Hello World!\");\n" +
"LogMessage(\"Number of ScanState changes: \" + StringFromIntg("+ TriggerCounterVariableName + ", 10));
\n" + TriggerCounterVariableName + " = " + TriggerCounterVariableName + " + 1;");
ScriptBodyTextAttribute.SetValue(v);
```

The following screen shots show the Script Extension attributes provided by the code.

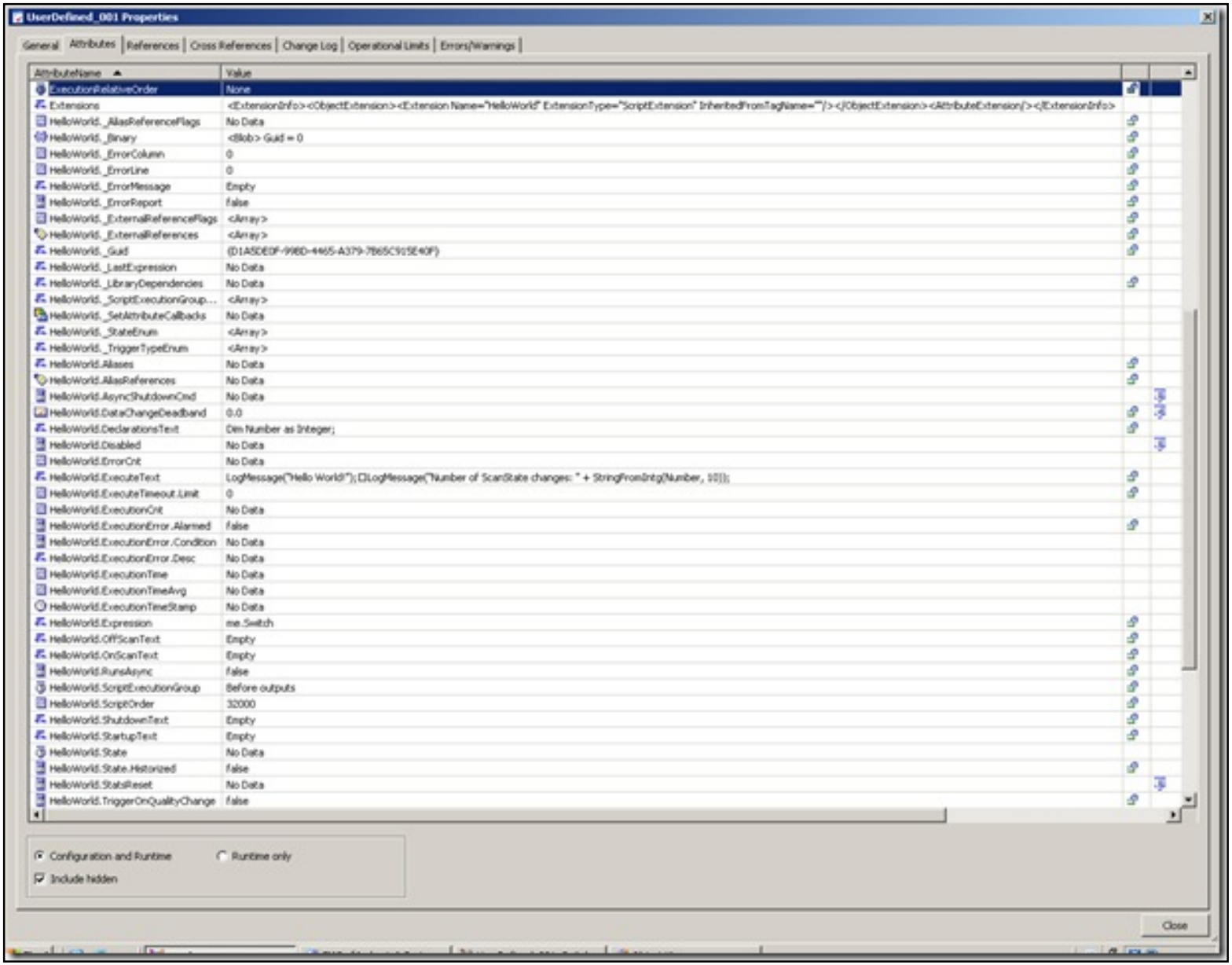

Figure 3: Script Extension Attributes

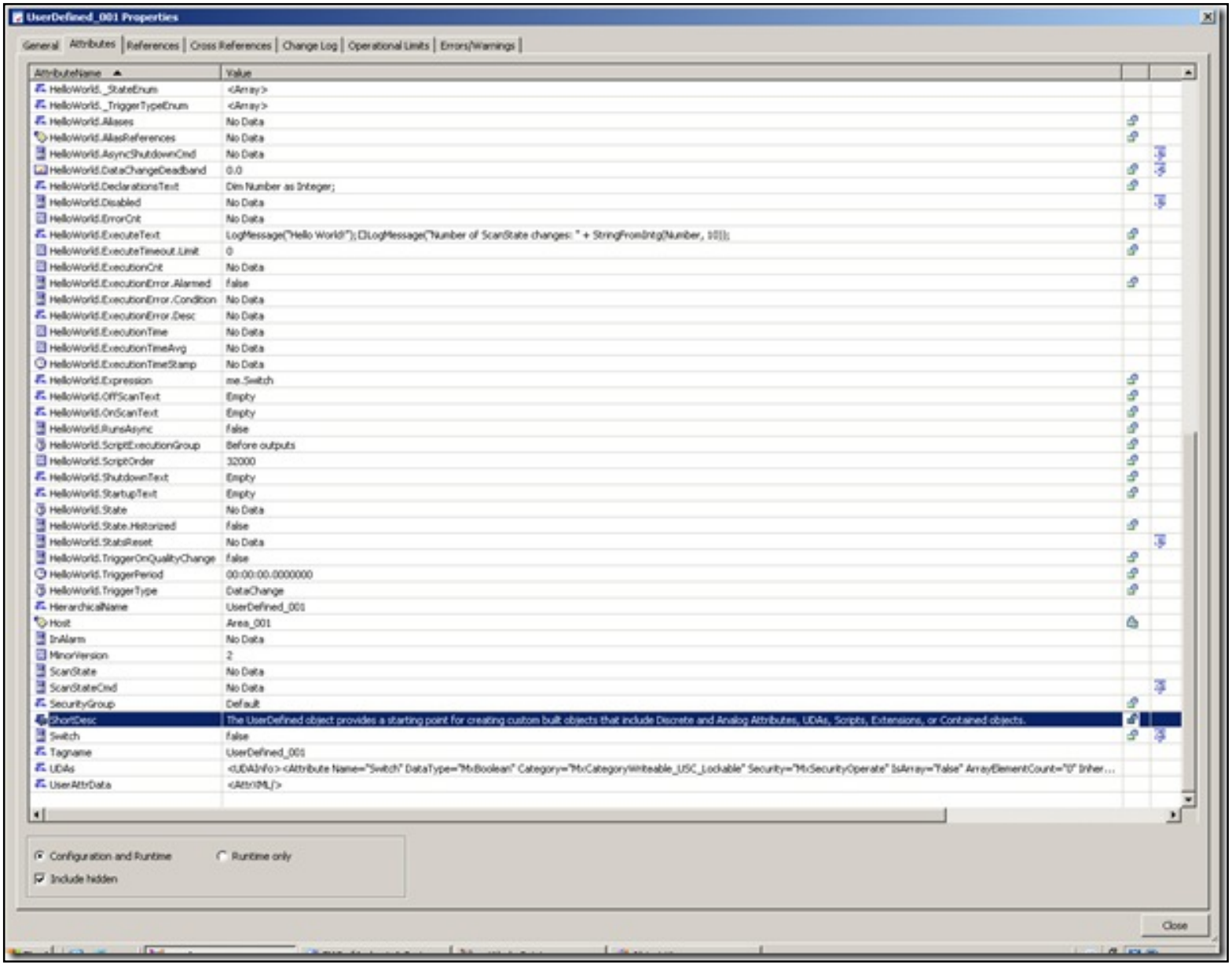

Figure 4: Script Extension Attributes

Save the Object and Check It In

```
ObjectInstance.Save();
ObjectInstance.CheckIn("");
```

The Complete Source File

View the complete program file. You can copy/paste the content to a .cs file at your convenience.

Compilation

To compile the program.cs file run the following command in the Visual Studio 2005 command prompt:

```
csc /out:CreateScript.exe program.cs /r:"C:\Program Files\Common Files\ArchestrA\ArchestrA.GRAccess.dll"
```

Test Run

To execute CreateScript.exe run the following command:

```
CreateScript gr=localhost g=PENGUINS u=Administrator pw=ww on=UserDefined_001 dc=Switch
sn=HelloWorld vn=Number
```

Summary

Creating a script using GRAccess is a two-step process.

- First the Script Extension Primitive needs to be added to the object using the **AddExtension()** method. Saving the object generates the attributes that are required to configure the script.
- Setting these Script Extension Attributes in the usual way is the second step.

K. Graefensteiner

Tech Notes are published occasionally by Wonderware Technical Support. Publisher: Invensys Systems, Inc., 26561 Rancho Parkway South, Lake Forest, CA 92630. There is also technical information on our software products at **www.wonderware.com/support/mmi**

For technical support questions, send an e-mail to **support@wonderware.com**.

 **Back to top**