Tech Note 528
Automating Object Configuration Tasks Using GRAccess: Creating and Configuring Field Attributes

Topic#: 002265
Created: March 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 FieldAttribute of a $UserDefined template 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 Add FieldAttributes

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

1. First create a derived template of the $UserDefined template.
2. Give the new derived template the name **$UserDefined_021** and open the template editor.

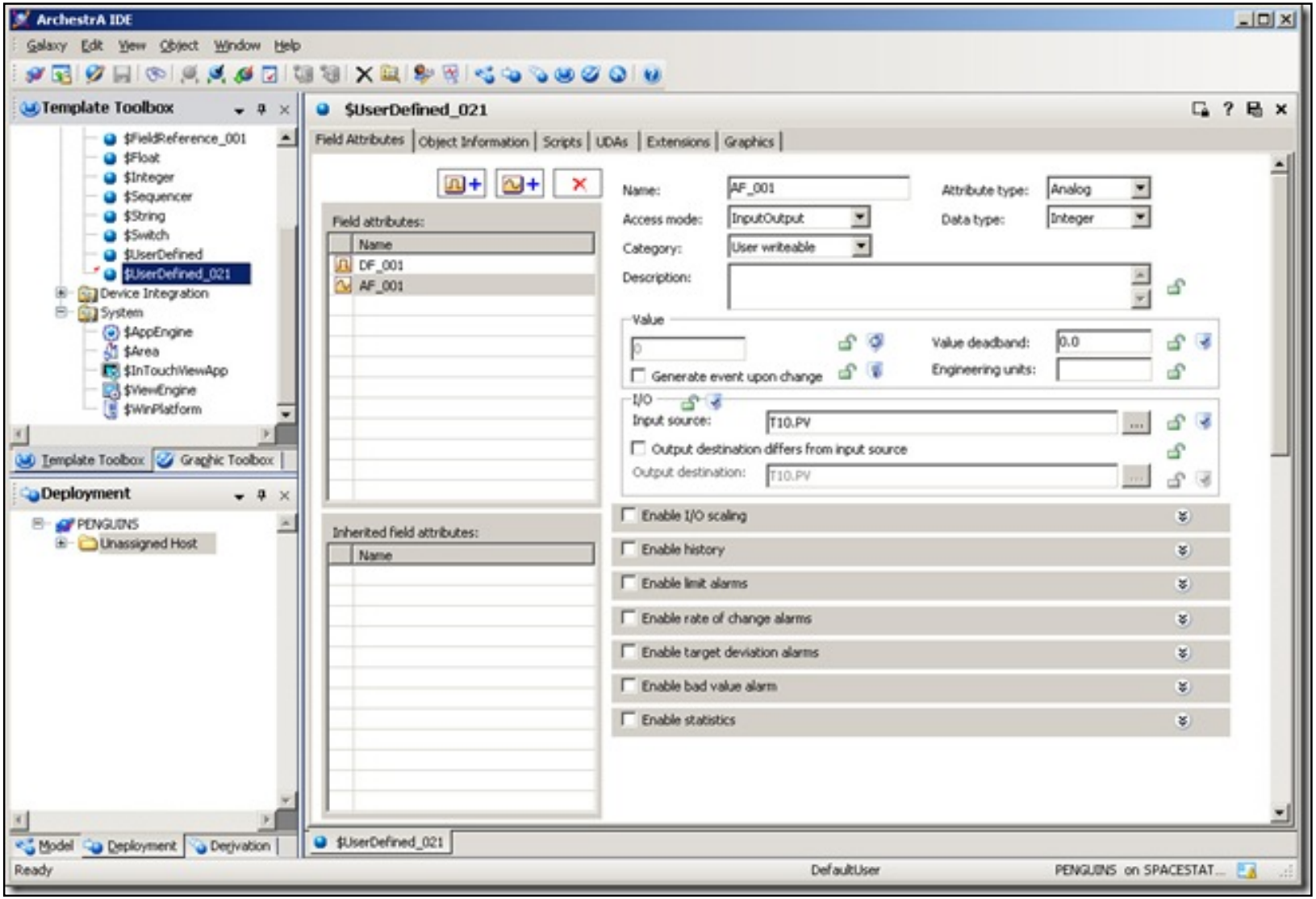

Figure 1: $UserDefined_021 Derived Template

3. Click the **Field Attributes** tab and add a new Analog Field Attribute called **AF_001** and a new Discrete Field Attribute called **DF_001** to the Field Attributes list.

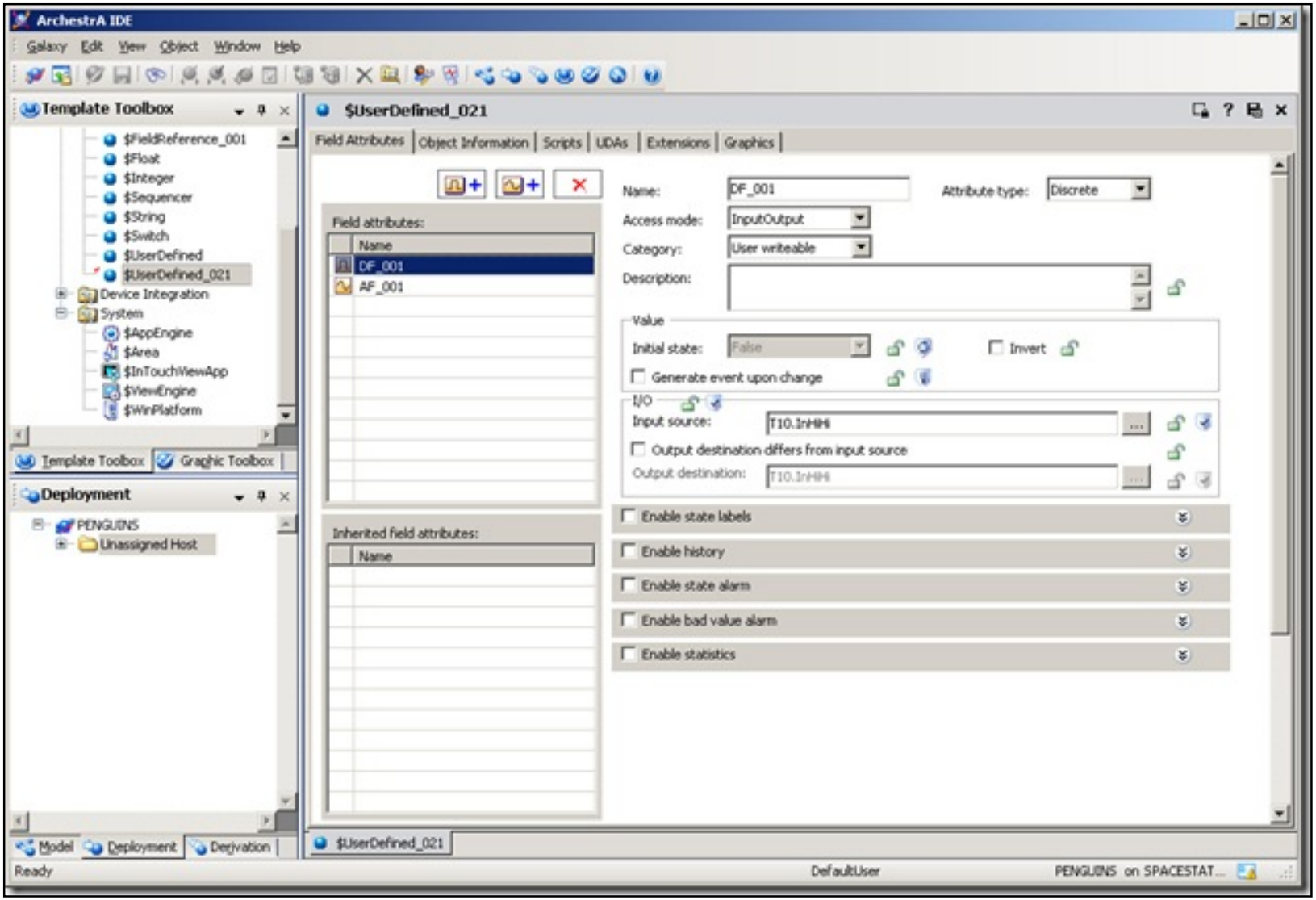

Figure 2: Analog and Discrete Field Attributes

4. Change the **Input source** attribute of the new Analog Field Attribute AF_001 to **T10.PV** and the Input source property of the new Discrete Field Attribute **DF_001** to **T10.InHiHi**.

5. Save and close the object.

What Happens at the API Level

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

Creating the Derived Template

Query the Galaxies and Login

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

Get the $UserDefined Template and Derive a New Template

```
string[] Names = { "$UserDefined" };
Objects = G.QueryObjectsByName(EgObjectIsTemplateOrInstance.gObjectIsTemplate, ref Names);
T = (ITemplate)Objects[1];
Parent = T.CreateTemplate(UDOTemplateName, true);
```

Checkout Template

```
Parent.CheckOut();
```

Adding New Field Attributes

This task is the key to successfully applying the **GRAccess API** in this case.

Knowing that the list of Field Attributes gets maintained by a XML string in the Attribute "UserAttrData" is crucial. Adding a new Analog Field Attribute can be accomplished by inserting a **<AnalogAttr>** element into the XML string of the **UserAttrData** attribute. The "name" XML attribute of the **<AnalogAttr>** element gives the new Field Attribute its name.

Creating a Discrete Field Attribute works the same way. First insert a **<DiscreteAttr>** XML element and set the name XML attribute equal to the name that you would like the Field Attribute to be called.

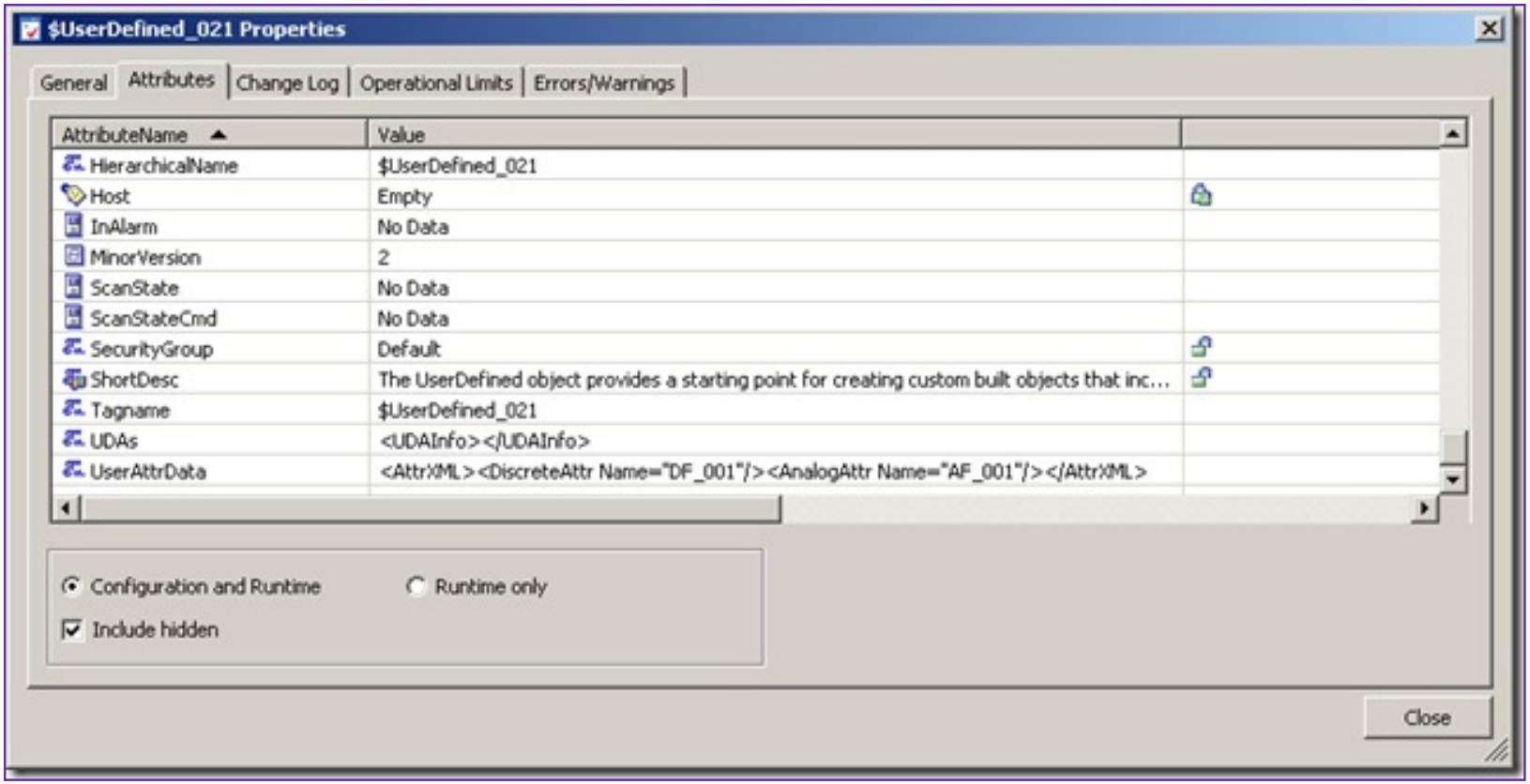

Figure 3: $UserDefined_021 XML Elements

```
IAttributes UDOAttributes = Parent.ConfigurableAttributes;
IAttribute UDOUserAttrDataAttribute = UDOAttributes["UserAttrData"];
IAttribute DiscreteFieldAttribute;
IAttribute AnalogFieldAttribute;
MxValue MxVal = new MxValueClass();
MxVal.PutString("<AttrXML><DiscreteAttr Name=\"" + DiscreteFieldAttributeName + "\"/><AnalogAttr Name=
\"" + AnalogFieldAttributeName + "\"/></AttrXML>");
UDOUserAttrDataAttribute.SetValue(MxVal);
```

Save the Template

This step is also very important. Saving the object at this point processes the XML string of the **UserAttrData** attribute and generates new attributes representing the Field Attributes that we just added.

```
Parent.Save();
```

Configure the Input Source properties

At this point we are able to configure the Field Attribute the conventional way.

Set the Input Source of the Discrete Field Attribute

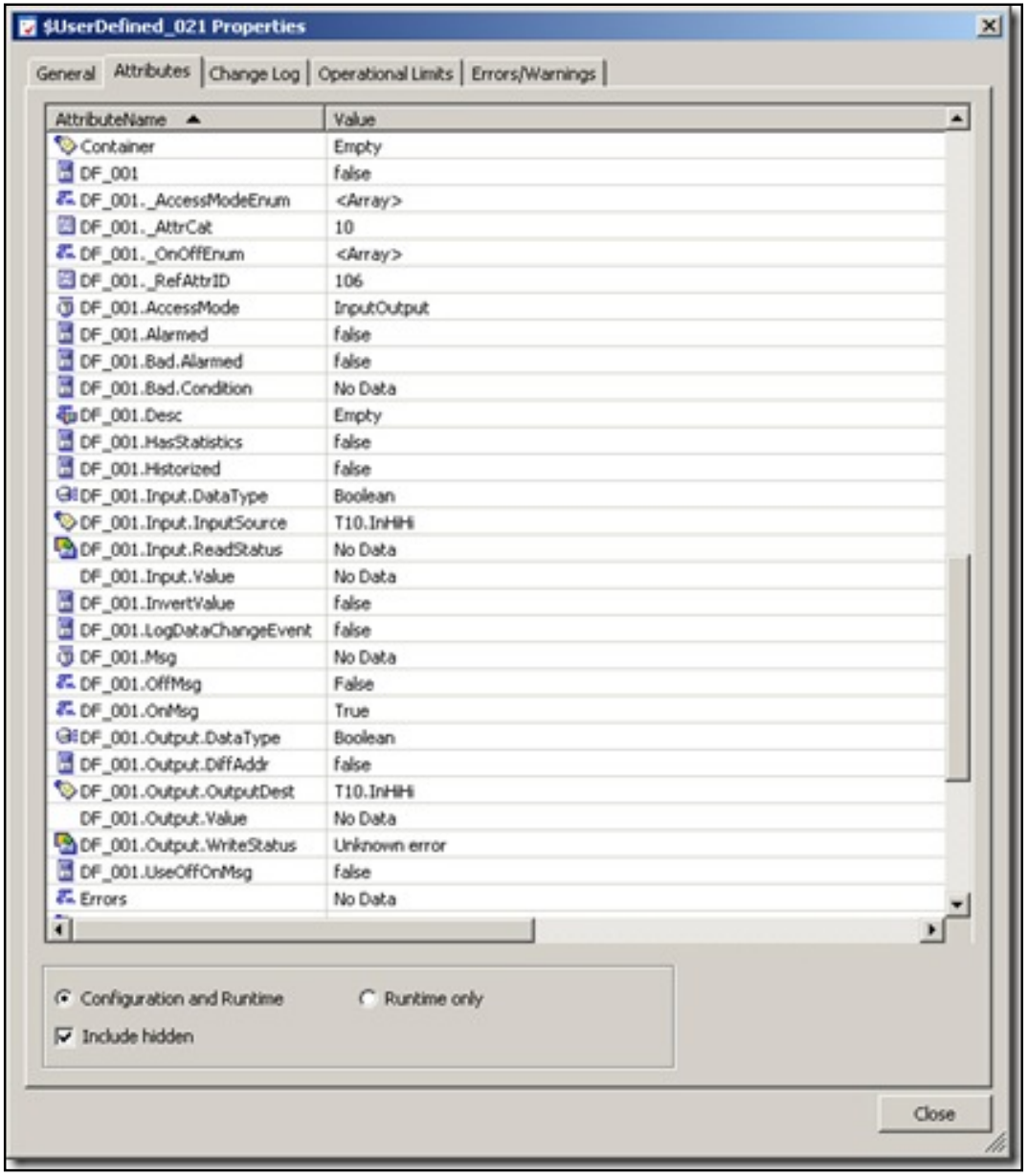

Figure 4: Discrete Field Attribute Input Source

```
//Now configure attributes as usual
UDOAttributes = Parent.ConfigurableAttributes;
DiscreteFieldAttribute = UDOAttributes[DiscreteFieldAttributeName + ".Input.InputSource"];
IMxReference MXRef;
MXRef = DiscreteFieldAttribute.value.GetMxReference();
MXRef.FullReferenceString = DiscreteFieldAttributeReferenceName;
MxVal.PutMxReference(MXRef);
DiscreteFieldAttribute.SetValue(MxVal);
Parent.Save();
```

Set the Input Source of the Analog Field Attribute

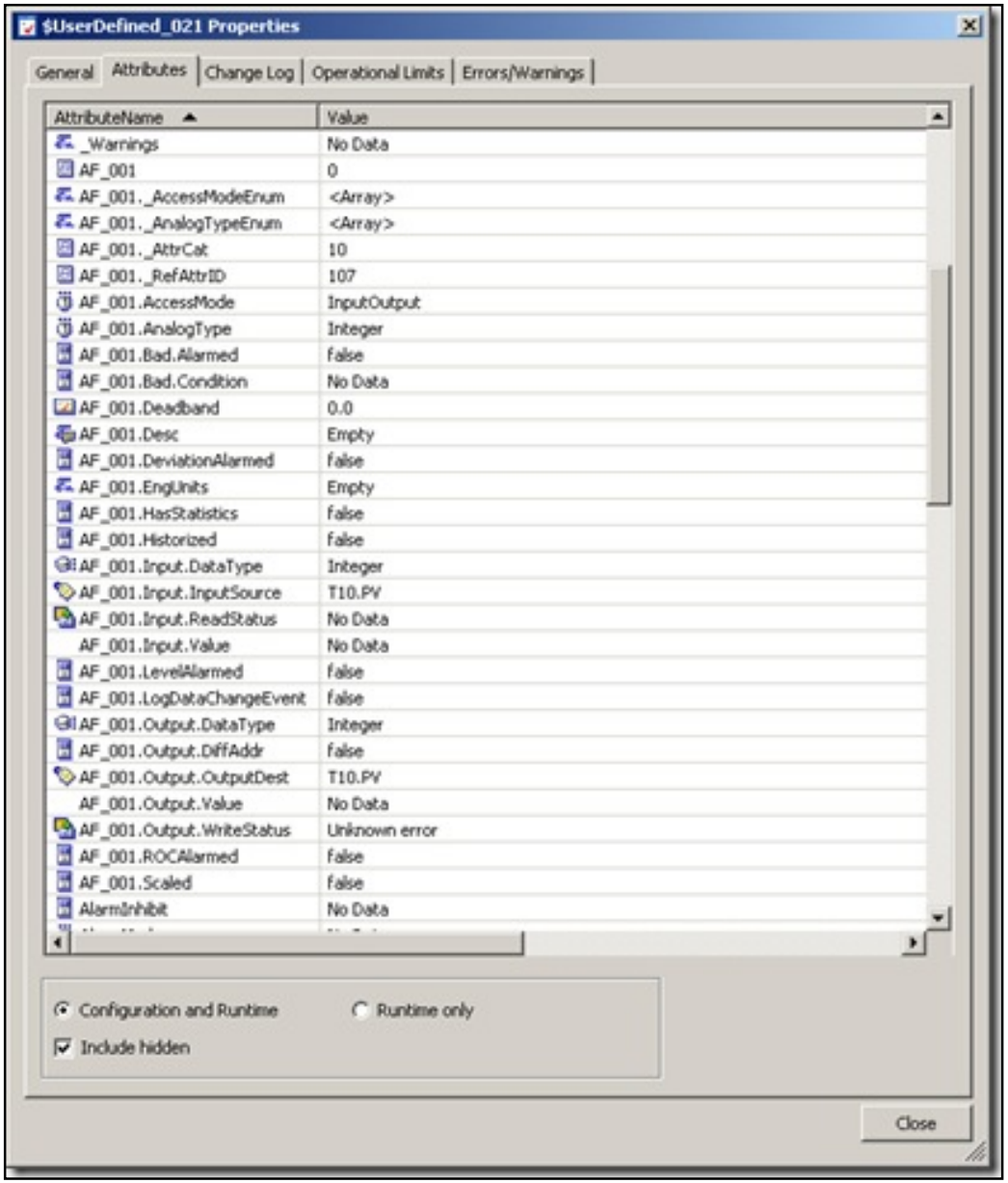

Figure 5: Analog FieldAttribute Input Source and Properties

```
//Now configure attributes as usual
UDOAttributes = Parent.ConfigurableAttributes;
AnalogFieldAttribute = UDOAttributes[AnalogFieldAttributeName + ".Input.InputSource"];
MXRef = AnalogFieldAttribute.value.GetMxReference();
MXRef.FullReferenceString = AnalogFieldAttributeReferenceName;
MxVal.PutMxReference(MXRef);
```

```
AnalogFieldAttribute.SetValue(MxVal);
Parent.Save();
```

Save and CheckIn

```
Parent.Save();
Parent.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:CreateFieldAttributes.exe program.cs /r:"C:\Program Files\Common Files\ArchestrA\ArchestrA.
GRAccess.dll"
```

Test Run

To execute **CreateFieldAttributes.exe** run the following command. In the following command line PENGUINS is the Galaxy name. Change it to reflect your server name:

```
CreateFieldAttributes gr=localhost g=PENGUINS u=Administrator pw=ww af=AF_001 df=DF_001 ar=T10.PV
dr=T10.InHiHi
```

Summary

- Knowing that the list of Field Attributes gets maintained by a XML string in the Attribute "UserAttrData" is crucial.
- Adding a new Analog Field Attribute can be accomplished by inserting a **<AnalogAttr>** element into the XML string of the **UserAttrData** attribute. The "name" XML attribute of the **<AnalogAttr>** element gives the new Field Attribute its name. Creating a Discrete Field Attribute works the same way.
- First insert a **<DiscreteAttr>** XML element and set the name XML attribute equal to the name that you would like the

Field Attribute to be called.

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**