Tech Note 432

Industrial Application Server Script Execution Types and Their Uses

Topic#: 002120
Created: June 2005

Introduction

This Tech Note outlines the proper usage of the 5 script execution types within Industrial Application Server 2.1 and older.

Script Execution Types

Script execution types and when they should be used are as follows:

Startup

Called when the object is loaded into memory (deployment, platform or engine start). This script method is intended to be used to instantiate COM objects and .NET objects. Depending on load and other factors, it may be possible that sets to object attributes from this script method may fail. Attributes that reside off-object are not available to this script method. Therefore it is not recommended to use this script method for any scripting beyond its intended use.

******Also see deployment considerations note.

OnScan

Called the first time an AppEngine calls this object to execute after the object scan state is changed to OnScan. This script method is intended to be used to initiate local object attribute values, or to provide more flexibility in the creation of .NET or COM objects. Attributes that are off-engine are not available to this script method. It is not recommended to use this script method for any scripting beyond its intended use.

******Also see deployment considerations note.

Execute

Called every time the AppEngine performs a scan and the object is OnScan.

The execute script method is the workhorse of the scripting methods. This is the place that runtime scripting should be done to ensure that all attributes and values are available to the script. This script method in analogous to the the InTouch scripts with the following conditional trigger types:

- **On True**: Executes if the expression validates from a false on one scan to a true on the next scan.[1]
- **On False**: Executes if the expression validates from a true on one scan to a false on the next scan.[1]
- **While True**: Executes scan to scan as long as the expression validates as true at the beginning of the scan.[1,2]
- **While False**: Executes scan to scan as long as the expression validates as false at the beginning of the scan. [1,2]
- **Periodic**: Executes whenever the elapsed time evaluates as true.
- **Data Change**: Executes when there is a change in data from one scan to the next.

[1] Data changes between each scan are not evaluated, only the value at the beginning of each script is used for evaluation purposes, in other words if a boolean attribute were to change from True to False to True again during 1 scan cycle, this change would not be evaluated as a data change as the value is True at the beginning of each scan cycle.

[2] Time-based script considerations (a trigger period of **0** means that the script execute every scan).

Time-based scripts, **WhileTrue**, **WhileFalse**, and **Periodic** are evaluated and executed based on the elapsed time from a timestamp generated from the previous execution, not on an elapsed time counter. Because of this it is possible that a change in the system clock can result in the interval between execution of these script to be off.

Example

A periodic script is set to run every 60 minutes. The script executes at 11:13 am, therefore we would expect it to execute 60 minutes later at 12:13 PM. However, let's assume that a time sync event has occurred and the node's time is adjusted from 11:33 am to 11:30 am. The script will still execute when the system time reaches 12:13 PM. But because of the time change the actual (True) time period that has elapsed between executions is 63 minutes.

OffScan

Called when the object is taken OffScan. Primarily used to clean up the object and account for any needs that should be addressed as a result of the object no longer executing.

Shutdown

Called when the object is about to be taken out of memory, usually as a result of the AppEngine stopping. Primarily used to destroy COM objects and .NET objects and clean up memory.

Deployment Considerations

The deployment of objects is both critical and a load-intensive process to the IAS Galaxy.

The effects of implementing scripting in the **Startup** and **OnScan** methods can have a negative impact on the deployment and redundancy performance of the IAS Galaxy.

While objects are being deployed, their Startup and OnScan scripts are executed. These scripts must complete within the deployment timeout period in order for the deployment to be successful.

Placing large numbers of scripts, or scripts that require heavy processing power into the Startup or OnScan script methods can cause deployment or failover to be slowed down or fail.

In addition to the load that is placed on the system at deployment time, further consideration should be given to the type or scripting done in the Startup and OnScan methods, since these scripts may execute in a sequence other than what is desired or expected. During deployment and restart, the Startup and OnScan script methods do not execute objects based on execution order. Objects are started up and placed on scan based on their Alpha-Numeric tagname within their hosting Area.

Please follow the recommendation made for each type of script method to help determine what scripting practices should be followed in each script method.

Examples of scripting that should *not* be placed in the Startup or OnScan scripting are:

- Database access.
- File system access, .csv, .xml, .txt, etc..
- Off-object referencing.
- Dynamic referencing.

Dynamic Referencing Considerations

Dynamic reference scripting is one the biggest causes of deployment failures. It is one of the most common misuses of the Startup and OnScan methods.

Rather than placing dynamic referencing scripts in the Startup or OnScan methods, dynamic referencing should be performed in the **Execute** method. Several advantages can be gained by using the Execute method for this scripting type:

- Deployment is quicker.
- Deployment is more reliable.
- Deterministic execution order is guaranteed.
- Off-object and off-engine attributes are available.
- After a failover occurs, the startup of the redundant engine is more stable and can be quicker.

Example

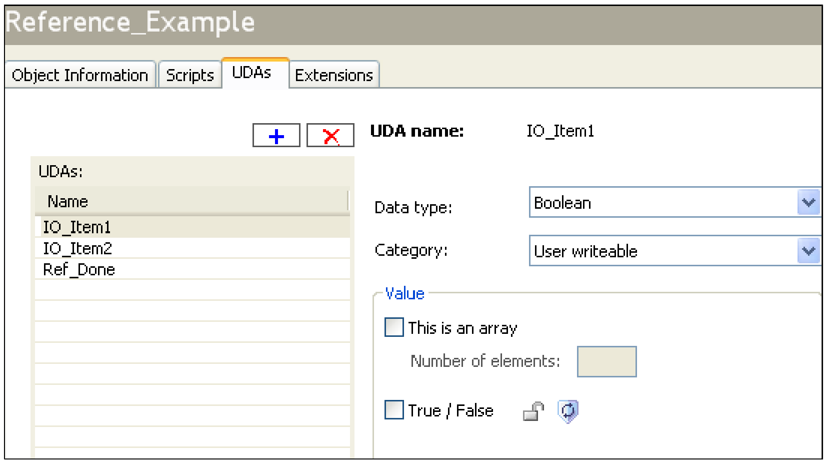
Simple Dynamic Reference Script

1. Create a Boolean UDA (Figure 1 below).

   The UDA indicates whether or not the referencing script has been completed.

   In this example I have created Ref_Done.

   IO_Item1 and **IO_Item2** are the I/O points referenced in this example:



UDA name: IO_Item1

UDAs:

Name
IO_Item1
IO_Item2
Ref_Done

Data type: Boolean

Category: User writeable

This is an array

Number of elements:

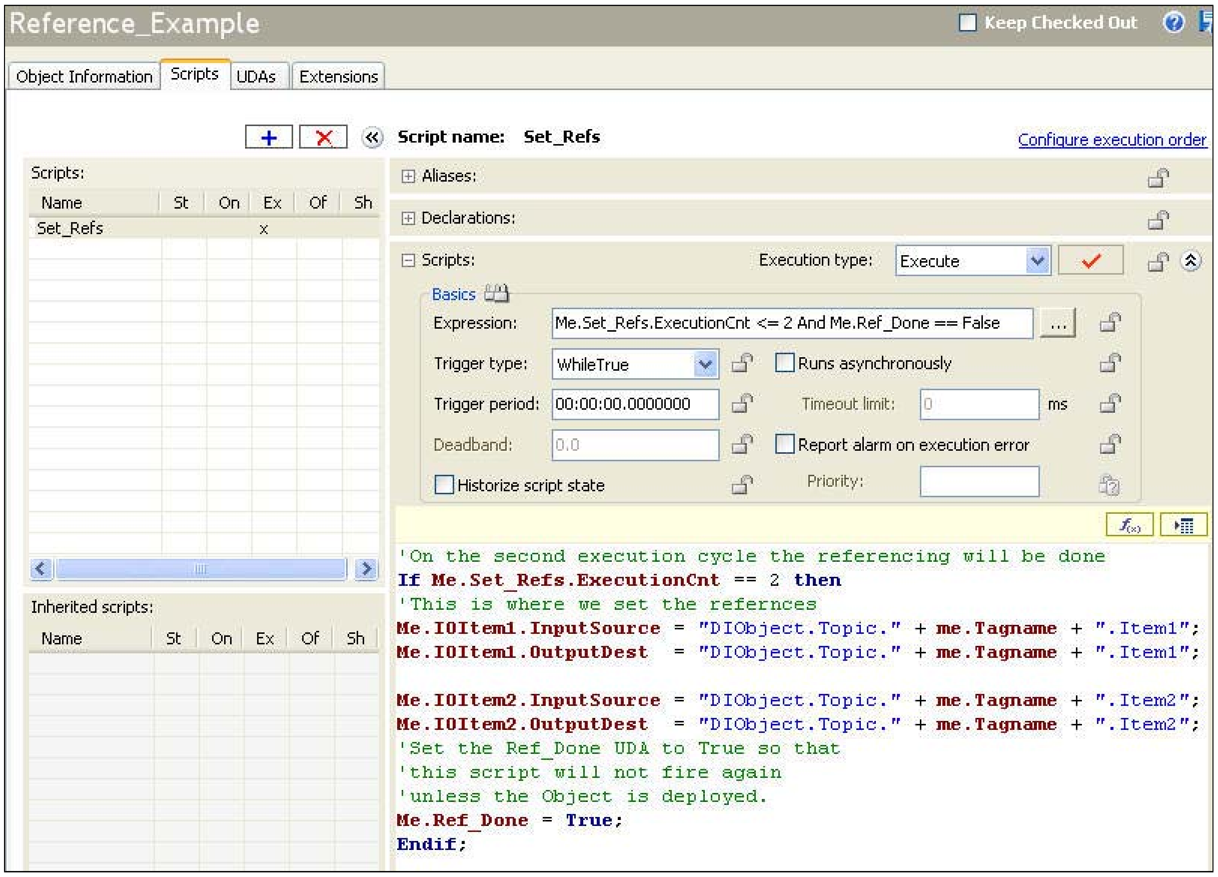
True / False

FIGURE 1: UDA REFERENCES

2. Create the script.

   The script in this example is called **Set_Refs**.

   The script has a trigger type of **While True** with a **0** trigger period:



Script name: Set_Refs

Scripts:

Name	St	On	Ex	Of	Sh
Set_Refs			x		

Inherited scripts:

Name	St	On	Ex	Of	Sh

Scripts: Execution type: Execute

Expression: Me.Set_Refs.ExecutionCnt <= 2 And Me.Ref_Done == False

Trigger type: WhileTrue

Runs asynchronously

Trigger period: 00:00:00.0000000

Timeout limit: 0 ms

Deadband: 0.0

Report alarm on execution error

Historize script state

Priority:

```
'On the second execution cycle the referencing will be done
If Me.Set_Refs.ExecutionCnt == 2 then
'This is where we set the refernces
Me.IOItem1.InputSource = "DIObject.Topic." + me.Tagname + ".Item1";
Me.IOItem1.OutputDest  = "DIObject.Topic." + me.Tagname + ".Item1";

Me.IOItem2.InputSource = "DIObject.Topic." + me.Tagname + ".Item2";
Me.IOItem2.OutputDest  = "DIObject.Topic." + me.Tagname + ".Item2";
'Set the Ref_Done UDA to True so that
'this script will not fire again
'unless the Object is deployed.
Me.Ref_Done = True;
Endif;
```

FIGURE 2: SET_REFS SCRIPT EXAMPLE

The script (without comments) is reproduced below for reference:

```
If Me.Set_Refs.ExecutionCnt == 2 then
Me.IOItem1.InputSource = "DIObject.Topic." + me.Tagname + ".Item1";
Me.IOItem1.OutputDest = "DIObject.Topic." + me.Tagname + ".Item1";

Me.IOItem2.InputSource = "DIObject.Topic." + me.Tagname + ".Item2";
Me.IOItem2.OutputDest = "DIObject.Topic." + me.Tagname + ".Item2";
Me.Ref_Done = True;
Endif;
```

This script is designed to allow the system to stabilize after going on scan before setting the references. This script will execute on the first two scans of the object as long as the Boolean attribute **Ref_Done** is false.

As the script is executed a check is made against the execution count. If the count equals 2 the script proceeds to perform the referencing operations. Once the reference attributes are set on the UDAs, the **Ref_Done** UDA is set to **True**. At this point the expression for the script is no longer true.

The three attributes that are set in this script are checkpointed, eliminating the need to run this script except on deployment. The next time the object is started, placed on scan, or failed over there is no need to recreate the references the items.

B. Hunter

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 **Wonderware Technical Support**

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

back to top