TECHNICAL WHITE PAPER: CUSTOM INVENTORY FOR ITMS 7.1

Symantec
Confidence in a connected world.

Custom Inventory for ITMS 7.1

Creating Custom Inventory with VBScript

Steven Riley, Technical Field Enablement
Inventory Solution

Symantec Technical White Paper

Version 1.0 Released

Custom inventory for ITMS 7.1

Creating Custom Inventory with VBScript

Table of Contents

Introduction to Custom Inventory

The approach to creating custom inventory for an environment follows some simple steps from data class creation, script testing to validating that the data flows through the system and associates with the appropriate machine. This paper steps through the initial creation of a Custom Inventory and demonstrates how to create the data classes along with three simple scripting examples; a registry read example, an INI file read example and then capturing an Operating System event example.

We will demonstrate what we believe are best practices for the creation of these script items including the process to follow for creating them. It is important to realize that many functions are available directly within Inventory Solution itself please review the Inventory Solution documentation so you are clear what you are able to do from the solution before starting with custom inventory.

Introduction to Data Classes

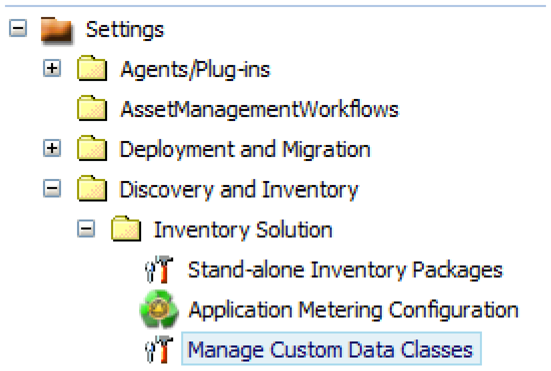
Custom Inventory has a specific area were Data Classes are to be created, these are located through the Symantec Management Console from the console menu select settings->Discovery and Inventory->Manage Custom Data Classes

So what are Data Classes? Data Classes in the world of Symantec Altiris products are simply tables that are created within the data base. So when we are referring to Data Classes we are talking about the fields, the data type and actual information created in the database. This data is usually associated with a computer or other resource of some type. We typically will talk about “custom” data types only within Inventory Solution and Asset Management Suite.

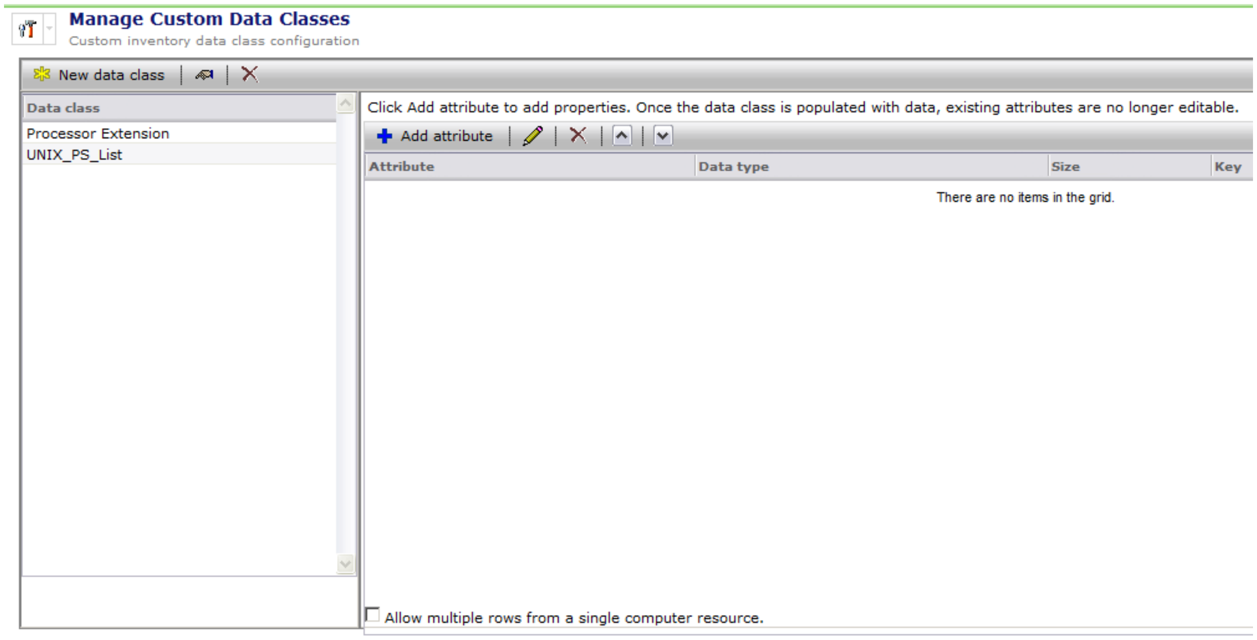
Let’s have a look at how custom data classes are created for Inventory Solution and discuss the location from a Symantec Management Console point of view and then look at the specific tables within the database. We’ll also discuss some of the elements that you require to be able to write scripts that send information to these data classes.

First let’s create a Data Class for ourselves called Monitor Print Spooler, and within that data class we’ll create a field called ‘Status’ were we will store information about the Microsoft Print Spooler, and send information such as is it “Running” or is it “Stopped”?

We always create Custom Inventory Data Classes through the Manage Custom Data Classes interface located here:



Click on the Manage Custom Data Classes to bring up the following screen:

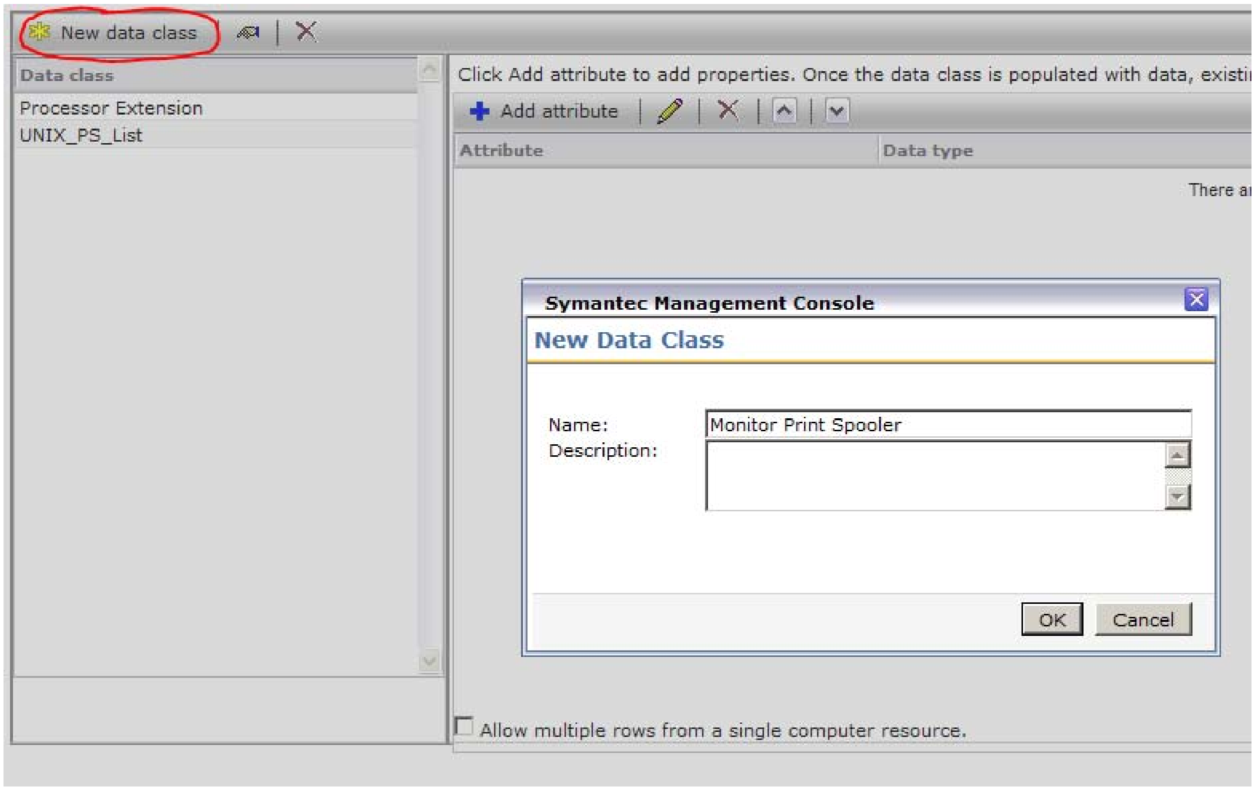


Click Add attribute to add properties. Once the data class is populated with data, existing attributes are no longer editable.

There are no items in the grid.

Allow multiple rows from a single computer resource.

We now need to create the Custom Data Class, to do this click on the New Data Class button found top right to display the New Data Class; here we'll create a data class called Monitor Print Spooler:



Click Add attribute to add properties. Once the data class is populated with data, existi

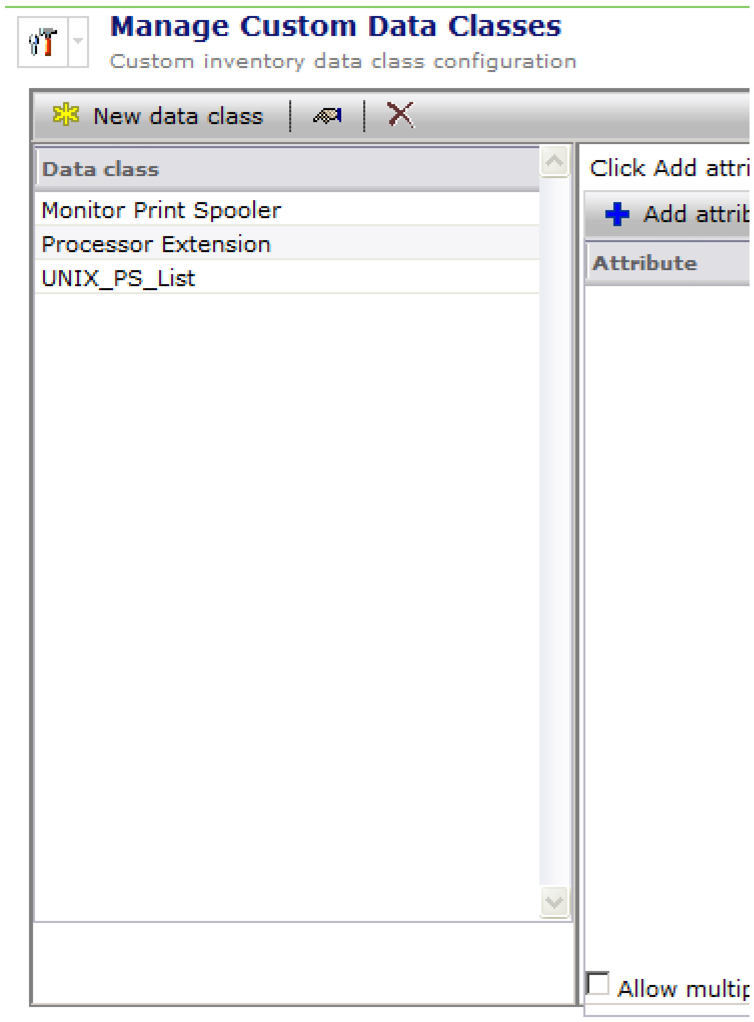
There a

Name: Monitor Print Spooler
Description:

Allow multiple rows from a single computer resource.

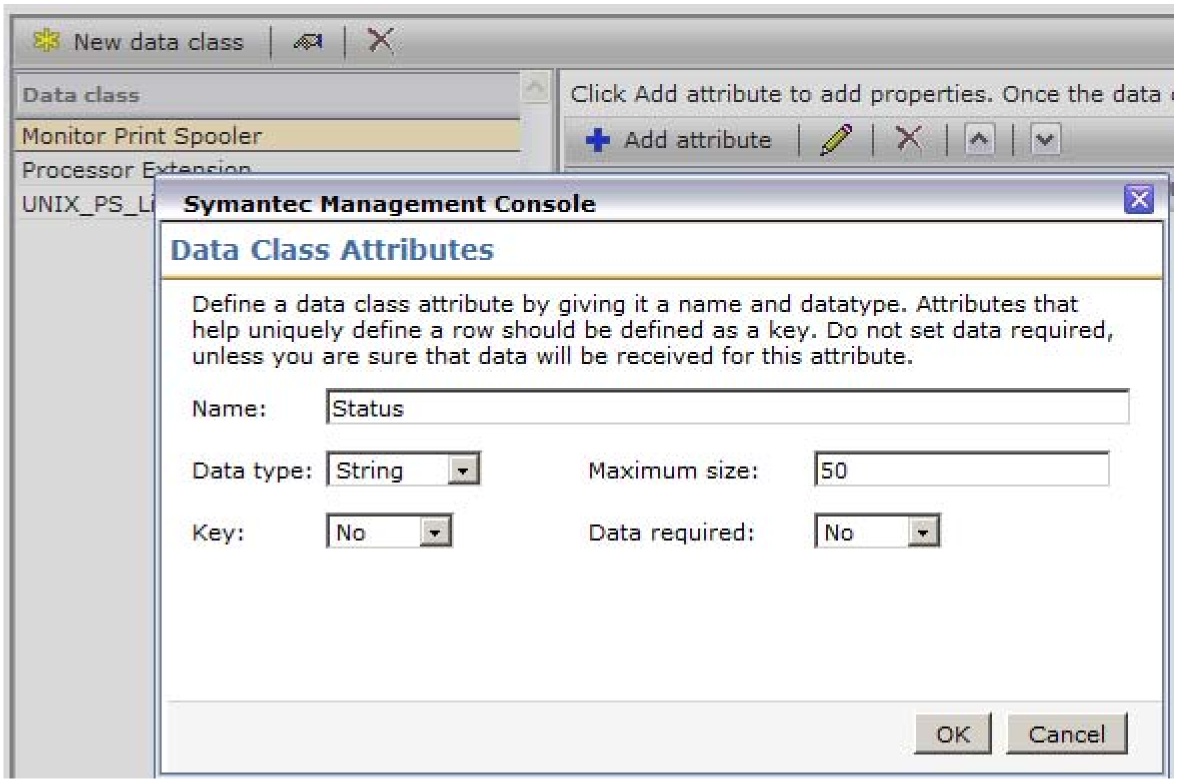
Click OK for the initial Data Class to be created.

You should see a screen similar to this with the Monitor Print Spooler data class listed.



Monitor Print Spooler
Processor Extension
UNIX_PS_List

Now we need to add some attributes by clicking the button. Before you click Add Attribute, click on the Monitor Print Spooler data class and then add a field called Status; ensure that you have clicked on Monitor Print Spooler first:



Click Add attribute to add properties. Once the data

Symantec Management Console

Data Class Attributes

Define a data class attribute by giving it a name and datatype. Attributes that help uniquely define a row should be defined as a key. Do not set data required, unless you are sure that data will be received for this attribute.

Name: Status

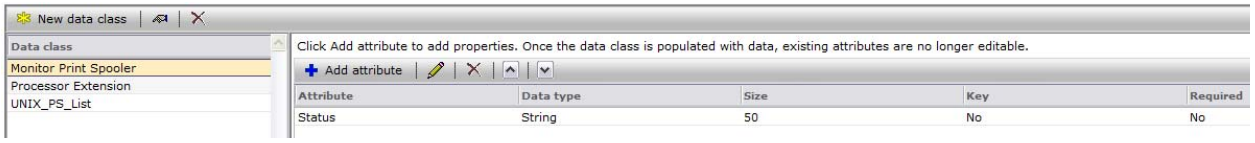
Data type: String

Maximum size: 50

Key: No

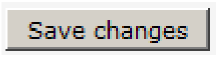
Data required: No

As indicated above we do not need a key as we are not storing multiples of these, in this case we only want a one to one relationchip, meaning that we only want to know when the item is runnning or when it is stopped. Click OK, your screen should look like the following:



Click Add attribute to add properties. Once the data class is populated with data, existing attributes are no longer editable.

Attribute	Data type	Size	Key	Required
Status	String	50	No	No

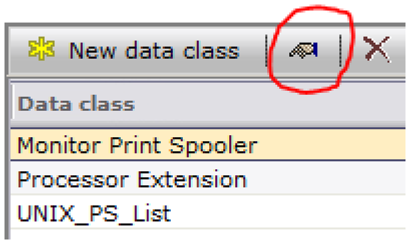
You MUST click on the  button found at the bottom of the screen.

You have now successfully created a data class that you can use for collecting data. Next let’s look at some of the details behind this data class we just created.

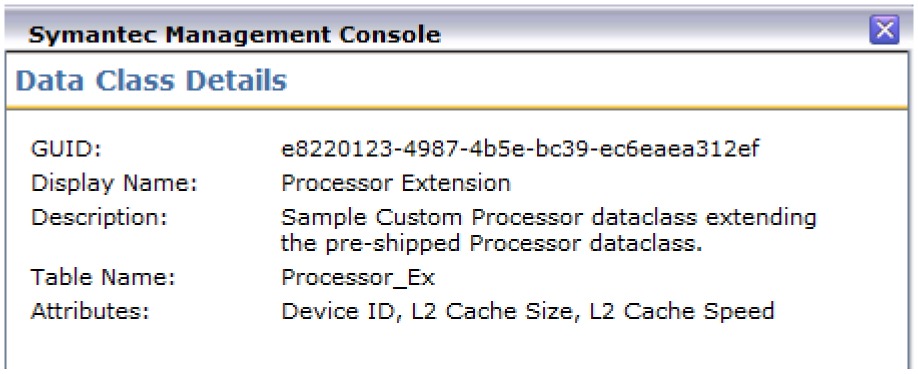
Data Class in Detail

So you have created a Data Class, great, so where does all this information get stored you might ask yourself? Let’s look at a few locations you may look to find the data class you just created. Those that have used or are familiar with Asset Management Suite will know a little about these locations already.

The first location you can look at details about your data class is by clicking on the properties icon found just above the data classes themselves as shown here:



This will open a screen showing you the GUID, Display name and Table name along with the fields you just created e.g.

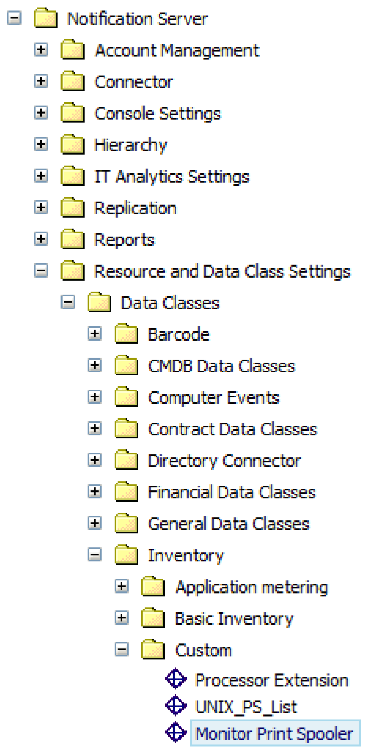

Symantec Management Console

Data Class Details

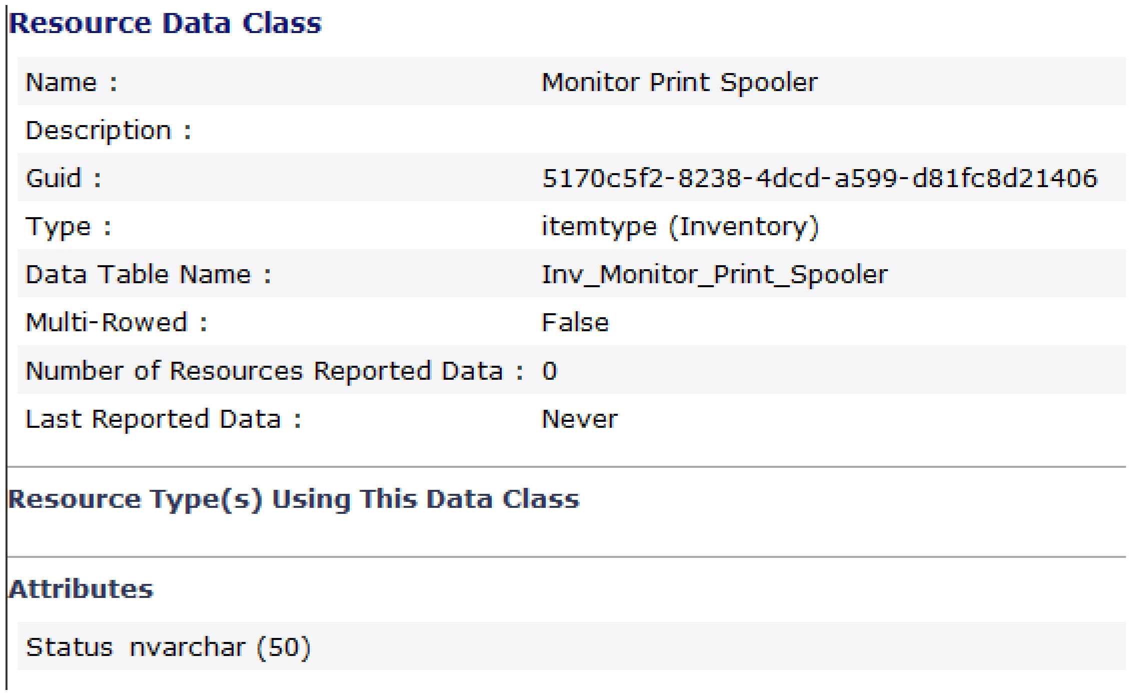
GUID:	e8220123-4987-4b5e-bc39-ec6eaea312ef
Display Name:	Processor Extension
Description:	Sample Custom Processor dataclass extending the pre-shipped Processor dataclass.
Table Name:	Processor_Ex
Attributes:	Device ID, L2 Cache Size, L2 Cache Speed

This is great information, you need to know the table to open up from an MS SQL perspective and you have the GUID which you'll need for your VBscript; you'll want to reference the data class via it's GUID name when you write your final script.

The other location you will find this information is also located under the Notification Server under Data Classes:



If you just click on the Monitor Print Spooler you get the following information:



Resource Data Class

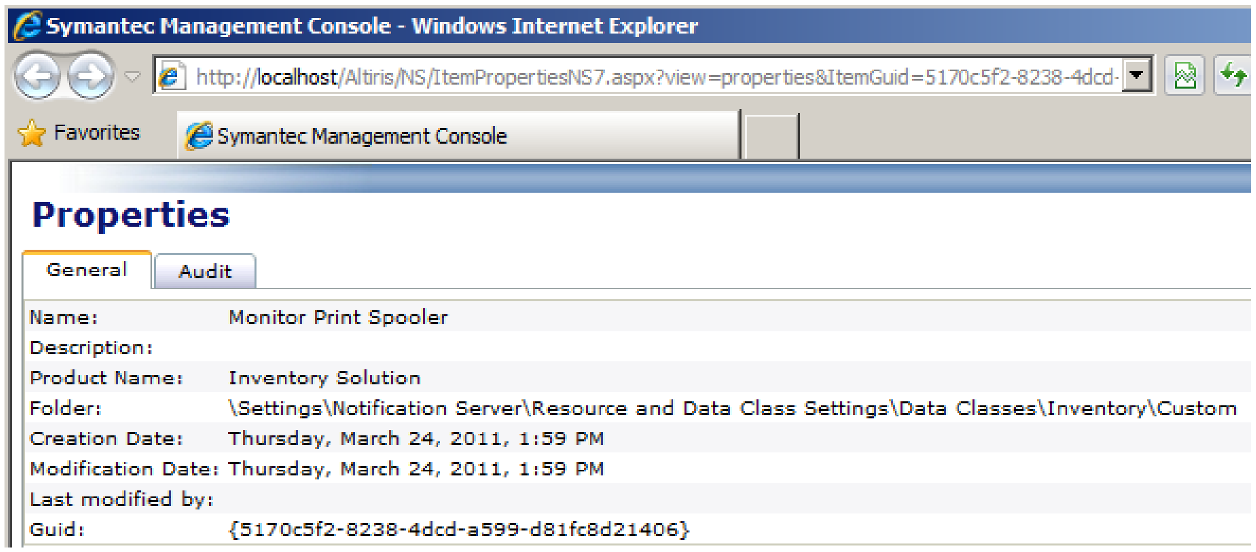
Name :	Monitor Print Spooler
Description :	
Guid :	5170c5f2-8238-4dcd-a599-d81fc8d21406
Type :	itemtype (Inventory)
Data Table Name :	Inv_Monitor_Print_Spooler
Multi-Rowed :	False
Number of Resources Reported Data :	0
Last Reported Data :	Never

Resource Type(s) Using This Data Class

Attributes

Status nvarchar (50)

When you click on the properties from the right click menu you will have the following displayed:

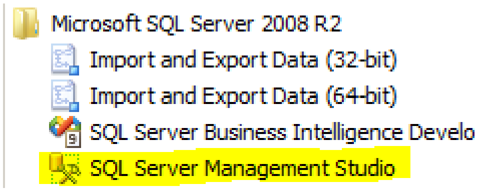


Symantec Management Console - Windows Internet Explorer
http://localhost/Altiris/NS/ItemPropertiesNS7.aspx?view=properties&ItemGuid=5170c5f2-8238-4dcd-
Favorites
Symantec Management Console

Properties

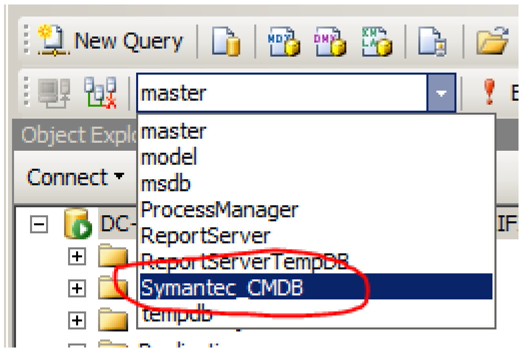
Name:	Monitor Print Spooler
Description:	
Product Name:	Inventory Solution
Folder:	\Settings\Notification Server\Resource and Data Class Settings\Data Classes\Inventory\Custom
Creation Date:	Thursday, March 24, 2011, 1:59 PM
Modification Date:	Thursday, March 24, 2011, 1:59 PM
Last modified by:	
Guid:	{5170c5f2-8238-4dcd-a599-d81fc8d21406}

This gives you the GUID and the Table name. To view the Table open up the SQL Server Management Studio:

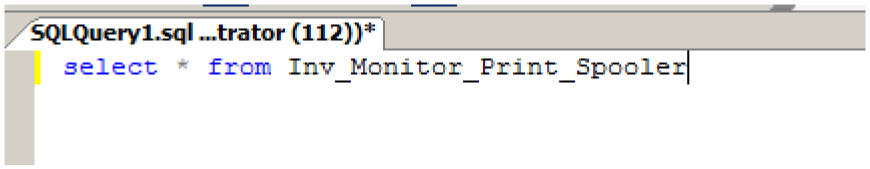


Microsoft SQL Server 2008 R2
Import and Export Data (32-bit)
Import and Export Data (64-bit)
SQL Server Business Intelligence Develo
SQL Server Management Studio

Click on New Query found at the top right of your screen and select the Symantec_CMDB database as shown:

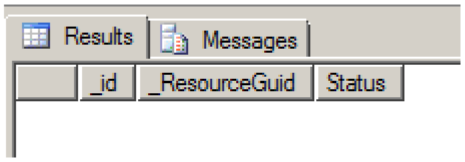


Within the Query windows right the following code as shown here:

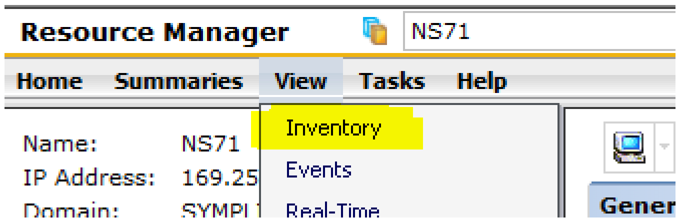


```
select * from Inv_Monitor_Print_Spooler
```

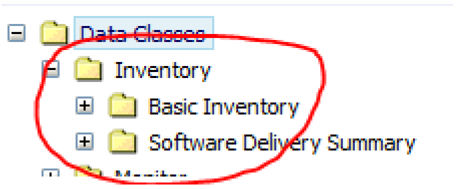
You'll see the folllowing window with no results in it; this is OK as we haven't collected any data at this time.



When the data class is created and you are tracking the flow of data against a specific machine you can find the detail by running the resource explorer against that specific machine and selecting inventory from the view menu:



Look under Data Classes Inventory and if the data came in correctly you'll see another Folder entry called Custom and you'll find the data class within that folder:

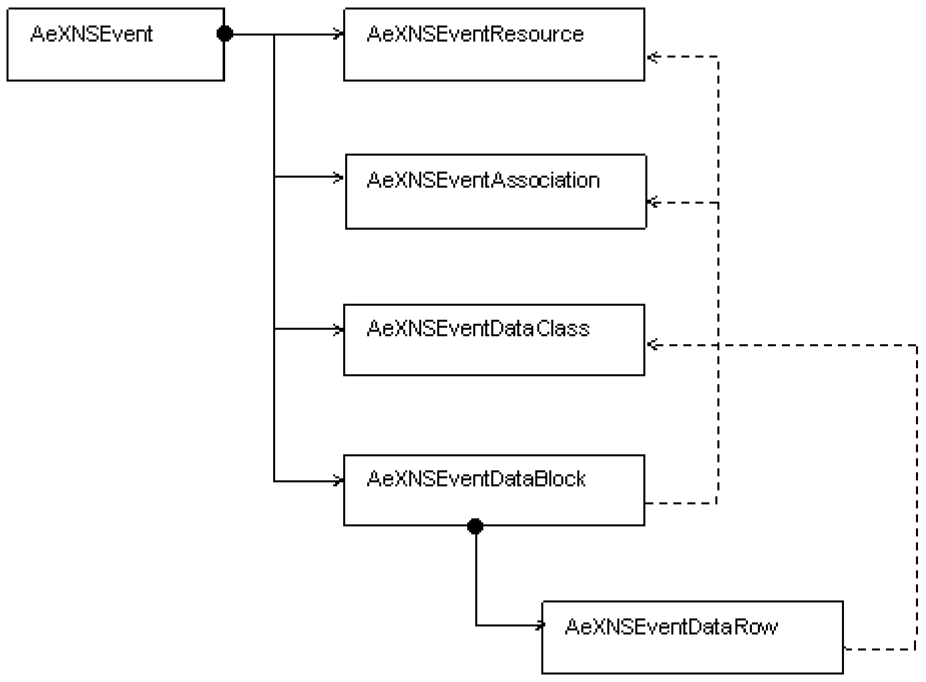


We'll show this later in the document when showing how the scripts work.

Now that all the pieces are in place for storing the data our next steps will be to go ahead and write a script to collect the information we want, then modify the script to work with the Altiris Agent and send data back to the Notification Server. The next sections will cover writing various scripts to collect the information.

Development Objects and Methods

The agent NSE Generator provides an object model that allows the developer to create an NSE without working directly with XML.



Object Model

AeXNSEvent Object

AeXNSEvent represents the overall event object. This object has methods to create resources, dataclasses, and datablocks; to generate the XML and hashes, and to send the event to the NS; and properties to manage the overall event.

When the object is serialized to XML, either to obtain the XML or send the XML to a server as an event, any special characters that have been added to the data are escaped.

Properties

Name	Mode	Type	Description
To	get/set	string	Guid or alias of the item that handles the event
Priority	get/set	int	The priority of the event
Guid	get	GUID	The GUID of the event. This is generated when the event is created.
DefaultResource	get	IAeXNSEvent*	The default resource object, this represents the computer that the agent is running on. *Requires the Altiris Agent be installed.*
Xml	get	string	Returns the event XML.
Hash	get	string	Returns the hash for the entire event, excluding any data that is excluded from hashing.

Methods

Name	Parameter	Type	Required	Description
AddResource				Adds a new resource to the event.
	return	IAeXNSEventResource*		The newly added resource.
	typeGuid	GUID	Required	The type GUID of the resource.
	guid	VARIANT	Optional	The GUID of the resource if it is known. If the GUID is not known it may be omitted, and keys should be added to the returned resource object.
AddDataClass				Adds a new dataclass to the event.
	return	IAeXNSEventDataClass*		The newly added dataclass
	dataClassName	VARIANT	Required	Either the name or GUID of the dataclass must be specified.
AddDataBlock				Adds a datablock to the event. A dataclass and resource must be specified. For simple dataclasses the name or GUID of the dataclass may be specified instead of adding a dataclass. If the data applies to the computer resource, the resource may be omitted.
	return	IAeXNSEventDataBlock*		The newly added datablock.
	dataClass	VARIANT	Required	The dataclass that the new data will be used to populate. This may be an AeXNSEventDataClass object, the name of the dataclass, or the GUID of the dataclass.
	resource	VARIANT	Optional	The resource that the data applies to. If this is omitted the default resource (the computer) is used.
AddAssociation				Adds a resource association. Either resource GUIDs or resource objects may be specified to create the association.
	return	IAeXNSEventAssociation*		The newly added association.

	type	GUID	Required	The GUID of the resource association type.
	parent	VARIANT	Required	The resource that is the parent of the association. This may be an AeXNSEventResource object or a GUID.
	child	VARIANT	Optional	The resource that is the child of the association. This may be an AeXNSEventResource object or a GUID. If this parameter is omitted children must be added to the association.
Send				Sends the event to the server. *Requires the Altiris Agent be installed.*
	server	VARIANT	Optional	The name of the server. If omitted the event is sent to the NS that the agent is reporting to.
SendQueued				Queues the event for the server. *Requires the Altiris Agent be installed.*
	server	VARIANT	Optional	The name of the server. If omitted the event is sent to the NS that the agent is reporting to.

AeXNSEventResource Object

AeXNSResource represents a single resource that has data contained in the event. Using the resource object it is possible to manage resource keys.

Properties

Name	**Mode**	**Type**	**Description**
Guid	get	GUID	Returns the resource GUID, if it has been specified
TypeGuid	get	GUID	Returns the resource type GUID.
Name	get/set	string	The name of the resource, this will be used if the resource is created.
NumKeys	get	int	The number of keys that have been added
KeyName	get	string	The name of the indexed key
KeyValue	get	string	The value of the indexed key
KeyType	get	KeyOperationMode	The type of the indexed key, whether it is add or replace.
Deleted	get/set	bool	Set this to true to delete the resource from the server.

Methods

Name	**Parameter**	**Type**	**Required**	**Description**
AddKey				Adds a key to the resource.
	name	string	Required	The name of the key, such as "name.domain"
	value	string	Required	The value of the key.
	operation	KeyOperationMode	Required	The operation to be performed with the key, either KEYOP_ADD which will add new keys to the resource, or KEYOP_REPLACE which will replace existing keys on the resource with the new keys.
AddAssociation				Adds a new association with this resource being the parent. For a description of this method and it's parameters see IAeXNSEvent::AddAssociation ()

AeXNSEventDataClass Object

AeXNSDataClass represents a dataclass. Using the dataclass object it is possible to specify columns names for the dataclass, as well as which columns participate in hashing.

It is not strictly necessary to specify column names, since columns are represented in the event using the short names, and column names are only used when specifying data using column names, in order to look up the short name.

Any columns that do not participate in hashing should be specified to ensure that correct hashes are generated.

Properties

Name	**Mode**	**Type**	**Description**
Name	get	string	The name or GUID of the dataclass.
IgnoreHash	get/set	bool	Controls whether the indexed column is excluded from hashing. This is the most common use of the dataclass object.
NumColumns	get	int	The number of columns defined for the dataclass.
FirstColumn	get/set	int	The short name index for the first column. This is usually either 0 or 1, representing short names of *c0* and *c1*, respectively.

Methods

Name	**Parameter**	**Type**	**Required**	**Description**
AddColumn				Adds a named column to the dataclass, whether the column is hashed may also be specified.
	name	string	Required	The name of the column.
	ignore	VARIANT	Optional	A boolean indicating whether the column is ignored when hashing. If omitted this defaults to *false*.
SetColumn				Sets the name of the indexed column, whether the column is hashed may also be specified.
	index	int	Required	The index of the column being specified.
	name	string	Required	The name of the column.
	ignore	VARIANT	Optional	A boolean indicating whether the column is ignored when hashing. If omitted this defaults to *false*.

AeXNSEventDataBlock Object

AeXNSDataBlock represents a block of data that applies to a particular resource, and is in the format described by a particular dataclass. This object has properties indicating whether the datablock should be used to perform a partial or full update; and methods that allow adding additional rows of data as well as inserting field data into the rows.

Properties

Name	Mode	Type	Description
DataClass	get	IAeXNSEventDataClass*	Returns the dataclass object which specifies the datablock schema.
Resource	get	IAeXNSEventResource*	Returns the resource object that the data applies to.
PartialUpdate	get/set	bool	Specifies whether the datablock should completely replace the existing data, or be appended.
Hash	get	string	The hash of all of the data in the datablock.

Methods

Name	Parameter	Type	Required	Description
AddRow				Begins a new row, and returns a row object.
	return	IAeXNSEventDataRow*		The newly added row.
	type	VARIANT/RowType	Optional	The type of row. This may be REPLACE_ROW to replace the row, which is default if omitted; MERGE_ROW to merge rows with missing columns into the existing data (which is the default behaviour when fields are missing); and DELETE_ROW to delete rows if they already exist.

AeXNSEventDataRow Object

AeXNSEventDataRow represents a single row of data.

Properties

Name	Mode	Type	Description
Hash	get	string	The hash of all of the fields in the row.

Methods

Name	Parameter	Type	Required	Description
AddField				Adds a field to the current row.
	value	VARIANT	Optional	The value of the field. This may be a string, a number, a GUID, a resource object to reference another resource defined in the event, NULL to represent a database NULL value, or omitted to represent the default value (the default value is specified for the column when the dataclass is configured on the NS.)
SetField				Sets the value of the indexed field.
	index	VARIANT	Required	The index of the field to set, this may be an integer index or the name of the field.
	value	VARIANT	Optional	The value of the field. This may be a string, a number, a GUID, a resource object to reference another resource defined in the event, NULL to represent a database NULL value, or omitted to represent the default value (the default value is specified for the column when the dataclass is configured on the NS.)

AeXNSEventAssociation Object

AeXNSEventAssociation represents an association between a resource and one or more other resources.

Properties

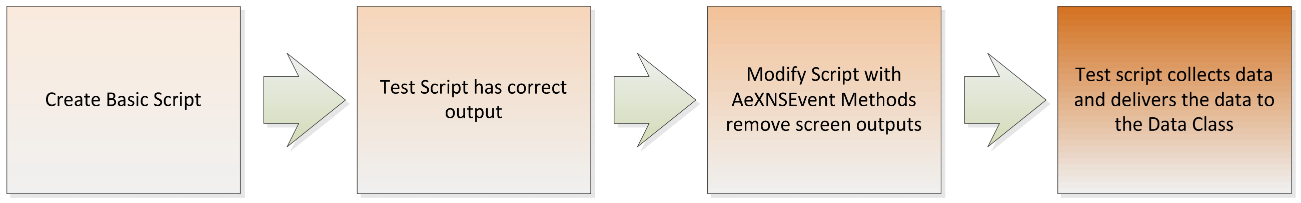
Name	Mode	Type	Description
Mode	get/set	AssocMode	Determines the action taken when the association is loaded into the database. This may be ASSOC_ADD (the default), which will add the child resources to any existing association; ASSOC_REPLACE which will replace any existing associated resources with the child resources; and ASSOC_DELETE which will remove the child resources from any existing associations.
Hash	get	string	The hash of the association.

Methods

Name	Parameter	Type	Required	Description
AddChild				Adds a child resource to the association, this must be done if one is not specified when the association is created.
	child	VARIANT	Required	The resource to associate with the parent. This may be the GUID of the resource or an AeXNSEventResource object.

Scripting using VBScript

So now that we have created the data classes here we will look at various methods of collecting data and sending it back to the notification server for processing. Note that initial testing does not need to be run through task server or any other scheduled tasks, this should be a manual exercise so that you can slowly watch the data flow through the system and identify coding mistakes, as a best practice the following process is recommended:


Create Basic Script
Test Script has correct output
Modify Script with AeXNSEvent Methods remove screen outputs
Test script collects data and delivers the data to the Data Class

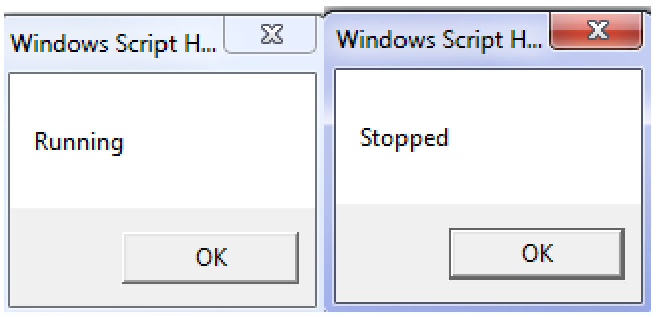
Process for Developing Scripts

Monitoring a Service Example

First we need to create the basic script that will display the monitoring service (we'll assume at this stage you have already created the data class as indicated in previous sections of this paper); the following is a basic illustration of a VBscript that will pull that information back from your machine:

```
----------------------VBSCRIPT STARTS HERE---------------------
strComputer = "."
Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\" & strComputer &
"\root\cimv2")
Set colRunningServices = objWMIService.ExecQuery("select * from win32_service")
For each objService in colRunningServices
   if objService.DisplayName = "Print Spooler" then
    wscript.echo objService.State
   end if
Next
----------------------VBSCRIPT ENDS HERE---------------------
```

When you run this script you'll see output like the following:



```
Windows Script H...
Running
OK
```

```
Windows Script H...
Stopped
OK
```

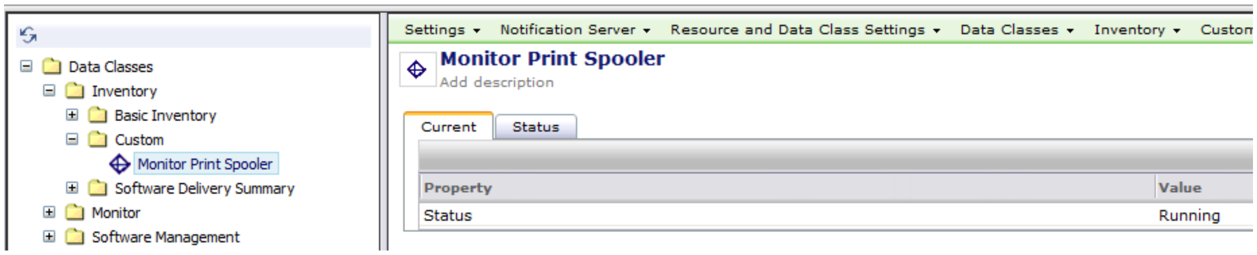
We test that all our scenarios work, now we go back and modify our script putting in the methods we need to call for sending the NSE; the following is an example script that will work for this scenario, compare the calls with the methods listed previously in this document, notice that we use the same script but instead of sending objService.State to the screen we send it to the Notification Server:

```
----------------------VBSCRIPT STARTS HERE---------------------
strComputer = "."
Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\" & strComputer &
"\root\cimv2")
Set colRunningServices = objWMIService.ExecQuery("select * from win32_service")
'==============================================
'Create instance of Altiris NSE component
dim nse
set nse = WScript.CreateObject ("Altiris.AeXNSEvent")
nse.To = "{1592B913-72F3-4C36-91D2-D4EDA21D2F96}"
nse.Priority = 1
dim objDCInstance
set objDCInstance = nse.AddDataClass ("{XXX}")
dim objDataClass
set objDataClass = nse.AddDataBlock (objDCInstance)
For each objService in colRunningServices
   if objService.DisplayName = "Print Spooler" then
     dim objDataRow
     set objDataRow = objDataClass.AddRow
     objDataRow.SetField 0, objService.State
   end if
Next
nse.SendQueued
----------------------VBSCRIPT ENDS HERE---------------------
```

In the above script there are two specific areas highlighted in yellow with text in red. The first of these is a GUID, this GUID is to remain the same, never change it. On any Notification Server if you look under Jobs and Tasks->Samples->Discovery and Inventory->Inventory Samples->Custom->Custom Inventory – Processor you'll find an example script with this GUID listed there as well so you will always have it somewhere at your disposal when developing.

The second item is XXX highlighted; this needs to be modified so that the XXX is replaced with the GUID of the data class that you created. Look at the previous section on Data Classes to see how to get the GUID.

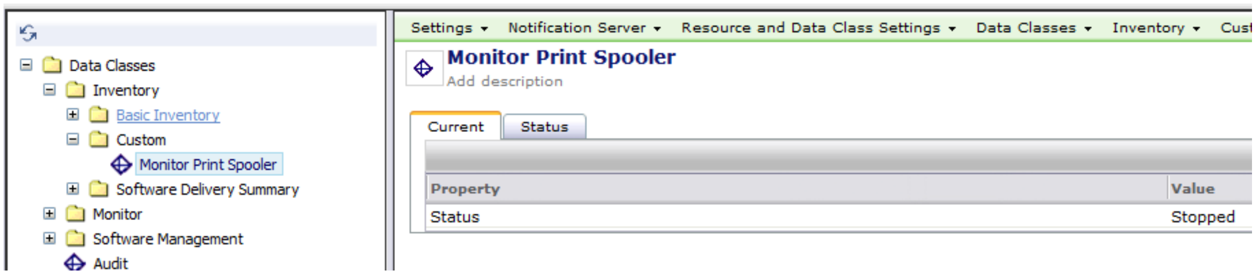
Finally when you run the script the following data should appear on the Notification Server. I run this script manually on the Notification Server itself, so in my case I right clicked on the computer I ran the script on and selected resource manager and drill into my custom inventory, I see the following:



Monitor Print Spooler

Property	Value
Status	Running

Note unlike before when we looked at the Resource Explorer the folder Custom is now available along with the data class showing the property that we created in the data class along with the value sent back to the notification server from our script. One more test to see that it works when we stop the service:



Monitor Print Spooler

Property	Value
Status	Stopped

The new value of Stopped came back, now we're ready to put the script into a Client side task and target the machines we want it to run on and create a schedule. See creating the Script at the end of this particular section.

Reading an INI File Example

First we need to create the basic script that will display the INI file content, in this case I'm only interested in the first data line so I created a very basic script that collected my basic data (we'll assume at this stage you have already created the data class as indicated in previous sections of this paper); the following is a basic illustration of a VBscript that will pull that information back from your machine:

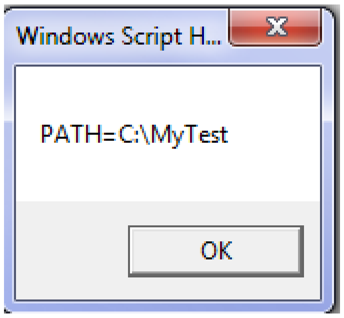
```
----------------------INI FILE DATA STARTS HERE---------------------
[INSTALL]
PATH=C:\MyTest
----------------------INI FILE DATA ENDE HERE---------------------
```

```
----------------------VBSCRIPT STARTS HERE---------------------
Dim arrFileLines()
i = 0
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.OpenTextFile("C:\data\MYTEST.INI", 1)
```

```
Do Until objFile.AtEndOfStream
    Redim Preserve arrFileLines(i)
    arrFileLines(i) = objFile.ReadLine
    i = i + 1
Loop
objFile.Close
Wscript.Echo arrFileLines(1)
----------------------VBSCRIPT ENDS HERE---------------------
```

Note: You may want to modify the highlighted directory and test file name for your purposes.

When you run this script you'll see output like the following:



```
PATH=C:\MyTest
```

We test that all our scenarios work, now we go back and modify our script putting in the methods we need to call for sending the NSE; the following is an example script that will work for this scenario, compare the calls with the methods listed previously in this document, notice that we use the same script but instead of sending arrFileLines(1) to the screen we send it to the Notification Server:

```
----------------------VBSCRIPT STARTS HERE---------------------
Dim arrFileLines()
i = 0
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.OpenTextFile("C:\data\MYTEST.INI", 1)
Do Until objFile.AtEndOfStream
    Redim Preserve arrFileLines(i)
    arrFileLines(i) = objFile.ReadLine
    i = i + 1
Loop

dim nse
set nse = WScript.CreateObject ("Altiris.AeXNSEvent")
nse.To = "{1592B913-72F3-4C36-91D2-D4EDA21D2F96}"
nse.Priority = 1
dim objDCInstance
```

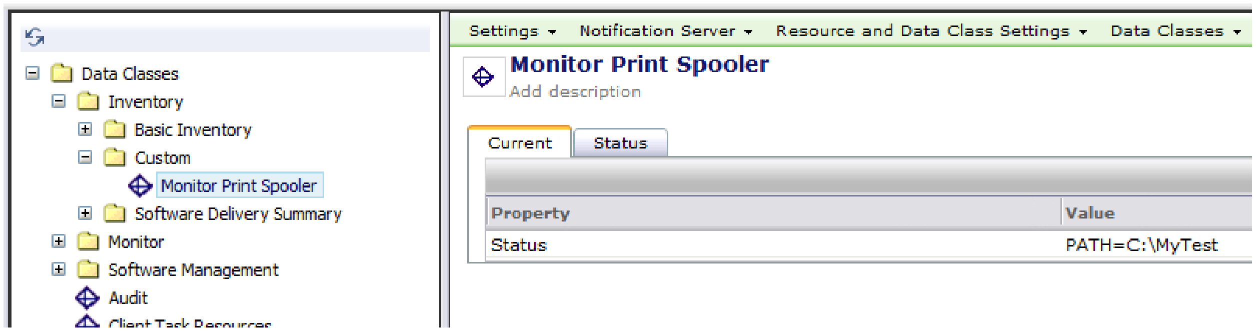
```
set objDCInstance = nse.AddDataClass ("{XXX}")
dim objDataClass
set objDataClass = nse.AddDataBlock (objDCInstance)
dim objDataRow
set objDataRow = objDataClass.AddRow
objFile.Close
objDataRow.SetField 0, arrFileLines(1)
nse.SendQueued

----------------------VBSCRIPT ENDS HERE--------------------
```

In the above script there are two specific areas highlighted in yellow with text in red. The first of these is a GUID, this GUID is to remain the same, never change it. On any Notification Server if you look under Jobs and Tasks->Samples->Discovery and Inventory->Inventory Samples->Custom->Custom Inventory – Processor you'll find an example script with this GUID listed there as well so you will always have it somewhere at your disposal when developing.

The second item is XXX highlighted; this needs to be modified so that the XXX is replaced with the GUID of the data class that you created. Look at the previous section on Data Classes to see how to get the GUID.

Finally when you run the script the following data should appear on the Notification Server. I run this script manually on the Notification Server itself, so in my case I right clicked on the computer I ran the script on and selected resource manager and drill into my custom inventory, I see the following:



Monitor Print Spooler

Property	Value
Status	PATH=C:\MyTest

Note unlike before when we looked at the Resource Explorer the folder Custom is now available along with the data class showing the property that we created in the data class along with the value sent back to the notification server from our script.

NOTE: that I provisioned the Monitor Print Spooler data class for demonstration purposes here.

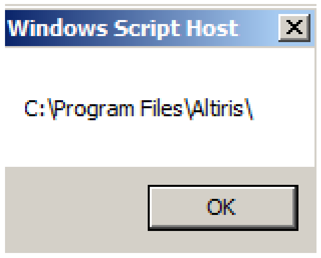
Reading the Registry Example

First we need to create the basic script that will display the INI file content, in this case I'm only interested in the first data line so I created a very basic script that collected my basic data (we'll assume at this stage you have already created the data class as indicated in previous sections of this paper); the following is a basic illustration of a VBscript that will pull that information back from your machine:

```
----------------------REGISTRY KEY START ---------------------
HKEY_LOCAL_MACHINE\Software\Altiris
          InstallPath    REG_SZ    C:\Program Files\Altiris\
---------------------- REGISTRY KEY END---------------------
```

```
----------------------VBSCRIPT STARTS HERE---------------------
'****Get the registry value***
const HKEY_LOCAL_MACHINE = &H80000002
strComputer = "."
Set oReg=GetObject("winmgmts:{impersonationLevel=impersonate}!\\" &_
 strComputer & "\root\default:StdRegProv")
strKeyPath = "SOFTWARE\Altiris"
strValueName = "InstallPath"
oReg.GetStringValue HKEY_LOCAL_MACHINE,strKeyPath,strValueName,strValue
wscript.echo strValue
----------------------VBSCRIPT ENDS HERE---------------------
```

When you run this script you'll see output like the following:



```
C:\Program Files\Altiris\
```

We test that all our scenarios work, now we go back and modify our script putting in the methods we need to call for sending the NSE; the following is an example script that will work for this scenario, compare the calls with the methods listed previously in this document, notice that we use the same script but instead of sending strValue to the screen we send it to the Notification Server:

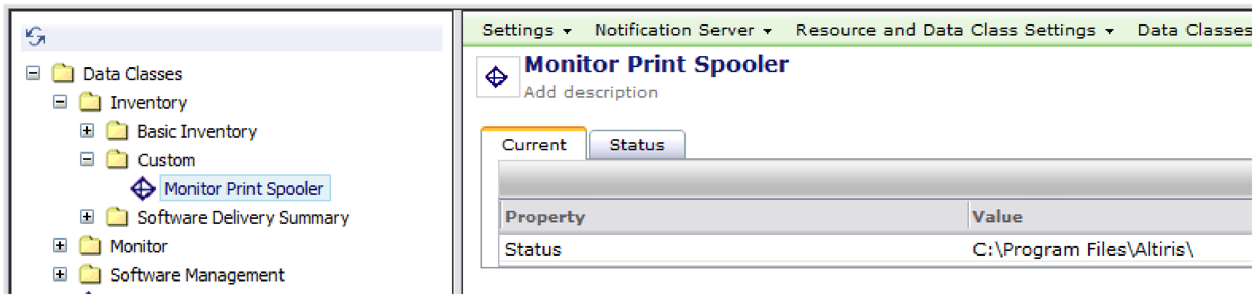
```
----------------------VBSCRIPT STARTS HERE---------------------
'****Get the registry value***
const HKEY_LOCAL_MACHINE = &H80000002
strComputer = "."
Set oReg=GetObject("winmgmts:{impersonationLevel=impersonate}!\\" &_
 strComputer & "\root\default:StdRegProv")
strKeyPath = "SOFTWARE\Altiris"
strValueName = "InstallPath"
oReg.GetStringValue HKEY_LOCAL_MACHINE,strKeyPath,strValueName,strValue

dim nse
set nse = WScript.CreateObject ("Altiris.AeXNSEvent")
nse.To = "{1592B913-72F3-4C36-91D2-D4EDA21D2F96}"
nse.Priority = 1
dim objDCInstance
set objDCInstance = nse.AddDataClass ("{XXX}")
dim objDataClass
set objDataClass = nse.AddDataBlock (objDCInstance)
dim objDataRow
set objDataRow = objDataClass.AddRow
objDataRow.SetField 0, strValue
nse.SendQueued
----------------------VBSCRIPT ENDS HERE---------------------
```

In the above script there are two specific areas highlighted in yellow with text in red. The first of these is a GUID, this GUID is to remain the same, never change it. On any Notification Server if you look under Jobs and Tasks->Samples->Discovery and Inventory->Inventory Samples->Custom->Custom Inventory – Processor you'll find an example script with this GUID listed there as well so you will always have it somewhere at your disposal when developing.

The second item is XXX highlighted; this needs to be modified so that the XXX is replaced with the GUID of the data class that you created. Look at the previous section on Data Classes to see how to get the GUID.

Finally when you run the script the following data should appear on the Notification Server. I run this script manually on the Notification Server itself, so in my case I right clicked on the computer I ran the script on and selected resource manager and drill into my custom inventory, I see the following:



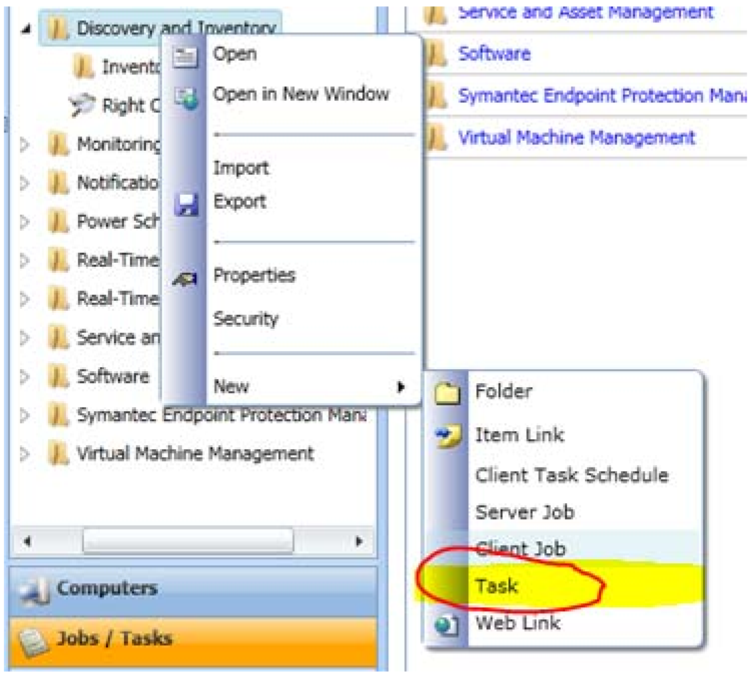
Property	Value
Status	C:\Program Files\Altiris\

Note unlike before when we looked at the Resource Explorer the folder Custom is now available along with the data class showing the property that we created in the data class along with the value sent back to the notification server from our script.

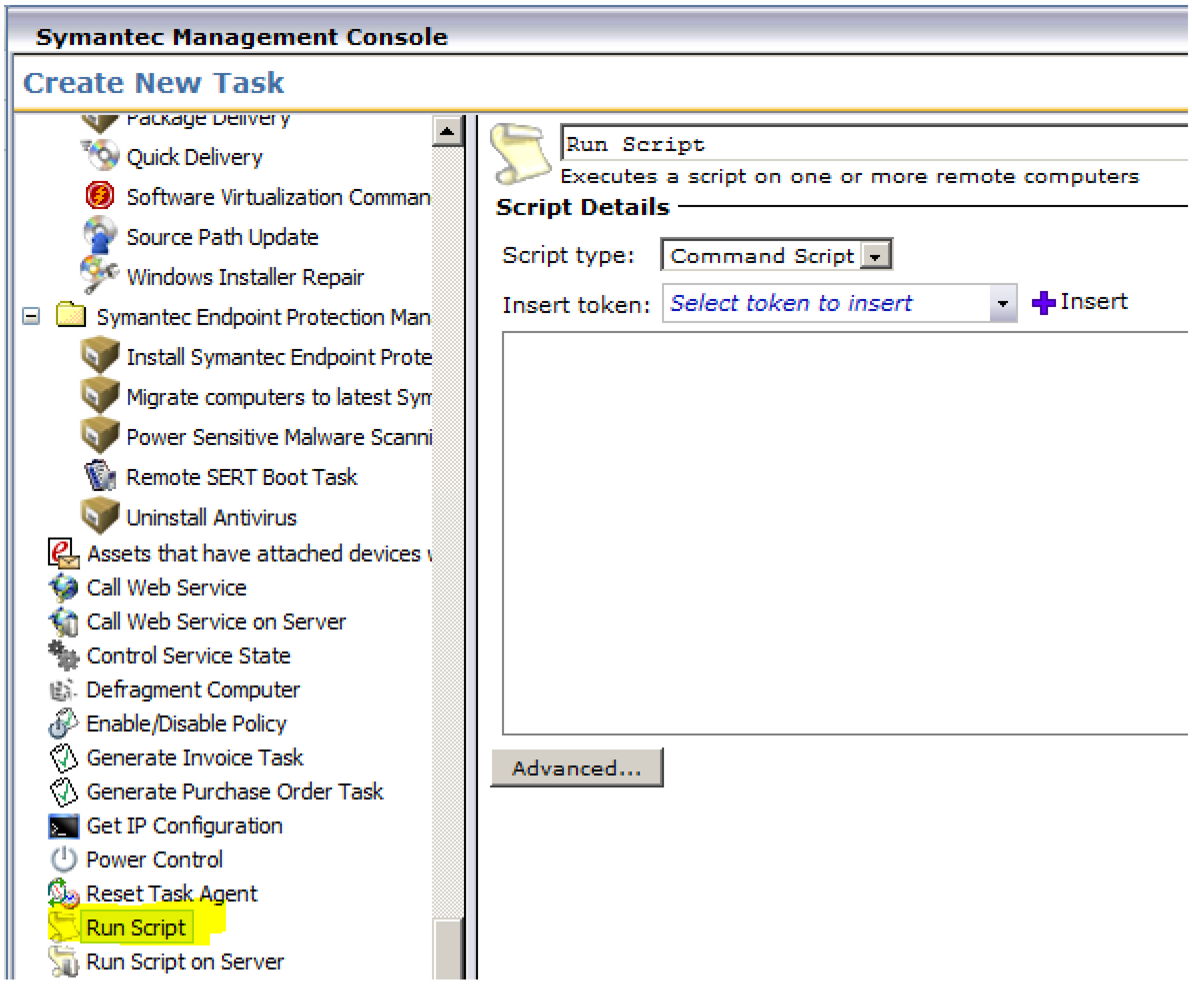
NOTE: that I provisioned the Monitor Print Spooler data class for demonstration purposes here.

Creating a Task for the Script

The script needs to be run as a client side script, the following steps through creating a client side script. First we need to drill down to Jobs and Tasks, then into an appropriate folder structure in Discovery and Inventory and right click and select



Scroll down to the Run Script task.



Run Script

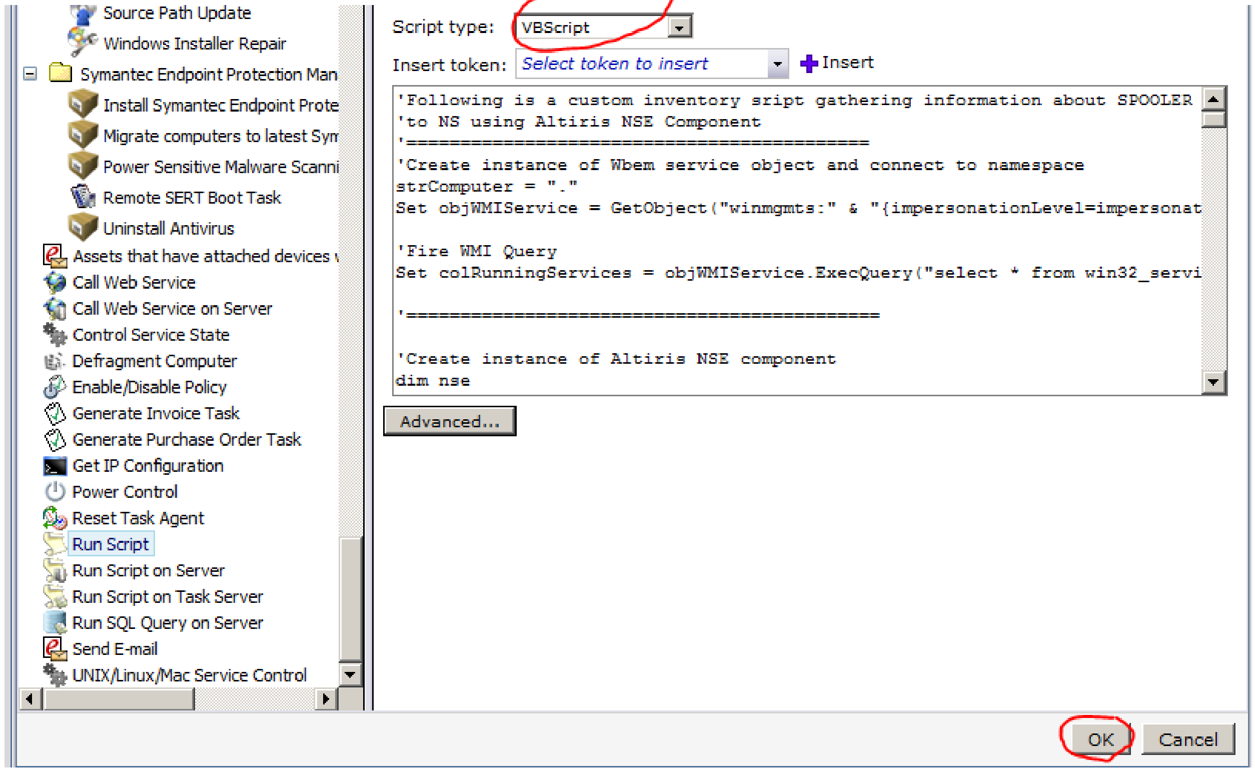
Executes a script on one or more remote computers

Script Details

Script type: Command Script

Insert token: Select token to insert

Place your script into the task and select VBScript from the Script Type then click OK.



Script type: VBScript

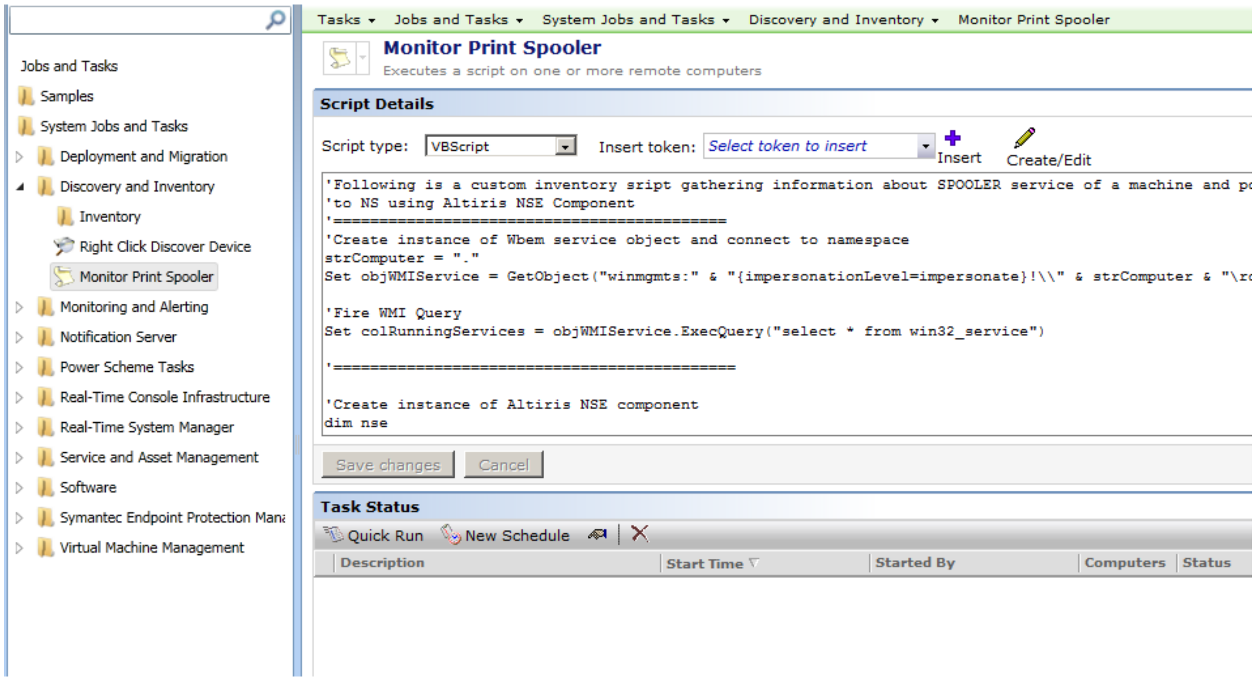
Insert token: Select token to insert

```
'Following is a custom inventory sript gathering information about SPOOLER
'to NS using Altiris NSE Component
'===========================================
'Create instance of Wbem service object and connect to namespace
strComputer = "."
Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonat
'Fire WMI Query
Set colRunningServices = objWMIService.ExecQuery("select * from win32_servi

'============================================

'Create instance of Altiris NSE component
dim nse
```

You should now have a Task created ready to schedule against your designated workstations. Remember to test against a small group first testing variations of the machines you wish to target.



Script type: VBScript

Insert token: Select token to insert

```
'Following is a custom inventory sript gathering information about SPOOLER service of a machine and pc
'to NS using Altiris NSE Component
'===========================================
'Create instance of Wbem service object and connect to namespace
strComputer = "."
Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\" & strComputer & "\rc

'Fire WMI Query
Set colRunningServices = objWMIService.ExecQuery("select * from win32_service")

'============================================

'Create instance of Altiris NSE component
dim nse
```

In Summary

This paper went through the process of creating data classes and looking at these data classes, creating basic scripts and then modifying those scripts so that you could then send NSE data back to the notification server. This is a start to your development exercises and using these techniques you will be able to work through the process of creating and working with more sophisticated inventories in the near future.

About Symantec

Symantec is a global leader in providing security, storage and systems management solutions to help businesses and consumers secure and manage their information. Headquartered in Cupertino, Calif., Symantec has operations in 40 countries. More information is available at www.symantec.com .