

Tutorial

Class Definition and Registry Discovery

Steve Wilson

3/13/2007

This tutorial is designed for the management pack novice and walks you through creating a new MP, a class to represent a Windows application and a discovery for this class. It also introduces the use of an XML editor and the use of the MP schema and MP Verify. This is MP Authoring 101.

Contents

Tutorial Overview.....	3
Get Ready to Start.....	3
Create the Manifest.....	3
Create the Language Pack.....	5
Create a Class.....	6
Create a Discovery	8
Update the Discovery.....	10
Add Properties to the Class.....	12
Add a View to the Console.....	14
Testing Discovery Updates.....	16
Deleting the Management Pack.....	16

Tutorial Overview

The purpose of this tutorial is to walk you through creating your first MP from start to finish. The result of this will be a fully functional MP with:

- A new application class with properties
- Discovery for class using the registry
- A view folder and state view

Subsequent tutorials will extend this MP to include monitors, rules, tasks, diagnostics, recoveries and plenty more.

Note that this tutorial does not explain all the concepts that are necessary to fully understand the MP you are creating. You should use the rest of the AuthorMPs site for that!

Get Ready to Start

This management pack will be created using XML. You can use any XML editor or even notepad if you really want. For this walkthrough I will use Visual Studio. You can get a number of Express versions of Visual Studio for free from Microsoft.com. Any edition will do. Get them here:

<http://msdn.microsoft.com/vstudio/express/downloads/default.aspx>

You also need to get the Management Pack schema files from Microsoft. These are currently available via the Microsoft Connect site for registered Beta testers and TAP partners. Once Operations Manager releases, these will be available publicly. Sorry I cannot share these on this site. Copy all the schema files into a directory somewhere on your hard drive. I will assume you use C:\OpsMgr\MPSchema. You need to keep the correct directory structure i.e. keep the maml and reporting directories. The MP schema references these directories and files.

The final step is to install the Operations Manager UI component on your workstation if it is not there (you need the SDK assemblies to perform MP Verify and import the MP using MP Import). You also should copy all the .MP files from your root management server to a directory on your file system. When MP Verify verifies your MP it needs to open the reference MPs to check that what you reference is actually in those MPs. I will assume you copy these to a directory called C:\OpsMgr\SystemLibraries.

Create the Manifest

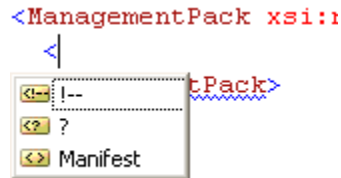
Open your chosen edition of Visual Studio and create a new XML file. Copy and paste the following into the editor to create the references to the MP schema and the xsi and xsd namespaces. You will start every management pack with this declaration.

```
<ManagementPack
xsi:noNamespaceSchemaLocation="C:\OpsMgr\MPSchema\ManagementPackSchema.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
</ManagementPack>
```

Now save the file as “**AuthorMPs.Tutorials.MyFirstMP.xml**” – I will assume you save it to C:\OpsMgr\MyMPs.

The first thing I will do is create the manifest section. This is the only required section of the management pack. If you move your cursor inside the Management Pack element and type a < character, Visual Studio smart tags will kick in and you will be given the available elements that you can put in here:



Since the schema defines the manifest section must be the first section, this is the only option. Select the manifest element and close the element with the > character. You will notice a blue underline indicating a schema error and your XML will now be:

```
<ManagementPack
xsi:noNamespaceSchemaLocation="C:\OpsMgr\MPSchema\ManagementPackSchema.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Manifest></Manifest>
</ManagementPack>
```

If you look in the error list (use View, Error List) you will see the following error:

“The element 'Manifest' has incomplete content. List of possible elements expected: 'Identity'”

Visual Studio validates the XML you type against the schema as you go. You should ensure you have no errors in the error list before you attempt to import the MP to Operations Manager. The first thing MP Import does is call MP Verify. The first thing MP Verify does is validate against the MP Schema. You can continue to type the XML yourself or cut and paste the manifest section in below:

```
<ManagementPack
xsi:noNamespaceSchemaLocation="C:\OpsMgr\MPSchema\ManagementPackSchema.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Manifest>
    <Identity>
      <ID>AuthorMPs.Tutorials.MyFirstMP</ID>
      <Version>1.0.0.0</Version>
    </Identity>
    <Name>My First MP</Name>
    <References>
      <Reference Alias="System">
        <ID>System.Library</ID>
        <Version>6.0.4941.0</Version>
      </Reference>
    </References>
  </Manifest>
</ManagementPack>
```

```

    <PublicKeyToken>31bf3856ad364e35</PublicKeyToken>
  </Reference>
  <Reference Alias="Windows">
    <ID>Microsoft.Windows.Library</ID>
    <Version>6.0.4941.0</Version>
    <PublicKeyToken>31bf3856ad364e35</PublicKeyToken>
  </Reference>
  <Reference Alias="SC">
    <ID>Microsoft.SystemCenter.Library</ID>
    <Version>6.0.4941.0</Version>
    <PublicKeyToken>31bf3856ad364e35</PublicKeyToken>
  </Reference>
</References>
</Manifest>
</ManagementPack>

```

The manifest defines the ID and version of the MP. Note that the ID has to be same as the filename (this is so MP Verify can find MPs on the file system when it needs to verify the MPs that you reference).

I have defined three references to some very common MPs. You will always reference these three MPs for any Windows monitoring you are doing. I will add additional references in later tutorials.

This is now a valid MP. Just to show you this, save the file and open a command prompt. Go to your Operations Manager install directory e.g. (C:\Program Files\System Center Operations Manager 2007) and type the following and hit return (change the directory names if you did not use the ones I suggested):

```
MPVerify.exe /I C:\OpsMgr\SystemMPs C:\OpsMgr\MyMPs\AuthorMPs.Tutorials.MyFirstMP.xml
```

This will open the MP you have created, validate against the schema and then run a series of verification tests to ensure your MP is valid. You should see the following output:

```

Loading management pack file: C:\OpsMgr\MyMPs\AuthorMPs.Tutorials.MyFirstMP.xml
Running verification tests for MP: AuthorMPs.Tutorials.MyFirstMP
Verification succeeded.

```

Congratulations you have created your first MP in XML. You could go and import this to Operations Manager. It would not do anything but you could now save MP objects and MP template output in this MP if you wanted.

You should get into the habit of running MP Verify as you add to your MP to avoid a whole bunch of errors later on. Leave the command window open so it is easy to rerun the command you just executed.

Create the Language Pack

The management pack schema is designed so that no localizable text is used in the majority of the management pack. Instead all localizable text e.g. names, descriptions, knowledge articles are kept in a separate section of the MP called the Language Pack. You can actually ship multiple languages in the same MP. The first thing you should do is create a name and description for the MP. You do this by adding a language packs section after the manifest closing element:

```

<LanguagePacks>
  <LanguagePack ID="ENU" IsDefault="true">
    <DisplayStrings>
      <DisplayString ElementID="AuthorMPs.Tutorials.MyFirstMP">
        <Name>AuthorMPs Tutorials - My first MP</Name>
        <Description>This is my first management pack created using a
tutorial from AuthorMPs.com</Description>
      </DisplayString>
    </DisplayStrings>
  </LanguagePack>
</LanguagePacks>

```

See how I refer back to an element ID, in this case the ID of the management pack. You will see how this continues as I create a new class. Save and run MP Verify on your updated MP to make sure you put things in the right place.

Create a Class

One of the very first things you normally do when designing an MP to monitor some custom application is to create a class to represent your application. I will start with the bare essentials then add some properties as I progress. This management pack is going to monitor a very simple application that I have running on some of my servers. This application is a standalone application that can be discovered by the existence of a registry key.

This application seems to fit nicely into a class already defined in the Windows.Library MP called the Windows Local Application class. This is an abstract class that I will specialize to create a new class. My application is called "Application Z". I create the class definition in a new section of the MP called Type Definitions that comes after the Manifest section but before the Display Strings section. I enter my definition as follows:

```

<TypeDefinitions>
  <EntityTypes>
    <ClassTypes>
      <ClassType ID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ"
Abstract="false" Base="Windows!Microsoft.Windows.LocalApplication"
Accessibility="Internal" Hosted="true"/>
    </ClassTypes>
  </EntityTypes>
</TypeDefinitions>

```

This defines a class with the following semantics

- An ID that is unique within my MP
- A specialization of the Windows Local Application class
- A non abstract class (there can be instances of this class)
- An internal class (if I was to seal this MP no other MPs could reference the class)
- The class is hosted by another class (instances cannot exist without a host)

Since the class specializes the Windows Local Application, it inherits a relationship stating that a Windows Computer hosts it. There is no need for me to define a new hosting relationship.

Just for fun lets actual make a mistake so I can show you the usefulness of MP Verify. Change the base class to something invalid:

```
Base="Windows!Microsoft.Windows.DoesNotExist"
```

Notice that there are no schema errors. This is valid against the schema since you have correctly specified a base class. However schema validation can only go so far. The only way this reference can be validated is to locate and open the MP referenced with the Windows alias and see if the class exists. This is where MP Verify comes in.

Save MP and run MPVerify. You will see the following:

```
Loading management pack file: C:\OpsMgr\MyMPs\AuthorMPs.Tutorials.MyFirstMP.xml
Running verification tests for MP: AuthorMPs.Tutorials.MyFirstMP
: Verification failed with [1] errors:
-----
Error 1:
: Failed to verify Class Type ID=AuthorMPs.Tutorials.MyFirstMP.ApplicationZ
Failed to verify Class Type ID=AuthorMPs.Tutorials.MyFirstMP.ApplicationZCannot find
ManagementPackElement [Type=ManagementPackClass, ID=Microsoft.Windows.DoesNotExist] in
ManagementPack [ManagementPack:[Name=Microsoft.Windows.Library,
KeyToken=31bf3856ad364e35, Version=6.0.4941.0]]
-----

Failed to verify Class Type ID=AuthorMPs.Tutorials.MyFirstMP.ApplicationZFailed to
verify Class Type ID=AuthorMPs.Tutorials.MyFirstMP.ApplicationZCannot find
ManagementPackElement [Type=ManagementPackClass, ID=Microsoft.Windows.DoesNotExist] in
ManagementPack [ManagementPack:[Name=Microsoft.Windows.Library,
KeyToken=31bf3856ad364e35, Version=6.0.4941.0]]
```

Some MP Verify error messages can take a little bit of getting used to. If you have multiple errors in the MP that will all be reported at once. Often one error can lead to other errors so you should fix the first error and try again. In this example there is one error – MP Verify cannot find the class I tried to use. Put the base class back to the correct version:

```
Base="Windows!Microsoft.Windows.LocalApplication"
```

If you run MP Verify there should be no more errors.

The final thing to do I want to do is add a display string for my class. Add the following display string to the language pack section you created earlier.

```
<DisplayString ElementID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ">
  <Name>Application Z</Name>
</DisplayString>
```

If you were to import the MP to Operations Manager now you could create MP Objects such as rules, tasks and monitors targeted to this new class. However they would never execute because we have not defined how to discover the class.

Create a Discovery

Now I have defined the class for my application, I need to discover it. To do this I create a discovery. For this example I am going to discover using the registry. I will create a discovery that is targeted to the Windows Server Computer class. This means the discovery will run on every Windows Server I am monitoring in my Operations Manager management group. An instance of the application will be created only where I find the required key. I am going to look for a registry key with the following path:

HKLM\Software\AuthorMPs\ApplicationZ

I suggest you don't create this key yet so you can make sure we don't discover the application when it does not exist. Now create a monitoring section in your MP file after the type definitions section and add a discovery:

```
<Monitoring>
  <Discoveries>
    <Discovery
ID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ.DiscoverApplication"
Enabled="true" Target="Windows!Microsoft.Windows.Server.Computer">
      <Category>Discovery</Category>
      <DiscoveryTypes/>
      <DataSource ID="DS"
TypeID="Windows!Microsoft.Windows.FilteredRegistryDiscoveryProvider">
        <ComputerName>$Target/Property[Type="Windows!Microsoft.Windows.Computer
"/NetworkName$</ComputerName>
          <RegistryAttributeDefinitions>
            <RegistryAttributeDefinition>
              <AttributeName>ApplicationZExists</AttributeName>
              <Path>SOFTWARE\AuthorMPs\ApplicationZ</Path>
              <PathType>0</PathType>
              <AttributeType>0</AttributeType>
            </RegistryAttributeDefinition>
          </RegistryAttributeDefinitions>
          <Frequency>30</Frequency>
        <ClassId>$MPElement[Name="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ"]$
</ClassId>
          <InstanceSettings>
            <Settings>
              <Setting>
                <Name>$MPElement[Name="Windows!Microsoft.Windows.Computer"]/PrincipalName$</Name>
                <Value>$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/PrincipalName$</Value>
              </Setting>
            </Settings>
          </InstanceSettings>
          <Expression>
            <SimpleExpression>
```

```

        <ValueExpression>
            <XPathQuery
Type="Boolean">Values/ApplicationZExists</XPathQuery>
            </ValueExpression>
            <Operator>Equal</Operator>
            <ValueExpression>
                <Value Type="Boolean">>true</Value>
            </ValueExpression>
        </SimpleExpression>
    </Expression>
</DataSource>
</Discovery>
</Discoveries>
</Monitoring>

```

This particular discovery uses the registry provider. This module requires configuration to specify:

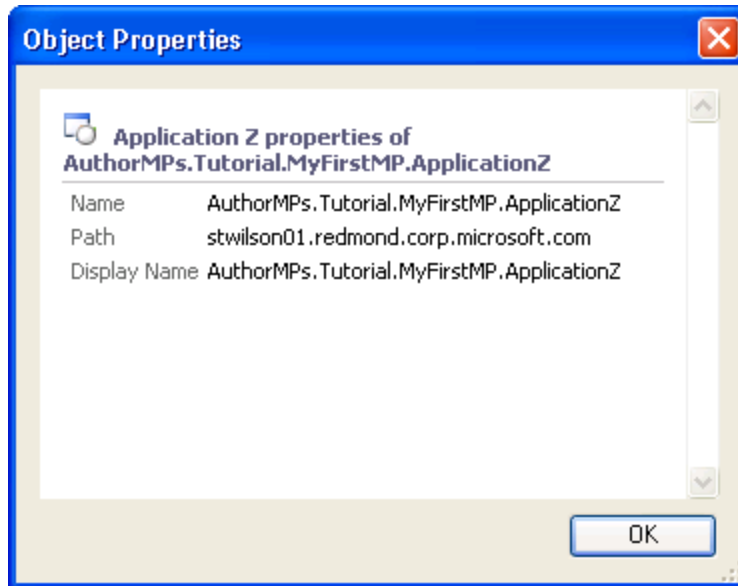
- The frequency to execute on (note I have set this very low – 30 seconds)
- The keys to look for
- The class to discovery
- How to map discovered data to a class instance

Different discoveries will have different configuration.

Save your MP and now import it to an Operations Manager group. Now the first thing we want to check is that an instance of a class is not discovered since there is no registry key present. Wait for configuration to be sent to the agent or Ops Mgr server you are testing on. Now open the Operations Console and go to the Discovered Inventory view in the monitoring space. This is a view that shows you all discovered instances of a particular type.

Change the target type (right click in the view and use the menu or use the Actions pane). Search for the **Application Z** class in the list, select it and click OK. You should not see any objects in the list view since nothing has been discovered.

Now create a registry key called **HKLM\Software\AuthorMPs\ApplicationZ** on your test agent or server. Wait for at least 30 seconds and refresh the view. You should see an instance of your class appear. If you bring up the properties of this object you will see something like:



You could now start targeting rules, monitors, tasks and more to this class and they would run everywhere the application is discovered.

Update the Discovery

We want to make things look a little better in the Operations Console and also let the user find the discovery in the object discoveries view. First thing we will do is add a display string for the Discovery and the data source module within the discovery. Add the following two display strings.

```
<DisplayString
ElementID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ.DiscoverApplication">
  <Name>Discover Application Z on Windows Servers</Name>
</DisplayString>
<DisplayString
ElementID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ.DiscoverApplication"
SubElementID="DS">
  <Name>Registry Probe</Name>
</DisplayString>
```

As you can see by the object properties screen shot we should tidy up the display strings. Every class inherits a property from the System.Entity class called DisplayName. When you discover objects you should set this property and the Operations Console will use this everywhere it shows the object.

Update the instance settings section of the discovery so it now read:

```
<InstanceSettings>
  <Settings>
    <Setting>
      <Name>$MPElement[Name="Windows!Microsoft.Windows.Computer"]/PrincipalName$</Name>
```

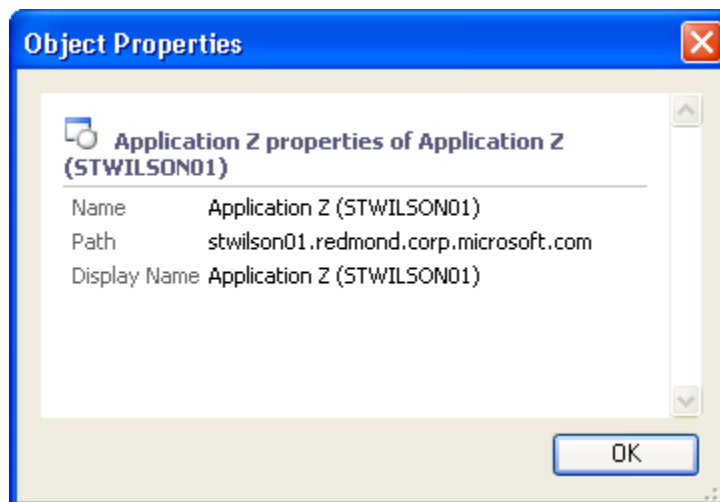
```

    <Value>$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/PrincipalName$</Value>
  </Setting>
</Setting>

  <Name>$MPElement[Name="System!System.Entity"]/DisplayName$</Name>
  <Value>Application Z
($Target/Property[Type="Windows!Microsoft.Windows.Computer"]/NetbiosComputerName$) </Value>
</Setting>
</Settings>
</InstanceSettings>

```

The display name of the object will now be **Application Z (computername)** where computername will be the NetBIOS name of the computer it is discovered on. The properties will now look like this:



The discovery has a section called Discovery Types that declares the classes and relationships that can be discovered by the discovery. This meta data is used by the Operations Console so you can find discoveries for particular types of object.

Everytime you crate a discovery you should delare the types you will discover. I will change the Discovered Types section to:

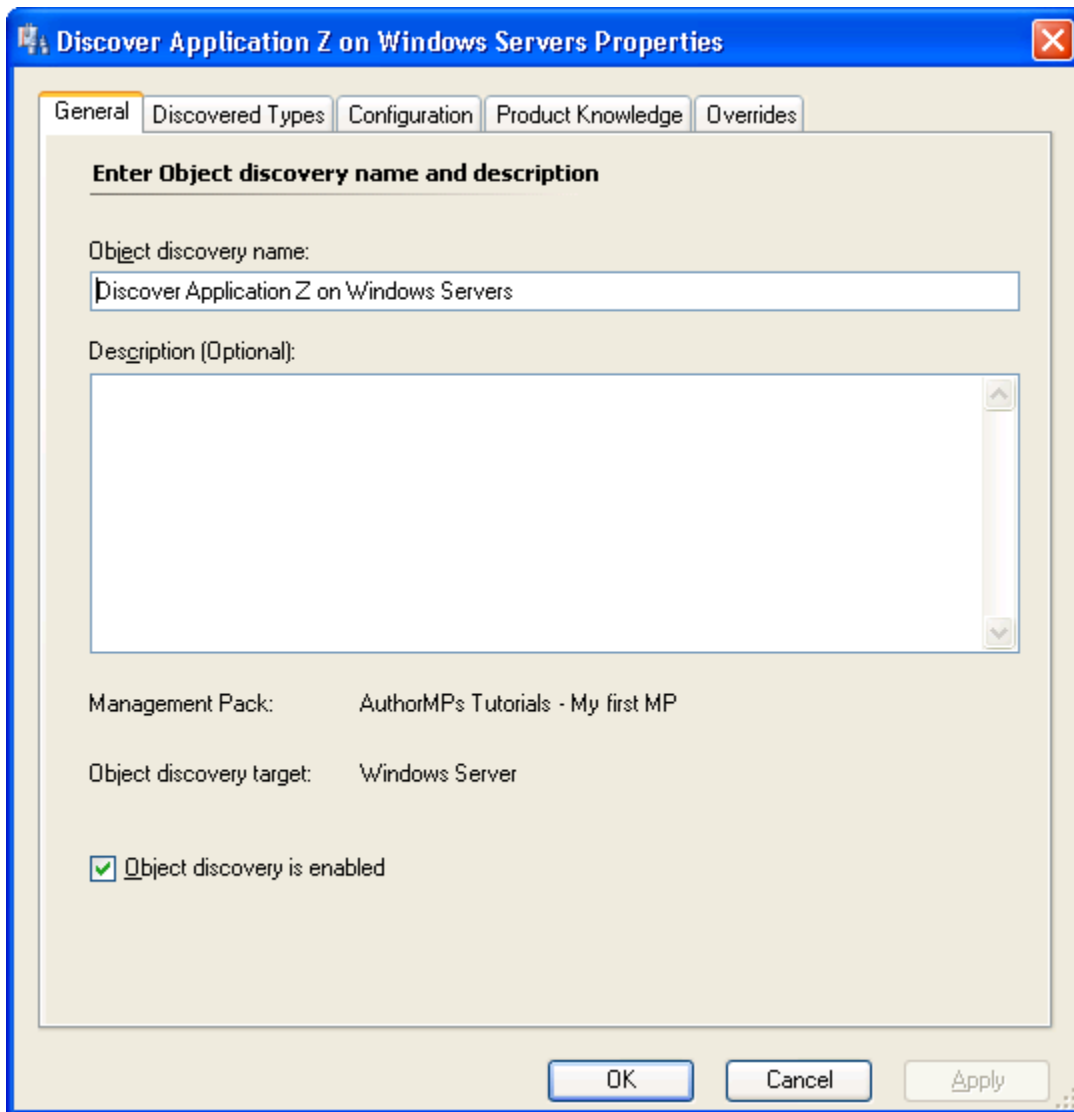
```

<DiscoveryTypes>
  <DiscoveryClass TypeID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ">
    <Property TypeID="System!System.Entity"
PropertyID="DisplayName"/>
  </DiscoveryClass>
  <DiscoveryRelationship
TypeID="Windows!Microsoft.Windows.ComputerHostsLocalApplication"/>

```

</DiscoveryTypes>

If you import the new MP to your management group you could now go into the authoring space of the console, go to the object discoveries view and scope to the Application Z class. You would now see the discovery listed and you could bring up the properties:



Add Properties to the Class

I will now add two properties to the class and update the discovery to populate these from registry values in the registry key I created before. The first step is to update my class definition to add two properties:

```

<ClassType ID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ" Abstract="false"
Base="Windows!Microsoft.Windows.LocalApplication" Accessibility="Internal"
Hosted="true">
  <Property ID="Version" Type="string"/>
  <Property ID="InstallPath" Type="string"/>
</ClassType>

```

I should also add some display strings for these properties in the language pack:

```

<DisplayString ElementID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ"
SubElementID="Version">
  <Name>Version</Name>
</DisplayString>
<DisplayString ElementID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ"
SubElementID="InstallPath">
  <Name>Install Path</Name>
</DisplayString>

```

Now I will update the discovery to look for two registry values. Change the registry attribute definition to the following:

```

<RegistryAttributeDefinitions>
  <RegistryAttributeDefinition>
    <AttributeName>ApplicationZExists</AttributeName>
    <Path>SOFTWARE\AuthorMPs\ApplicationZ</Path>
    <PathType>0</PathType>
    <AttributeType>0</AttributeType>
  </RegistryAttributeDefinition>
  <RegistryAttributeDefinition>
    <AttributeName>ApplicationZVersion</AttributeName>
    <Path>SOFTWARE\AuthorMPs\ApplicationZ\Version</Path>
    <PathType>1</PathType>
    <AttributeType>1</AttributeType>
  </RegistryAttributeDefinition>
  <RegistryAttributeDefinition>
    <AttributeName>ApplicationZPath</AttributeName>
    <Path>SOFTWARE\AuthorMPs\ApplicationZ\Path</Path>
    <PathType>1</PathType>
    <AttributeType>1</AttributeType>
  </RegistryAttributeDefinition>
</RegistryAttributeDefinitions>

```

Now update the instance settings of the discovery to map these new attributes to class properties:

```

<InstanceSettings>
  <Settings>
    <Setting>
      <Name>$MPElement [Name="Windows!Microsoft.Windows.Computer"]/PrincipalName$</Name>
      <Value>$Target/Property [Type="Windows!Microsoft.Windows.Computer"]/PrincipalName$</Value>
    </Setting>
  </Settings>
</InstanceSettings>

```

```

    <Name>$MPElement [Name="System!System.Entity"]/DisplayName$</Name>
        <Value>Application Z
($Target/Property [Type="Windows!Microsoft.Windows.Computer"]/NetbiosComputerName$) </Value>
    </Setting>
</Setting>

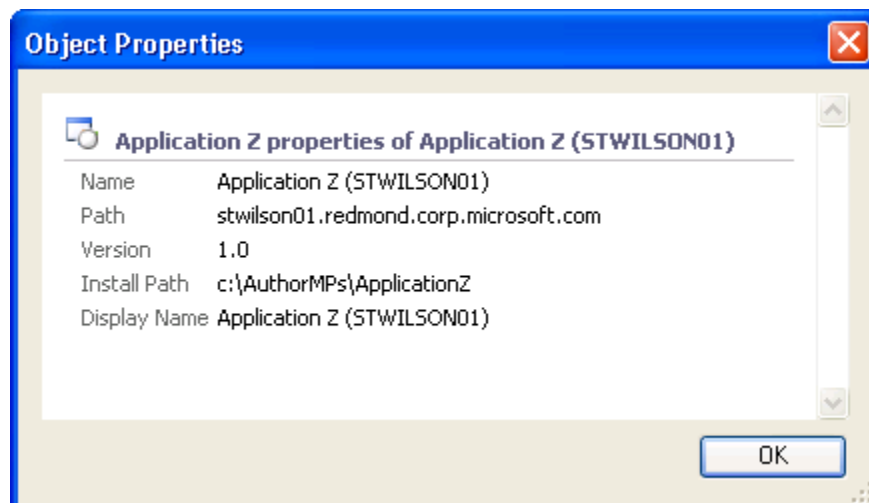
    <Name>$MPElement [Name="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ"]/Version$</Name>
        <Value>$Data/Values/ApplicationZVersion$</Value>
    </Setting>
</Setting>

    <Name>$MPElement [Name="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ"]/InstallPath$</Name>
        <Value>$Data/Values/ApplicationZPath$</Value>
    </Setting>
</Settings>
</InstanceSettings>

```

Now add two string values to the ApplicationZ registry key. One is called Version and the other Path. Populate these with some values.

You can now import the new MP and wait a few minutes for the updated discovery to run. Once this has run you should see the updated properties discovered:



Add a View to the Console

The final piece I want to add to this MP is a state view so that I don't have to go to the discovered inventory view to see objects I have discovered. I will create two new elements here – a folder for my MP and then the state view itself.

It is common practice for a management pack to introduce a new folder into the monitoring folder structure. This is done by defining a Folder object in your management pack. This folder must reference

a specific Folder in the Microsoft.SystemCenter.Library MP as its parent folder. If you do not add this reference, the folder will never show up in the Monitoring folder tree.

The Folder object goes into a presentation section of your management pack. This section comes after the monitoring section but before the language packs section. You should define the following:

```
<Presentation>
  <Folders>
    <Folder ID="AuthorMPs.Tutorials.MyFirstMP.ViewFolder"
Accessibility="Internal"
ParentFolder="SC!Microsoft.SystemCenter.Monitoring.ViewFolder.Root"/>
  </Folders>
</Presentation>
```

You also need a display string for this folder:

```
<DisplayString ElementID="AuthorMPs.Tutorials.MyFirstMP.ViewFolder">
  <Name>AuthorMPs Tutorial - My First MP</Name>
</DisplayString>
```

Now I need to add a state view for my application. This will show all discovered objects of my new class. To define a state view I create a View object in the Presentation section of the MP. Add the following to the presentation section before the folders section you defined before:

```
<Views>
  <View ID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ.State"
Accessibility="Internal" Target="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ"
TypeID="SC!Microsoft.SystemCenter.StateViewType">
    <Category>Operations</Category>
  </View>
</Views>
```

This defines a state view with no criteria. This will be fine for this demonstration. Don't forget a display string for this view:

```
<DisplayString ElementID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ.State">
  <Name>Application Z</Name>
  <Description>All my Application Z instances</Description>
</DisplayString>
```

The final step is to add the view to the folder I defined. This is done using a Folder Item element. Folder items come after the Folders section of the Presentation section. A folder item points to an MP element and an MP folder. The complete presentation section should now be:

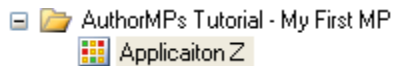
```
<Presentation>
  <Views>
    <View ID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ.State"
Accessibility="Internal" Target="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ"
TypeID="SC!Microsoft.SystemCenter.StateViewType">
      <Category>Operations</Category>
    </View>
  </Views>
  <Folders>
```

```

        <Folder ID="AuthorMPs.Tutorials.MyFirstMP.ViewFolder"
Accessibility="Internal"
ParentFolder="SC!Microsoft.SystemCenter.Monitoring.ViewFolder.Root"/>
    </Folders>
    <FolderItems>
        <FolderItem
ElementID="AuthorMPs.Tutorials.MyFirstMP.ApplicationZ.State"
Folder="AuthorMPs.Tutorials.MyFirstMP.ViewFolder"/>
    </FolderItems>
</Presentation>

```

OK let's try it out. You should import the finished MP. You should now see a new folder in your monitoring space:



The state view will show all instances of the Application Z class.

Testing Discovery Updates

Suppose someone uninstalls the Application Z on one of your computers. You would expect that Operations Manager will no longer monitor the application since it is not there. What will happen is that the discovery will run on the next polling interval (30 seconds in our case) and it will not find the registry key. It will submit an empty discovery packet which means the application is no longer there. When this gets to the database the object is removed.

You can test this out by deleting (or renaming) the Application Z registry key you created at the start of this tutorial. Once you have done this wait a minute or so and refresh your Application Z state view. You should see the object disappear.

Deleting the Management Pack

Once you are finished testing your management pack you can delete it from the Management Packs view of the Administration space by finding the MP, right clicking and selecting delete. Note this was not possible in MOM 2005! This will remove all the definitions you made in your MP and will remove any instances of the Application Z class.