

Tutorial

Using Abstract Classes

Steve Wilson

3/16/2007

This tutorial is designed to introduce you to the concept of an abstract class through by walking through a full MP example. Discovery , monitoring and views are all considered to show you how things work and how they appear in the Operations console.

Contents

Tutorial Overview.....	3
Create the Manifest.....	3
Create the Abstract Class.....	4
Create the non Abstract Classes.....	5
Create the Discoveries.....	6
Create State Views.....	12
Test Discovery.....	13
Add Generic Monitoring.....	15
Add Specific Monitoring.....	16

Tutorial Overview

The purpose of this tutorial is to walk through a full MP example using an abstract class to define common properties and monitoring between different types of application. The result of this tutorial will be a fully functional MP with:

- An abstract class that defines common properties
- Two specialized classes that extend the abstract class
- Registry discoveries for each non abstract class
- State views
- Common monitoring targeted at the base class
- Specific monitoring targeted at the specialized classes

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!

This tutorial assumes you have some knowledge of creating a management pack in XML. You should read the **Class Definition and Registry Discovery** tutorial for information about setting up your XML editor and building your first MP.

Create the Manifest

Open your chosen edition of Visual Studio and create a new XML file. Create your standard management pack root element:

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

Create the manifest section as shown below:

```
<Manifest>
  <Identity>
    <ID>AuthorMPs.Tutorials.UsingAbstractClasses</ID>
    <Version>1.0.0.0</Version>
  </Identity>
  <Name>AuthorMPs Demo MP - Using Abstract Classes</Name>
  <References>
    <Reference Alias="System">
      <ID>System.Library</ID>
      <Version>6.0.4941.0</Version>
      <PublicKeyToken>31bf3856ad364e35</PublicKeyToken>
    </Reference>
    <Reference Alias="SystemHealth">
      <ID>System.Health.Library</ID>
      <Version>6.0.4941.0</Version>
      <PublicKeyToken>31bf3856ad364e35</PublicKeyToken>
    </Reference>
  </References>
</Manifest>
```

```

    <Reference Alias="SystemPerf">
      <ID>System.Performance.Library</ID>
      <Version>6.0.4941.0</Version>
      <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>

```

You should also create the language pack and set the display string for the MP:

```

<LanguagePack ID="ENU" IsDefault="true">
  <DisplayStrings>
    <DisplayString
      ElementID="AuthorMPs.Tutorials.UsingAbstractClasses">
      <Name>AuthorMPs Demo MP - Using Abstract Classes</Name>
      <Description>This MP demonstrates how to use an abstract
class. The MP defines an abstract base class with common properties and then
specializes this into two different concrete classes.</Description>
    </DisplayString>
  </DisplayStrings>
</LanguagePack>

```

Now save the file as “**AuthorMPs.Tutorials.UsingAbstractClasses.xml**” .

Create the Abstract Class

You will create an abstract class that is representative of a certain type of Windows application. Let’s assume you work in a company that has lots of in house developed applications. These applications have some common properties such as version and install directory. Also you want to be able to see a list of all you applications regardless of the specific type of application and you want to apply some common monitoring to all applications.

I will create a class that inherits from the Windows Local Application class and is set as abstract. On this class I will define three properties that are common across all of my applications – version, install path and process name. Copy the following definition into your management pack in between your manifest and language pack:

```

<TypeDefinitions>
  <EntityTypes>
    <ClassTypes>

```

```

        <ClassType
ID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass" Abstract="true"
Accessibility="Internal" Hosted="true"
Base="Windows!Microsoft.Windows.LocalApplication">
        <Property ID="Version" Type="string"/>
        <Property ID="InstallPath" Type="string"/>
        <Property ID="ProcessName" Type="string"/>
    </ClassType>
</ClassTypes>
</EntityTypes>
</TypeDefinitions>

```

I will also add some display strings for the class and each property in the language pack I created before:

```

<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass">
    <Name>AuthorMPs Demo - Application Base Class</Name>
</DisplayString>
<DisplayString ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"
SubElementID="Version">
    <Name>Version</Name>
</DisplayString>
<DisplayString ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"
SubElementID="InstallPath">
    <Name>Install Path</Name>
</DisplayString>
<DisplayString ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"
SubElementID="ProcessName">
    <Name>Process Name</Name>
</DisplayString>

```

Now I have the abstract class created. I do not create any discovery for this class because I cannot have instances of an abstract class. This class is a template for other application classes and will be used for some monitoring targeting as well.

Create the non Abstract Classes

I will define two application classes that inherit from my new abstract class. I will call these Application A and Application B. Each one will add a new property to add to the properties they inherit from the abstract class. My class definitions are:

```

<ClassType ID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA"
Abstract="false" Accessibility="Internal" Hosted="true"
Base="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass">
    <Property ID="ServiceName" Type="string"/>
</ClassType>
<ClassType ID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationB"
Abstract="false" Accessibility="Internal" Hosted="true"
Base="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass">
    <Property ID="LogPath" Type="string"/>
</ClassType>

```

Application A is an application that runs as a Windows service so I add a property to contain the service name. Application B has a log file that I may want to monitor at some point. I add a property to store the directory for this log file.

I also specify the display strings for each class and property:

```
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA">
  <Name>AuthorMPs Demo - Application A</Name>
</DisplayString>
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA"
SubElementID="ServiceName">
  <Name>Service Name</Name>
</DisplayString>
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationB">
  <Name>AuthorMPs Demo - Application B</Name>
</DisplayString>
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationB"
SubElementID="LogPath">
  <Name>Log Path</Name>
</DisplayString>
```

Now I have all the classes I required defined. The next step is to add a discovery for each non-abstract class.

Create the Discoveries

I will discover both Application A and Application B from the registry. Before testing you management pack you need to create some registry keys and values. Create the following keys:

- HKLM\Software\AuthorMPs\ApplicationA
- HKLM\Software\AuthorMPs\ApplicationB

Create the following string values:

- HKLM\Software\AuthorMPs\ApplicationA\Version – set to 1.0
- HKLM\Software\AuthorMPs\ApplicationA\Path – set to C:\ApplicationA
- HKLM\Software\AuthorMPs\ApplicationB\Version – set to 2.0
- HKLM\Software\AuthorMPs\ApplicationB\Path – set to C:\ApplicationB
- HKLM\Software\AuthorMPs\ApplicationB\LogPath – set to C:\ApplicationB\Log

The discoveries check for the existence of the appropriate application key and then map the values to the class properties. Rather than discover the process name for both classes and the service name for

Application A I have hardcoded the value in the discovery mapper since I know this will always be the same. Note I have used some common processes and services so I can easily set up some test monitoring for the applications.

The discoveries go in the monitoring section of the management pack which comes after the type definitions section. You should copy and paste the following:

```
<Monitoring>
  <Discoveries>
    <Discovery
ID="AuthorMPs.Tutorials.UsingAbstractClasses.DiscoverApplicationA"
Target="Windows!Microsoft.Windows.Server.Computer" Enabled="true">
      <Category>Discovery</Category>
      <DiscoveryTypes>
        <DiscoveryClass
TypeID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA">
          <Property PropertyID="ServiceName"/>
        </Property>
TypeID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"
PropertyID="Version"/>
          <Property
TypeID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"
PropertyID="InstallPath"/>
          <Property
TypeID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"
PropertyID="ProcessName"/>
          <Property TypeID="System!System.Entity"
PropertyID="DisplayName"/>
        </DiscoveryClass>
        <DiscoveryRelationship
TypeID="Windows!Microsoft.Windows.ComputerHostsLocalApplication"/>
      </DiscoveryTypes>
      <DataSource ID="DS"
TypeID="Windows!Microsoft.Windows.FilteredRegistryDiscoveryProvider">
        <ComputerName>$Target/Property[Type="Windows!Microsoft.Windows.Computer
"/NetworkName$</ComputerName>
          <RegistryAttributeDefinitions>
            <RegistryAttributeDefinition>
              <AttributeName>ApplicationAExists</AttributeName>
              <Path>SOFTWARE\AuthorMPs\ApplicationA</Path>
              <PathType>0</PathType>
              <AttributeType>0</AttributeType>
            </RegistryAttributeDefinition>
            <RegistryAttributeDefinition>
              <AttributeName>ApplicationAVersion</AttributeName>
              <Path>SOFTWARE\AuthorMPs\ApplicationA\Version</Path>
              <PathType>1</PathType>
              <AttributeType>1</AttributeType>
            </RegistryAttributeDefinition>
          </RegistryAttributeDefinitions>
        </DataSource>
      </Discovery>
    </Discoveries>
  </Monitoring>
```

```

<AttributeName>ApplicationAPath</AttributeName>

<Path>SOFTWARE\AuthorMPs\ApplicationA\Path</Path>
    <PathType>1</PathType>
    <AttributeType>1</AttributeType>
    </RegistryAttributeDefinition>
</RegistryAttributeDefinitions>
<Frequency>60</Frequency>

<ClassId>$MPElement [Name="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA"]$</ClassId>
    <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 A
                ($Target/Property [Type="Windows!Microsoft.Windows.Computer"]/NetbiosComputerName$) </Value>

                </Setting>
            <Setting>

                <Name>$MPElement [Name="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"]/Version$</Name>

                <Value>$Data/Values/ApplicationAVersion$</Value>

                </Setting>
            <Setting>

                <Name>$MPElement [Name="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"]/InstallPath$</Name>

                <Value>$Data/Values/ApplicationAPath$</Value>

                </Setting>
            <Setting>

                <Name>$MPElement [Name="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"]/ProcessName$</Name>

                <Value>spoolsv</Value>

                </Setting>
            <Setting>

                <Name>$MPElement [Name="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA"]/ServiceName$</Name>

                <Value>spooler</Value>

                </Setting>
        </Settings>
    </InstanceSettings>

```

```

        <Expression>
            <SimpleExpression>
                <ValueExpression>
                    <XPathQuery
Type="Boolean">Values/ApplicationAExists</XPathQuery>
                    </ValueExpression>
                    <Operator>Equal</Operator>
                    <ValueExpression>
                        <Value Type="Boolean">>true</Value>
                    </ValueExpression>
                </SimpleExpression>
            </Expression>
        </DataSource>
    </Discovery>
    <Discovery
ID="AuthorMPs.Tutorials.UsingAbstractClasses.DiscoverApplicationB"
Target="Windows!Microsoft.Windows.Server.Computer" Enabled="true">
        <Category>Discovery</Category>
        <DiscoveryTypes>
            <DiscoveryClass
TypeID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationB">
                <Property PropertyID="LogPath"/>
            </DiscoveryClass>
            <DiscoveryClass
TypeID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"
PropertyID="Version"/>
                <Property
TypeID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"
PropertyID="InstallPath"/>
                <Property
TypeID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"
PropertyID="ProcessName"/>
                <Property TypeID="System!System.Entity"
PropertyID="DisplayName"/>
            </DiscoveryClass>
        </DiscoveryTypes>
    </DiscoveryRelationship>
    </DiscoveryTypes>
    <DataSource ID="DS"
TypeID="Windows!Microsoft.Windows.FilteredRegistryDiscoveryProvider">
        <ComputerName>$Target/Property[Type="Windows!Microsoft.Windows.Computer
"/NetworkName$</ComputerName>
            <RegistryAttributeDefinitions>
                <RegistryAttributeDefinition>
                    <AttributeName>ApplicationBExists</AttributeName>
                    <Path>SOFTWARE\AuthorMPs\ApplicationB</Path>
                    <PathType>0</PathType>
                    <AttributeType>0</AttributeType>
                </RegistryAttributeDefinition>
                <RegistryAttributeDefinition>
                    <AttributeName>ApplicationBVersion</AttributeName>
                    <Path>SOFTWARE\AuthorMPs\ApplicationB\Version</Path>
                    <PathType>1</PathType>

```

```

        <AttributeType>1</AttributeType>
    </RegistryAttributeDefinition>
</RegistryAttributeDefinition>

<AttributeName>ApplicationBPath</AttributeName>

<Path>SOFTWARE\AuthorMPs\ApplicationB\Path</Path>
    <PathType>1</PathType>
    <AttributeType>1</AttributeType>
</RegistryAttributeDefinition>
</RegistryAttributeDefinition>

<AttributeName>ApplicationBLogPath</AttributeName>

<Path>SOFTWARE\AuthorMPs\ApplicationB\LogPath</Path>
    <PathType>1</PathType>
    <AttributeType>1</AttributeType>
</RegistryAttributeDefinition>
</RegistryAttributeDefinitions>
<Frequency>60</Frequency>

<ClassId>$MPElement [Name="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationB"]$</ClassId>
    <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 B
                ($Target/Property [Type="Windows!Microsoft.Windows.Computer"]/NetbiosComputerName$) </Value>

                </Setting>
                <Setting>

                <Name>$MPElement [Name="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"]/Version$</Name>

                <Value>$Data/Values/ApplicationBVersion$</Value>

                </Setting>
                <Setting>

                <Name>$MPElement [Name="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"]/InstallPath$</Name>

                <Value>$Data/Values/ApplicationBPath$</Value>

                </Setting>
                <Setting>

```

```

    <Name>$MPElement [Name="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"]/ProcessName$</Name>
        <Value>lsass</Value>
    </Setting>
</Setting>

    <Name>$MPElement [Name="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationB"]/LogPath$</Name>

    <Value>$Data/Values/ApplicationBLogPath$</Value>
        </Setting>
    </Settings>
</InstanceSettings>
<Expression>
    <SimpleExpression>
        <ValueExpression>
            <XPathQuery
Type="Boolean">Values/ApplicationBExists</XPathQuery>
            </ValueExpression>
            <Operator>Equal</Operator>
            <ValueExpression>
                <Value Type="Boolean">>true</Value>
            </ValueExpression>
        </SimpleExpression>
    </Expression>
</DataSource>
</Discovery>
</Discoveries>
</Monitoring>

```

As always let's add the display strings for these discoveries:

```

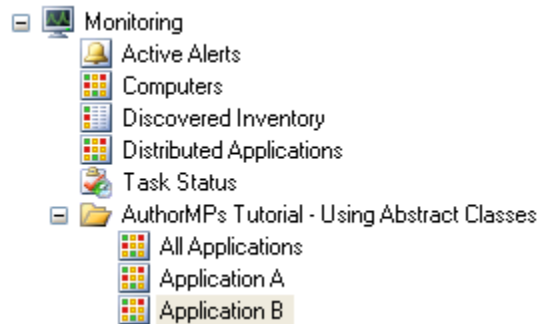
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.DiscoverApplicationA">
    <Name>Discover Application A on Windows Servers</Name>
</DisplayString>
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.DiscoverApplicationA"
SubElementID="DS">
    <Name>Registry Probe</Name>
</DisplayString>
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.DiscoverApplicationB">
    <Name>Discover Application B on Windows Servers</Name>
</DisplayString>
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.DiscoverApplicationB"
SubElementID="DS">
    <Name>Registry Probe</Name>
</DisplayString>

```

You could now import the management pack and the applications would be discovered on Windows Servers that have the appropriate registry keys. However before you do I will create a few views to make it easier to see the discovered applications.

Create State Views

I will create three state views. One will be targeted at the abstract base class, one targeted at Application A and one targeted at Application B. I will set the view structure so that it appears in the Operations Console as follows:



I do this by creating a folder, three state views and folder items to add the views in to the folder. The following presentation section should go into your management pack after the monitoring section:

```
<Presentation>
  <Views>
    <View
      ID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA.State"
      Accessibility="Internal"
      Target="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA"
      TypeID="SC!Microsoft.SystemCenter.StateViewType">
        <Category>Operations</Category>
      </View>
    <View
      ID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationB.State"
      Accessibility="Internal"
      Target="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationB"
      TypeID="SC!Microsoft.SystemCenter.StateViewType">
        <Category>Operations</Category>
      </View>
    <View
      ID="AuthorMPs.Tutorials.UsingAbstractClasses.AllApplications.State"
      Accessibility="Internal"
      Target="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"
      TypeID="SC!Microsoft.SystemCenter.StateViewType">
        <Category>Operations</Category>
      </View>
    </Views>
  <Folders>
    <Folder ID="AuthorMPs.Tutorials.UsingAbstractClasses.ViewFolder"
      Accessibility="Internal"
      ParentFolder="SC!Microsoft.SystemCenter.Monitoring.ViewFolder.Root"/>
  </Folders>
</Presentation>
```

```

    </Folders>
    <FolderItems>
      <FolderItem
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA.State"
Folder="AuthorMPs.Tutorials.UsingAbstractClasses.ViewFolder"/>
      <FolderItem
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationB.State"
Folder="AuthorMPs.Tutorials.UsingAbstractClasses.ViewFolder"/>
      <FolderItem
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.AllApplications.State"
Folder="AuthorMPs.Tutorials.UsingAbstractClasses.ViewFolder"/>
    </FolderItems>
  </Presentation>

```

The display strings should be:

```

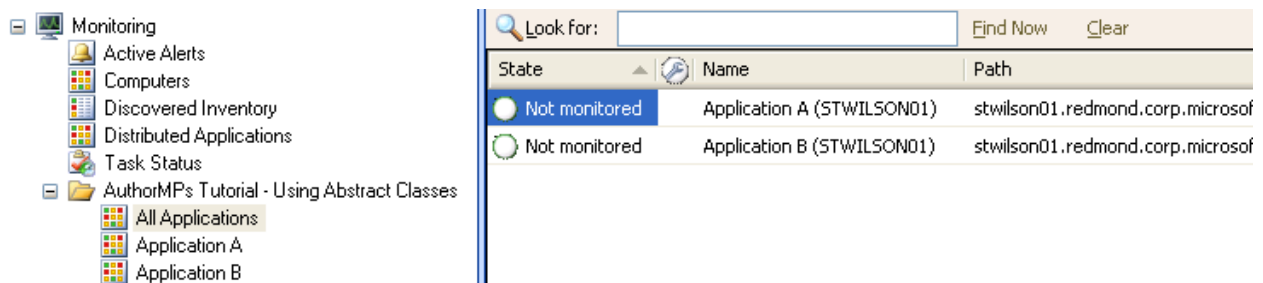
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.ViewFolder">
  <Name>AuthorMPs Tutorial - Using Abstract Classes</Name>
</DisplayString>
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA.State">
  <Name>Application A</Name>
  <Description>All my Application A instances</Description>
</DisplayString>
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationB.State">
  <Name>Application B</Name>
  <Description>All my Application B instances</Description>
</DisplayString>
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.AllApplications.State">
  <Name>All Applications</Name>
  <Description>All my Application instances</Description>
</DisplayString>

```

Test Discovery

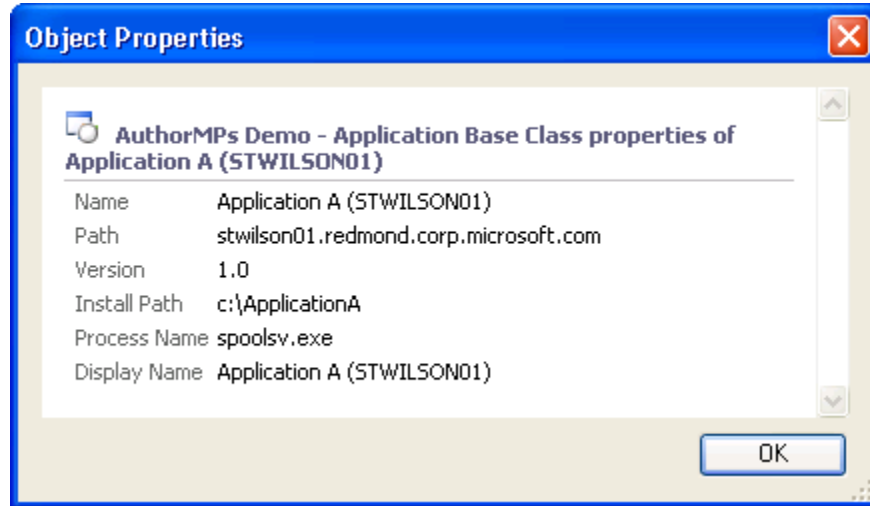
Ok we are ready to test the MP out. Save the management pack, ensure you have set the registry keys on your test server (you can use more than one server if you want to discover more than one instance of each application). Import the management pack and wait for discovery to happen.

If you select the **All Applications** view (this view is targeted at the abstract class) you should instances of both Application A and Application B:

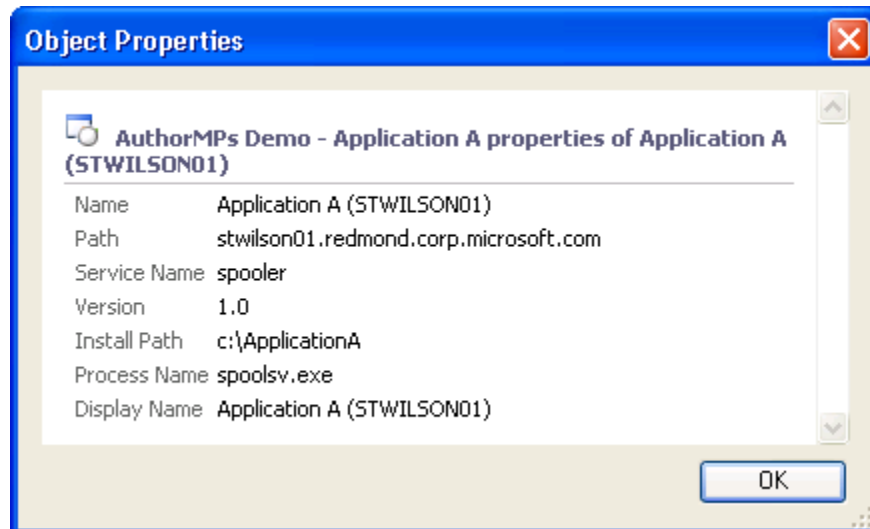


If you select the **Application A** view you will see only Application A instances, and not surprisingly if you select the **Application B** view you will see Application B instances.

You should note that if you bring up the object properties of an Application object from the **All Applications** view you only see properties defined on the base class:



If you bring up the properties from the **Application A** view you will also see the additional service name property that I defined:



Add Generic Monitoring

So far I have walked through the discovery process for the two applications. Now let's suppose I want to add some monitoring that is applicable to all my applications. Since I have defined a process name property on the base abstract class I am going to use this to monitor the memory utilization of the process for each application.

I can create a rule and target it to the abstract class I created. Every class that inherits from this base class will get this rule. I will define a basic performance collection rule. The rule must go in the monitoring section of your MP after the discoveries element:

```
<Rules>
  <Rule
    ID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass.ProcessPrivateBytes"
    Enabled="true" Target="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass">
    <Category>PerformanceHealth</Category>
    <DataSources>
      <DataSource ID="DS"
        TypeID="SystemPerf!System.Performance.OptimizedDataProvider">
        <ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>
        <CounterName>Private Bytes</CounterName>
        <ObjectName>Process</ObjectName>

        <InstanceName>$Target/Property[Type="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"]/ProcessName$</InstanceName>
        <AllInstances>>false</AllInstances>
        <Frequency>60</Frequency>
        <Tolerance>0</Tolerance>
        <MaximumSampleSeparation>1</MaximumSampleSeparation>
      </DataSource>
    </DataSources>
    <WriteActions>
      <WriteAction ID="WriteToDB"
        TypeID="SC!Microsoft.SystemCenter.CollectPerformanceData"/>
    </WriteActions>
  </Rule>
</Rules>
```

Notice how I use the process name property for the configuration of the rule. This will be dependent on the application that is being monitored. Don't forget the display string:

```
<DisplayString
  ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass.ProcessPrivateBytes">
  <Name>Collect Process Private Bytes</Name>
</DisplayString>
```

Now to make it easy to see the collected data let's add one more view:

```
<View
  ID="AuthorMPs.Tutorials.UsingAbstractClasses.AllApplications.Performance"
  Accessibility="Internal"
```

```

Target="AuthorMPs.Tutorials.UsingAbstractClasses.BaseClass"
TypeID="SC!Microsoft.SystemCenter.PerformanceViewType">
  <Category>Operations</Category>
</View>

```

Add a folder item to put it in the correct folder:

```

<FolderItem
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.AllApplications.Performance" Folder="AuthorMPs.Tutorials.UsingAbstractClasses.ViewFolder"/>

```

Finally add the view display string:

```

<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.AllApplications.Performance">
  <Name>All Performance</Name>
</DisplayString>

```

Now import the updated management pack. You will need to wait a few minutes for performance data to start to get collected. Open the new performance view and you should see a counter for every instance of Application A and B:

Show	Color	Path	Target	Rule	Object	Counter	Instance	Scale	Baseline
<input checked="" type="checkbox"/>	Blue	stwilson01.redm...	Application A (STWILSON01)		Process	Private Bytes	spoolsv	1x	No
<input checked="" type="checkbox"/>	Green	stwilson01.redm...	Application B (STWILSON01)		Process	Private Bytes	lsass	1x	No

Add Specific Monitoring

Some monitoring may only be applicable to specific applications. As an example, I will add a monitor targeted just to Application A. This monitor will monitor the state of the Windows service that Application A uses. Since Application B does not use a service it would not be appropriate to define it on the abstract class.

I am going to add a basic service check monitor to the Application A health model. I define the following in the monitoring section after the rules element:

```

<Monitors>
  <UnitMonitor
ID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA.ServiceState"
Accessibility="Internal"
ParentMonitorID="SystemHealth!System.Health.AvailabilityState"
Target="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA"
TypeID="Windows!Microsoft.Windows.CheckNTServiceStateMonitorType">
    <Category>AvailabilityHealth</Category>
    <OperationalStates>

```

```

        <OperationalState ID="Running" HealthState="Success"
MonitorTypeStateID="Running"/>
        <OperationalState ID="NotRunning" HealthState="Error"
MonitorTypeStateID="NotRunning"/>
    </OperationalStates>
</Configuration>

    <ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Com
puter"]/NetworkName$</ComputerName>

    <ServiceName>$Target/Property[Type="AuthorMPs.Tutorials.UsingAbstractCl
asses.ApplicationA"]/ServiceName$</ServiceName>
    </Configuration>
</UnitMonitor>
</Monitors>

```

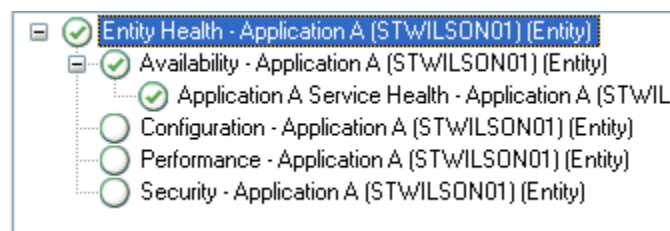
The display strings are:

```

<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA.ServiceState
">
    <Name>Application A Service Health</Name>
</DisplayString>
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA.ServiceState
" SubElementID="Running">
    <Name>Service Running</Name>
</DisplayString>
<DisplayString
ElementID="AuthorMPs.Tutorials.UsingAbstractClasses.ApplicationA.ServiceState
" SubElementID="NotRunning">
    <Name>Service Not Running</Name>
</DisplayString>

```

Once the MP is imported and has been deployed, go to the **All Applications** state view and look at the health of your applications. You should see that Application A objects have state, whereas the Application B objects do not. This is because I only added a monitor to the Application A class. If you open the health explorer you will see the following for an Application A object:



If you open the health explorer for an Application B object you will see:

