Lab LM07

Building and extending MOM using MCF and the SDK

Objectives

In this lab we'll explore the experience around using the MOM connector framework (MCF) web service to build a "product connector" that integrates with MOM. There are basically 2 types of connectors:

- Product connectors that submit Alerts to MOM from another *system* and keep them in sync bi-directionally after with this *system*
- Product connectors that subscribe for certain alerts within MOM and then create them as records in another *system* and then keep them in sync with that other *system*.

**System* – could be any other product including trouble ticketing tools, other management products or home grown tools.

We will also explore using the new MOM 2005 Management Server class library by writing managed code to do things like:

- Obtain an alert and change the resolution state
- Put a machine into maintenance
- Apply an override

NOTE: All code required to be written is available in the corresponding Exercise Directory at c:\HandsonLabExercises

Prerequisite

Before working on this lab, you should be familiar with the basic concepts of MOM 2000/2005 and have working development experience around web services and C#.

Lab Setup

To complete this lab, you will be using one computer running Microsoft Operations Manager 2005 beta 3. It has the MOM Connector framework installed (part of Microsoft Operations Manager 2005 Server installation) and ASP.NET installed and enabled. The MOM 2005 computer is named **MOMSERVER.**

Estimated time to complete this lab: 75 minutes for all exercises (they do not have to be done in order)

Before You Begin

Before you begin the lab, you should make yourself familiar with the lab computers and setup. If you have not used Microsoft® Virtual PC 2004 images before quickly review the Virtual PC 2004 Overview below.

Microsoft® Virtual PC 2004 Overview

This lab makes use of Virtual PC 2004, which is an application that allows you to run multiple virtual computers on the same physical hardware. During the lab you will be running one or more Virtual PC images on a lab host computer.

During this lab the there may be multiple Virtual PC images running. Some may be running in full screen mode while others maybe run in windowed mode. Here are some tips in navigating from the host desktop to each of your Virtual PC images:

Input focus for your mouse and keyboard passes to the Virtual PC when you click inside a Virtual PC window.

To move the input focus from a windowed Virtual PC to the desktop or another window move the mouse pointer outside the Virtual PC's window and click on the relevant window or desktop location.

To toggle a Virtual PC image from full screen to windowed or vice versa use the **ALT key to the right of the SPACEBAR + Enter**. The ALT key to the right of the SPACEBAR is also known as the **"Host Key"** so the full screen/windowed mode switching command is **"Host Key+Enter"**

To logon (**<CTRL-ALT-DEL>)** to a Virtual PC image use the **ALT key to the right of the SPACEBAR + Delete**. The ALT key to the right of the SPACEBAR is also known as the "Host Key" so the logon (**<CTRL-ALT-DEL>)** command is **"Host Key+Del".**

Note: Pressing <CTRL-ALT-DEL> while keyboard focus is on a Virtual PC will cause the Virtual PCs' host computer to capture the event rather than the Virtual PC.

You can find more information about Virtual PC 2004 by accessing the Virtual PC 2004 help from the Virtual PC Console.

Lab Setup

To complete this lab you will be using one Microsoft Virtual PC images. These images are configured as follows:

Name: MOMSERVER

Domain: Workgroup.

Administrator password: SMX#2001.

Software Installed:

- Windows Server 2003
- .NET Framework 1.1
- SQL Server 2000 SP3
- SQL Reporting Services
- Visual Studio .NET 2003

To Start the Lab

▶ **Logon to the Host computer**

If you are not already logged onto the Virtual PC Host computer, do so using the credentials supplied by the Instructor.

▶ **Start the Virtual PCs required for the lab**

▶ **Manually starting up the Virtual PC images (can be used in lieu of lab desktop icon):**

1. If Virtual PC Console is not running, click Start, point to All Programs or Programs, and then click **Microsoft Virtual PC**.
2. Check to make sure that the one Virtual PCs is in a “Running” state. You can see the state of each PC by looking within the Virtual PC console. Virtual PCs that are successfully running will be listed as “Running” within the console and you will have corresponding windows open for each Virtual PC.
3. If the Virtual PC is not listed as “Running” you must start or resume it. In the list of Virtual PCs, double-click the Virtual PC that you want to start or resume. In the Virtual PC window, on the Action menu, click Start or Resume.

Note: If a virtual PC must be started it may take some time for the image to boot. Once the computer has finished booting up, you can log onto the Virtual PC using the credentials supplied by the Instructor.

Exercise 1
Create a MCF product connector that subscribes for alerts from MOM

Scenario

In this lab we'll explore the MOM connector framework (MCF) web service and walk through a sample that subscribes for alerts within MOM through the MCF web service. This scenario is used when MOM needs to be integrated into another management product or trouble ticketing tool.

Goal

To show how to call and use the web service in the above scenario.

Tasks	Detailed Steps
1. Get familiar with the MCF	**a.** Open Internet Explorer and navigate to http://momserver:1271/connectorservice.asmx. Check out the different methods that are exposed and the WSDL types exposed. This is the MOM SP1 Connector feature pack version (released Fall 03) **b.** Now navigate to http://momserver:1271/connectorserviceV2.asmx. This is the MOM 2005 version.
2. Check out the MOM SDK docs that include MCF details (available for download here: http://www.microsoft.com/mom/downloads/sdk/default.mspx)	**a.** Launch "Shortcut to Exercises" on the desktop. Open MOMSDK.chm. Navigate to: MOM -> About MOM -> MOM Connectors MOM -> Using MOM -> Using the MOM Connector Framework MOM -> MOM Reference -> MOM .NET Framework Class libraries -> MOM connector framework Class Library Both the Connector and Connector.V2 docs are available here.
3. Open the MOM Administration Console and view Product Connectors	**a.** Launch Start -> All Programs -> Microsoft Operations Manager 2005 -> MOM Administration Console **b.** Expand **Administration** and select **Product Connectors** **c.** Note that there is no product connector called "Subscribe PC" – we'll create one through this exercise.
4. Launch Visual Studio 2003 and create a product connector **NOTE: All code written here is also available to copy and paste from c:\HandsonLabExercises\Exercise1\exercise1.txt**	**a.** Start -> All Programs -> Microsoft Visual Studio .NET 2003 -> **Microsoft Visual Studio .NET 2003** **b.** Create a new project. File -> New -> Project **c.** Create a Visual C# Console Application. Name the project **– MCFSubscribe** and save it in the **c:\HandsonLabExercises\Exercise1** directory. **d.** Right click "**References"** under MCFSubscribe in the Solution Explorer and select **Add Web Reference.** **e.** Type in **http://momserver:1271/connectorserviceV2.asmx** as the URL. If it prompts for credentials use: administrator with password SMX#2001. (if this doesn't work try localhost as the server name) **f.** Rename the **Web reference name** to **MCF** and click **Add Reference.** Now we have a web reference to the MCF web service running on MOMSERVER. **g.** In Class1.cs type the following under "using System;".

```
using MCFSubscribe.MCF;
using System.Net;
```

Now we can start using the MCF web service.

h. In the Main method type the following lines.

```
MCF.ConnectorServiceV2 connector = new ConnectorServiceV2();
connector.Credentials = CredentialCache.DefaultCredentials;
ForwardeeInfo info = new ForwardeeInfo();
info.ForwardeeName = "Subscribe PC";
Byte resolutionstate = Byte.Parse("151");
Guid guid = connector.Setup(info, resolutionstate);
Console.WriteLine(guid.ToString());
```

This defines a **product connector** called “Subscribe PC” for the resolution state “151” and a Registration id is returned – this is a GUID we will use later.

i. Build the MCFSubscribe project. (From Visual Studio .NET 2003 – Build -> Build Solution.)

j. Run MCFSubscribe.exe from a command line window by changing directory to be in c:\HandsonLabExercises\Exercise1\MCFSubscribe\bin\debug. Then run MCFSubscribe.exe.

k. Copy the GUID that is returned – we will need it later.

l. Switch back to the MOM Administrator Console. Expand **Administration** and select **Product Connectors.** The product connector that was created above now appears here.

m. Switch back to Visual Studio .NET 2003 and comment the following lines out:

```
//Guid guid = connector.Setup(info, resolutionstate);
//Console.WriteLine(guid.ToString());
```

Since the product connector is already setup we will not have to do that again.

n. Add the following code at the end within the Main method.

```
Guid registrationId = new System.Guid("{<paste-registrationid-returned-from-setup>}");
DataChanges chflag = DataChanges.NewAlerts | DataChanges.UpdatedAlerts;
connector.Initialize(registrationId, chflag);
```

This will Initialize the product connector – replace the registration id above with the initial one returned in the step above.

o. Build and run the MCFSubscribe.exe program again. Note in the MOM Administration console that the “Subscribe PC” is now initialized (you may need to refresh the console – F5)

p. Switch back to the MOM Administrator Console. Expand **Administration** and select **Product Connectors.** The product connector that was created above now is **Initialized**.

q. Switch back to Visual Studio .NET 2003 and comment the following:

<table>
<tr><td></td><td>

```
//connector.Initialize(registrationId, chflag);
```

This will stop Initializing the product connector.

r. Add the following code at the end within the Main method.

```
Data getdataAlerts = connector.GetData(registrationId,
chflag, 100);

Console.WriteLine(getdataAlerts.NewAlerts.Length.ToString(
));
```

This now calls GetData against MCF and returns both new and updated alerts. The MCFSubscribe program also outputs the total number of NewAlerts in the result set.

s. Build and run the MCFSubscribe.exe program again. Note that there are **0** New Alerts in the result set.

t. Open the MOM Operator Console. Start -> All Programs -> Microsoft Operations Manager 2005 -> **MOM Operator Console**

u. Select any open alerts from the **Alerts –All** view, right click on an alert and select “Set Alert Resolution State” -> **Subscribe PC**

v. Now run the MCFSubscribe.exe program again. 1 alert will be available in the NewAlert result set. If this were a real product connector it would send the alert to the other system and if was able to do this successfully it would Acknowledge this back to MCF
</td></tr>
<tr><td>5. Working with NewAlerts and UpdatedAlerts as a product connector</td><td>

a. In the MCFSubscribe to project add the following code:

```
Alert[] newAlerts = getdataAlerts.NewAlerts;

Console.WriteLine("New Alerts");

foreach (Alert newAlert in newAlerts)

{

Console.WriteLine("Name: " + newAlert.Name);

}

Alert[] updatedAlerts = getdataAlerts.UpdatedAlerts;

Console.WriteLine("Updated Alerts");

foreach (Alert updatedAlert in updatedAlerts)

{

Console.WriteLine("Name: " + updatedAlert.Name);

}
```

This will show the **NAME** property on both a New and UpdatedAlert.

b. Notice that the New Alert and Updated Alert are the same alert. This is because the first update for every alert is how the Alert History is obtained even though none of the Alert properties have changed.

c. Switch back to the MOM Administrator Console. Expand **Administration** and select **Product Connectors.** The product connector that was created above now has one NewAlert and UpdatedAlert waiting for it.
</td></tr>
<tr><td>6. Acknowledging New Alerts and Updated</td><td>

a. In the MCFSubscribe to project add the following code:

```
foreach (Alert newAlert in newAlerts)
```
</td></tr>
</table>

<table>
<tr><td>Alerts</td><td>

```
{
      Guid[] ack = new Guid[1];
      ack[0]= newAlert.AckId;
      connector.AckData(registrationId, ack);
}
foreach (Alert updatedAlert in updatedAlerts)
{
      Guid[] ack = new Guid[1];
      ack[0]= updatedAlert.AckId;
      connector.AckData(registrationId, ack);
}
```

This will call AckData against all the new alerts and updated alerts.

b. Switch back to the MOM Administrator Console. Expand **Administration** and select **Product Connectors.** The product connector that was created above now should have 0 updates waiting for it.</td></tr>
<tr><td>7. Cleaning up</td><td>**a.** When a product connector is done it should call the following methods. In the MCFSubscribe to project add the following code:

```
connector.Uninitialize(registrationId, chflag);
connector.Cleanup(registrationId);
```

The above will cleanup all resources in the MOM DB for a particular product connector. A product connector should call **Uninitialize** when it stops temporarily and should call Cleanup when uninstalling.

b. Switch back to the MOM Administrator Console. Expand **Administration** and select **Product Connectors.** The product connector that was created above should now not show up there anymore.</td></tr>
</table>

Summary

In this exercise a sample MOM product connector was created. It didn't connect to another system but it did call a number of the major methods on the MCF web service.

Exercise 2
Create a MCF product connector that inserts alerts into MOM

Scenario

In this lab we'll explore inserting alerts into the MOM connector framework (MCF) web service. This scenario is used when another management product needs to integrate its alerts or trouble tickets into MOM.

Goal

To show how to call and use the web service in the above scenario.

<table>
<tr><th>Tasks</th><th>Detailed Steps</th></tr>
<tr><td>1. Launch the "MCF-InsertAlerts" Visual Studio .NET 2003 solution</td><td>
a. Launch "Shortcut to Exercises" on the desktop. In the Exersice2\MCFInsert directory launch MCFInsert.sln
b. Walk through the sample code and read through the comments in the hidden regions.
c. Build MCFInsert Program and then run MCFInsert.exe from the command line. Copy the registration GUID that is returned so we can use it in the next step.
d. Comment out the following 2 lines:
<code>//Guid guid = connector.Setup(info, resolutionstate);</code>
<code>//Console.WriteLine(guid.ToString());</code>
e. Uncomment the Initialize region and the Exercise2-Insert region.
f. Build and run the MCFInsert program again.
g. Switch to the Operator Console. Note the alerts in the "Alerts – All" view. One of the alerts has a ruleId specified while the other doesn't.
h. Switch back to the MCFInsert program and uncomment:
<code>//connector.Initialize(registrationId, chflag);</code>
This will make sure the product connector isn't initialized again.
i. Build and run the MCFInsert program again.
j. Switch to the Operator Console. Note the alerts in the "Alerts – All" view. There should only be one new Security Breach Alert and the other should have a repeat count of 1 (which means it happened once and repeated once – ie. A total of 2 times). This is because the RuleID specified on one alert has a policy to suppress duplicates when certain fields are the same.
k. Change one of the alerts in MOM that was inserted. Add an owner to an alert that was inserted or change a custom field.
l. Now in the MCFInsert program uncomment the section in Exercise2-GetData and comment out everything in Exercise2-Insert.
m. Build and run the MCFInsert program again. Notice the outstanding Alert updates. All alerts have at least one Update waiting for them since the alert history is added by MOM.
n. Just like in exercise 1 a real product connector would need to process these Alert Updates and then Acknowledge that they were successfully updated in the other system. If it isn't
</td></tr>
</table>

	Acknowledged then MCF will continue to return the update to the product connector.

Summary

This is the main flow of data for this type of product connector: New alerts are inserted when they need to be and GetData is regularly called to obtain the changes to these inserted alerts so they can be updated in the system they came from.

Exercise 3

Walk through a working sample product connector that synchronizes MOM alerts to another custom system

Scenario

In this lab we'll explore a working sample product connector that uses the MOM Connector framework to synchronize MOM Alerts into another system. In this case the other system is a simple XML file.

Goal

To show how to call and use the web service in the above scenario.

Tasks	Detailed Steps
1. Understand and walk through the MOM sample or generic product connector	**a.** Launch "Shortcut to Exercises" on the desktop. In the Excercise3 directory open the "Shortcut to Connector" and then open MomSdkConnector.sln **b.** Check out the Connector.cs , *Forwarder.cs files **c.** The sample includes a small tool to view the XML file that the product connector creates. It also includes a setup program to install the product connector. **d.** This product connector is already installed and configured on this momserver computer.
2. Demo the sample product connector	**a.** Make sure the **MomSdkConnector** service from the Services snap-in is started. **b.** In the MOM Operator console change any alerts resolution state to "MomSdkConnector" **c.** Refresh the console until the resolution state switches back to New **d.** In the Excercise3 directory launch "**Shortcut to TroubleTicket.exe"** **e.** Click **Load.** The Alert that was just marked for forwarding should appear here. **f.** Change the alert (for example change the AssignedTo value from blank to "MOMAdmin") and select **Save**. **g.** Switch back to the Operator Console and refresh until you see the update is made. **h.** Mark some more alerts to forward and make some changes in either location. Observe how they keep in sync. Check the MOM Alert history for a complete view of what is happening with each alert.

Summary

In this exercise we drilled into how the MOM to generic sample product connector works and checked out some underlying code.

Exercise 4

Access MOM Operational Data programmatically using the new MOM Management Server Class Library

Scenario

In this lab we'll explore how to use the MOM Management Server class library to access data including alerts, events, performance data and attributes using Visual Studio .NET 2003.

Goal

To show how easy it is to build software that accesses MOM data collected through MOM management packs programmatically.

Tasks	Detailed Steps
1. Start programming against the MOM Management Server Class Library	**a.** Launch "Shortcut to Exercises" on the desktop. Open the MOMSDK.chm. Navigate to: MOM -> -> MOM Reference -> MOM .NET Framework Class libraries -> Microsoft.Enterprise.Management.Mom Namespace **b.** Expand Classes and look at the methods on the Computer class **c.** Under Classes expand the Alert class and view the properties
2. Add a reference to the MOM Management Server API **NOTE: All code written here is also available to copy and paste from c:\HandsonLabExercises\Exercise4\exercise4.txt**	**a.** Launch **Visual Studio 2003 .NET** **b.** Create a new C# console application. Create it in the Exercise4 directory under c:\HandsonLabExercises and name it **MOMOpData** **c.** Add a reference to the MOM Management Server API by right clicking on **References** and selecting **Add Reference**. Click **Browse** and navigate to "c:\program files\Microsoft Operations Manager 2005\SDK Bin" and select **Microsoft.Mom.Sdk.dll**. Then click Open and Ok. **d.** Follow the same steps for **MOM.Context.dll** as above. **e.** Add the following code to the project at the top: `using Microsoft.EnterpriseManagement.Mom;` Now we can start using the MOM Management Server Class library. **f.** Add the following code in the main method: `Administration momadmin = Administration.GetAdministrationObject();` This returns a MOM Management Server Administration Object and is the main object that can be used to launch in to all other objects and methods.
3. Access Alerts associated to a computer	**a.** Add the following code: `AlertQuery alertQuery = momadmin.GetAlertQueryObject();` `AlertsCollection alerts = alertQuery.GetAlertsByResolutionStateIdAndComputer(0, "MOMSERVER", "WORKGROUP");`

<table>
<tr><td></td><td>

```
foreach (Alert alert in alerts)
{
      alert.OwnerName = "Admin";
      alert.ResolutionState = new
AlertResolutionState(85);
      alert.Update();
      Console.WriteLine(alert.Name);
};
```

This returns all alerts associated to the MOMSERVER machine and changes the resolution state to acknowledged and specifies an owner.</td></tr>
<tr><td>4. Access computer attributes collected</td><td>

a. Add the following code:

```
ComputersCollection computers =
momadmin.GetAgentManagedComputers();
Console.WriteLine(computers.TotalCount.ToStr
ing());
foreach(Computer c in computers)
{
      Console.WriteLine(c.Name);


      AttributesCollection attributes =
c.GetAssociatedAttributes();
      foreach(Microsoft.EnterpriseManagement
.Mom.Attribute a in attributes)
      {
            Console.WriteLine(a.Name + "=" +
a.Value.ToString());
      }
}
```

This returns all the computer attributes collected by MOM as specified by MOM management packs.</td></tr>
<tr><td>5. Show the Rule Groups associated to the MOM Management Server</td><td>

a. Add the following code:

```
ComputersCollection mgmtservers =
momadmin.GetManagementServers();
Console.WriteLine(mgmtservers.TotalCount.ToS
tring());


foreach(Computer c in mgmtservers)
{
```

</td></tr>
</table>

	`` RuleGroupsCollection rulegroups = c.GetAssociatedRuleGroups(); foreach(RuleGroup rg in rulegroups) { Console.WriteLine(rg.Name); } } ``` This returns a set of Rule Groups associated to the MOMSERVER machine. This will take some time to return.
6. Show the performance counters associated a machine MOM is managing	**a.** Add the following code: ``` foreach(Computer c in mgmtservers) { PerformanceDataCollection perfData = c.GetPerformanceData(); foreach(PerformanceData perf in perfData) { Console.WriteLine("Object Name: " + perf.Object + "Counter Name: " + perf.Counter + " Instance Name: " + perf.Instance + " Last Value:"); if (perf.GetLatestSample() != null) { Console.WriteLine((perf.GetLatestSample())); } } }; ``` This returns a set of performance counters and their latest values associated to the MOMSERVER machine.

Summary

In this exercise we started to uncover the different objects that expose MOM operational data programmatically. These can be leveraged in many scenarios like building a portal to MOM data, automating actions like resolving alerts or assigning alert ownership. These objects are over the data MOM stores in the Operational Database not reporting data warehouse.

Exercise 5

Access MOM configuration data programmatically using the new MOM Management Server Class Library

Scenario

In this lab we'll explore how to use the MOM Management Server class library to access data including computers, computer groups, overrides and many related actions using Visual Studio .NET 2003.

Goal

To show how easy it is to build software to control MOM itself programmatically.

Tasks	Detailed Steps
1. Start programming against the MOM Management Server Class Library	**a.** Launch "Shortcut to Exercises" on the desktop. In the Exersice5\MOMConfig directory launch MOMConfig.sln **b.** Walk through the sample code and read through the comments in the different regions.
2. Show Computer groups region	**a.** Expand the ComputerGroup region and note the code required to enumerate all the computer groups within this current management group. **b.** Build and run MOMConfig from a cmd window. **c.** Note the output – these are all the top level computer groups. Open the MOM Administration Console and expand **Management Packs** and **Computer Groups.** The top level set of computer groups should match. **d.** Switch back to MOMConfig in Visual Studio 2003 and uncomment the section in the ComputerGroup region. Build and run MOMConfig again. **e.** Note the output now contains the first level of subgroups under each of the Top level computer groups.
3. Show Agent managed computers region	**a.** Expand and uncomment the AgentManaged region. **b.** Build and run MOMConfig from a cmd window. **c.** Note the output. This shows the entire list of computers that are being monitoring with an agent and checks to see if a computer is in maintenance mode (in this demo only the MOMSERVER itself is being managed)
4. Set a machine in maintenance mode region	**a.** Expand and uncomment the MaintenaceMode region. **b.** Build and run MOMConfig from a cmd window. **c.** Note the output. This has now put the MOMSERVER machine into maintenance. Open the MOM Operator Console and click the Computer view and see if the computer is in Maintenance Mode. While in this mode alerts will be resolved (they are still collected along with performance data and events but no MOM Management server side responses are run including email responses). The computer will automatically come out of maintenance mode in 2 hours. It can be taken out of maintenance programmatically or by the user. **d.** Switch back to the MOMConfig code and comment out the Maintenance mode region and then Build and run the program again. Does the

	MOMSERVER computer appear in maintenance? What is the user name that put the machine into maintenance?
5. Set an override on a computer region	**a.** Expand and uncomment the Override region **b.** Build and run MOMConfig from a cmd window. **c.** Note the output. This sample writes out all the current overrides associated to all the computers MOM is managing (there are none by default on this demo). It then applies an override to a computer called MOMSERVER. **d.** View the Override from the MOM Administration Console. Expand Management Packs -> **Override Criteria**. The override that was just created should appear here. **e.** It is also possible to see the same override on the actual rule that uses it (multiple rules can use the same override). To see this go to Management Packs -> Test Rule Group -> Performance Processing Rules -> **Sample CPU Threshold.** Select the Threshold tab and click **Set Criteria**. The Override Name is visible here.

Summary

In this exercise we started to uncover the different objects that expose MOM configuration programmatically. These can be leveraged in many scenarios like tuning management packs and overrides for the different types of computers or scheduling maintenance for known downtime.

Completing the Lab

When you are done with the exercises, please shut down the Virtual PC image by following these instructions:

1. Close the Virtual PC window by clicking the red-X icon in the upper right of the window. If the Virtual PC image is in full screen mode, you can get back to a regular window by typing **Right-Alt + Enter**.
2. When prompted select ‘**Turn off and delete changes**’ and then click **OK.**