Developing and Deploying vSphere Solutions, vServices, and ESX Agents

vmware®

You can find the most up-to-date technical documentation on the VMware website at:

https://docs.vmware.com/

If you have comments about this documentation, submit your feedback to

docfeedback@vmware.com

Contents

Developing and Deploying vSphere Solutions, vServices, and ESXi Agents

Developing and Deploying vSphere Solutions, vServices, and ESX Agents provides information about how to extend the function of vSphere by integrating solutions with ExtensionManager, vCenter Server Extensions, vServices, and vSphere ESX Agent Manager.

Intended Audience

This information is intended for anyone who wants to deploy vSphere extensions as vSphere solutions by integrating with ExtensionManager, vCenter Server Extensions, vServices, and ESX Agent Manager features. This publication does not describe how to develop vSphere extensions in detail.

Related Documentation

For information about how to develop vSphere Web Client extensions, see the *VMware vSphere Web Client Extensions Programming Guide*.

For information about the data objects that the vSphere ESX Agent Manager provides, see the *vSphere ESX Agent Manager API Reference* documentation.

For information about developing vSphere applications, see the *vSphere Web Services SDK Programming Guide* and the *VMware vSphere API Reference* documentation.

Introduction to vSphere Solution Development

1

You can add functions to vSphere by developing software applications that you register as vCenter Server extensions. A vSphere Solution is an extension that registers with vCenter Server and implements some or all of the extension features of the vSphere API.

Developing and Deploying vSphere Solutions, vServices, and ESX Agents uses the terms solution and extension interchangeably.

A vSphere Solution is an object or program that you create by extending specific classes in the vSphere Web Services API. After you register your solution with the instance of the `ExtensionManager` managed object associated with your vCenter Server, you see your solution under **Home > Administration > Solutions > vCenter Server Extensions** of your vSphere Web Client.

You can create a vService solution to provide access for a specific application to connect to a service across the network.

You can create an ESX Agent to extend the functions of an ESXi host and provide additional services that a vSphere solution requires.

The vService Manager and ESX Agent Manager are pre-built solutions that are part of the vCenter Server Extensions functionality within the vCenter Server.

The vSphere ESX Agent Manager includes an API library that allows you to adapt and modify the ESX Agent Manager (EAM) Sample Solution and view it using the vSphere Web Client.

The extension features in the vSphere Web Services API include functions so that you can perform the following tasks:

- Register permissions, faults, and events for an extension.

- Identify and prevent manual operations on virtual machines and vApps that extensions deploy.

- Integrate extensions with ExtensionManager.

- Store data about extensions in the vSphere database.

- Provide user interface plug-ins that extend the vSphere Web Client.

To use the extension functions of the vSphere API, a solution must register itself with the `ExtensionManager` that runs in a vCenter Server instance. By registering with `ExtensionManager`, a solution can access the extension features of the vSphere extension API.

You can develop solutions that add functions to the standard functions of vCenter Server. You can deploy a solution as an Open Virtualization Format (OVF) package, with optional VMware vSphere Installation Bundles (VIB). You can also install solutions by using an installer, such as Windows Installer (MSI) or RPM Package Manager. Most of the extension functions in the vSphere API are independent of the technology that you use to deploy a solution. If you deploy a solution by using OVF, you can use the vCenter Extension vService to simplify the registration of the solution with vCenter Server.

vCenter Server 6.7 provides built-in solutions.

- vSphere ESX Agent Manager

- vService Manager

vSphere ESX Agent Manager and vService Manager are part of a standard vCenter Server installation. These solutions appear in vCenter Server Extensions and ExtensionManager with any other solutions that register with ExtensionManager.

- Overview of the vSphere Extension APIs

  The vSphere SDK provides a set of APIs that you can use to register extensions with vCenter Server. To develop vSphere extensions, the most important object in the vSphere Extension API is the `ExtensionManager` managed object.

- Introducing the vCenter Server Extensions

  The vCenter Server Extensions allows you to monitor and interact with solutions that are registered with a vCenter Server instance.

- Introducing vSphere ESX Agents and Agencies

  A vSphere ESX agent is a virtual machine and an optional vSphere Installation Bundle (VIB) that extends the functions of an ESXi host to provide additional services that a vSphere solution requires.

- Introducing vSphere ESX Agent Manager

  vSphere ESX Agent Manager automates the process of deploying and managing vSphere ESX agents.

- Introducing vServices

  A vService is a service that a solution provides to specific applications that run inside virtual machines and vApps. A solution can provide several types of vServices. Virtual machines or vApps can have dependencies on several types of vServices.

- Introducing vService Manager

  vService Manager allows you to manage the set of vServices that extensions provide and to configure virtual machines and vApps to be dependent on these vServices.

- Introducing the vCenter Extension vService

  The vCenter Extension vService is a standard vService that vSphere 6.7 provides. With the vCenter Extension vService you can register virtual machines as vCenter Server extensions with minimal user interaction.

Overview of the vSphere Extension APIs

The vSphere SDK provides a set of APIs that you can use to register extensions with vCenter Server. To develop vSphere extensions, the most important object in the vSphere Extension API is the `ExtensionManager` managed object.

Use the `ExtensionManager` managed object in your vCenter Server instance to register a new extension. Extensions can add new objects to the vCenter inventory. Extensions define tasks, events, and faults that relate to actions that the solution performs on the objects, the events that occur in the extension, and the problems that the objects encounter. You can also use `ExtensionManager` to add user interface elements to the vSphere Web Client to allow users to interact with the objects that your extension provides to vCenter Server.

vCenter Server extensions can have both a client and a server component, or they can consist of just a client component or just a server component. Each instance of an extension is represented by an `Extension` data object.

You can indicate that an extension manages the virtual machines or vApps that it creates by setting the `managedBy` property in the `ManagedEntityInfo` data object for that virtual machine or vApp. When you register an extension with vCenter Server, you can define an icon that represents the virtual machines that the extension manages. Virtual machines that an extension manages display the icon that you define in the inventory of virtual machines in the vSphere Web Client. The vCenter Server shows a warning if users try to perform manual operations on a virtual machine or vApp that the extension manages.

ExtensionManager adds data objects to the vSphere Extension APIs that integrate extensions as vCenter solutions that you can manage.

For information about how to develop vSphere Web Client extensions, see the *VMware vSphere Web Client Extensions Programming Guide*.

For information about developing vSphere applications, see the *vSphere Web Services SDK Programming Guide* and the *VMware vSphere API Reference* documentation.

Introducing the vCenter Server Extensions

The vCenter Server Extensions allows you to monitor and interact with solutions that are registered with a vCenter Server instance.

The vCenter Server Extensions shows four standard tabs for each running solution. The tabs list the virtual machines that a solution deploys and manages, show the status, name, company URL, and version of the solution.

A solution can also use the **Solutions** portlet on the **Summary** tab to add links to Web pages that provide some functionality specific to the solution. For example, the pages can be used to allow users to configure the solution, or to provide access to the functions of the solution.

Standard Tabs in the vCenter Server Extensions

vCenter Server Extensions displays standard tabs for each solution that is running on a vCenter Server instance.

Table 1-1. Standard Tabs in the vCenter Server Extensions

Tab	Description
Summary	General information about the solution, including its name, vendor, and version.
Monitor	Filters all vSphere events and lists only the system activities that are related to the specific solution.
Configure	Displays one or more ESX Agents, the services they contain, and actions you can perform on them.
VMs	Information about the virtual machines and vApps that the solution deploys. You can perform all operations applicable for a VM and vApp.

Introducing vSphere ESX Agents and Agencies

A vSphere ESX agent is a virtual machine and an optional vSphere Installation Bundle (VIB) that extends the functions of an ESXi host to provide additional services that a vSphere solution requires.

For example, a solution might require a particular network filter or firewall configuration to function. A solution can use an ESX agent to connect to the vSphere Hypervisor and extend the host with functions specific to that solution. For example, the ESX agent can filter network traffic, act as a firewall, or gather other information about the virtual machines on the host.

ESX agent virtual machines are similar to services in Windows or Linux. They start when the operating system starts and they stop when it shuts down. The behavior of ESX agent virtual machines is transparent to the user. A vSphere host reaches the ready state when the ESXi operating system has started and all ESX agent virtual machines have been provisioned and powered on.

To integrate an agent with vSphere ESX Agent Manager and extend the capabilities of an ESXi server, an ESX agent must be packaged as an OVF or a VIB module.

ESX agencies act as containers for ESX agents. ESX agencies aggregate information about the agents that they manage. For example, ESX agencies provide an overview of the ESX agents that they contain by aggregating all the issues that relate to the ESX agents.

Introducing vSphere ESX Agent Manager

vSphere ESX Agent Manager automates the process of deploying and managing vSphere ESX agents.

The services that ESX Agent Manager provides include out-of-the-box integration of agents with vSphere features such as DRS, AddHost, High Availability, DRM, and maintenance mode. Each of these features can be difficult to manually integrate. ESX Agent Manager also allows you to monitor the health of ESX agents, and blocks users from performing certain operations on ESX agents that might affect the virtual machines that use them. For example, ESX Agent Manager can prevent an ESX agent virtual machine from being powered off or moved from an ESXi host that contains other virtual machines that use that agent.

ESX Agent Manager adds an **ESX Agencies** entry to the table of contents under the **Configure** tab of vSphere ESX Agent Manager. The **ESX Agencies** tab shows information about running agencies, lists any orphaned ESX agents, and logs information about the ESX agents that ESX Agent Manager manages.

Introducing vServices

A vService is a service that a solution provides to specific applications that run inside virtual machines and vApps. A solution can provide several types of vServices. Virtual machines or vApps can have dependencies on several types of vServices.

A vService is similar to a virtual hardware device upon which virtual machines and vApps can depend. Instead of providing a piece of virtual hardware, vServices typically provide access to a service across a network. By providing a vService, a solution can expose application-aware services to virtual machines and vApps. For example, a vService can provide a backup service or a logging service to virtual machines and vApps.

Virtual machines that use a vService have a vService dependency. If you mark a virtual machine as having a vService dependency, the virtual machine cannot power on unless you bind it to a provider of that vService.

If you deploy virtual machines by using Open Virtualization Format (OVF), you can specify in the OVF descriptor that the virtual machine depends on a vService. The OVF deployment wizard includes a **Service Binding** step that presents the user with the vService to which to bind the virtual machine. If you configure a virtual machine to have dependency on a vService, the solution that provides the vService receives notifications when specific events occur on that virtual machine. The vService can modify the OVF environment and OVF descriptor of the virtual machines and vApps that depend on it. With vServices, you can encode application-specific information in the OVF package about how a virtual machine or vApp interacts with a specific solution. You can also add solution-specific information to code running inside the guest operating system.

Introducing vService Manager

vService Manager allows you to manage the set of vServices that extensions provide and to configure virtual machines and vApps to be dependent on these vServices.

You access vService Manager from vCenter Server Extensions. You can add dependencies on vServices to virtual machines and vApps directly in the vSphere Web Client.

Introducing the vCenter Extension vService

The vCenter Extension vService is a standard vService that vSphere 6.7 provides. With the vCenter Extension vService you can register virtual machines as vCenter Server extensions with minimal user interaction.

Typically, you register an extension with vCenter Server by running an MSI installer or an installation script in which you enter the credentials of a vCenter Server administrator and the connection parameters of the vCenter Server system. You can avoid these manual steps by using the vCenter Extension vService to register extensions in vCenter Server. vCenter Extension vService performs the following functions.

- Provides the virtual machine with the connection parameters and a login token of the vCenter Server system on which you install the extension.

- Registers the vCenter extension certificate of the solution with vCenter Server.

To deploy an extension using the vCenter Extension vService, you must deliver the vCenter extension in a virtual appliance that you deploy using OVF.

Demonstrating vCenter Server Extensions and vSphere ESX Agent Manager

2

The vSphere ESX Agent Manager (EAM) SDK contains a reference solution (EAM Sample Solution), that demonstrates many features of ExtensionManager and ESX Agent Manager.

- Functions of the EAM Sample Solution

    The EAM Sample Solution is a vCenter Server solution that runs on a Web application server, such as Apache Tomcat.

- Run the EAM Sample Solution

    You can run the EAM Sample Solution to demonstrate how a solution integrates vCenter Server Extensions and ESX Agent Manager.

- Modifying the EAM Sample Solution

    You can modify the EAM Sample Solution to experiment with solution development.

Functions of the EAM Sample Solution

The EAM Sample Solution is a vCenter Server solution that runs on a Web application server, such as Apache Tomcat.

By default, the EAM Sample Solution deploys ESX agent virtual machines to demonstrate how to deploy an ESX agent. You can configure the EAM Sample Solution to install a vSphere Installation Bundle (VIB).

The EAM Sample Solution performs the following actions:

- Registers itself as a vCenter Server extension.

- Appears in the vCenter Server Extensions.

- Adds a **Solutions** portlet to the Summary page of the EAM Sample Solution that you can use to deploy ESX agents on the hosts that the vCenter Server instance manages.

- Uses OVF to deploy ESX agent virtual machines on the ESXi hosts that the vCenter Server instance manages.

- Creates an ESX Agents resource pool and adds the ESX agent virtual machines to the resource pool.

- Optionally, installs a vSphere Installation Bundle (VIB).

- ESX Agent Manager then performs the following tasks:

    - Registers the ESX agent virtual machines with the ESX Agent Manager.

- Prevents users who do not have the **EAM.View** or **EAM.Modify** permissions from performing operations on the running ESX agents that the ESX Agent Manager manages.

- Monitors ESX agent virtual machines, for example for changes in the power state, and reports issues that users can see and resolve.

You can modify the EAM Sample Solution to experiment with solution development. The EAM Sample Solution serves as the basis for other sections in this publication.

Run the EAM Sample Solution

You can run the EAM Sample Solution to demonstrate how a solution integrates vCenter Server Extensions and ESX Agent Manager.

This information guides you through the process of setting up and running the EAM Sample Solution to provide a high-level introduction to the concepts and functions of vCenter Server Extensions and ESX Agent Manager. For information about how to implement and integrate a solution with vCenter Server Extensions and ESX Agent Manager, see Chapter 4 Integrating Solutions with vCenter Server Extensions and Chapter 5 Integrating Solutions with vSphere ESX Agent Manager.

Procedure

1 Download the vSphere ESX Agent Manager SDK

   The ESX Agent Manager SDK is part of the vSphere SDK. The ESX Agent Manager SDK provides a working example of a vSphere solution, the EAM Sample Solution.

2 Set up and Start the EAM Sample Solution

   The EAM Sample Solution is a Web-based application. The solution demonstrates how to use OVF and vSphere Installation Bundle (VIB) files to deploy ESX agent virtual machines to ESXi servers that a vCenter Server instance manages.

3 View the EAM Sample Solution in vCenter Server Extensions

   After you start the EAM Sample Solution in an application server, it appears in vCenter Server Extensions in the vSphere Web Client. You can monitor, manage, and configure the EAM Sample Solution and use it to deploy ESX agents.

4 Configure the Virtual Infrastructure for ESX Agent Deployment

   Before the EAM Sample Solution can deploy ESX agents on ESXi hosts, you must configure the virtual infrastructure so that vSphere can assign networks and datastores to the ESX agent virtual machines.

5 Deploy ESX Agents from OVF Using the EAM Sample Solution

   The EAM Sample Solution uses an OVF template to deploy ESX agent virtual machines from an application server onto ESXi hosts.

6 Monitor and Manage the Sample ESX Agency and Agent

   You can monitor the status of the sample ESX agent in ESX Agent Manager. The EAM Sample Solution provides basic management actions that you can perform in the EAM Sample Solution view.

7 Deploy ESX Agents and a VIB Using the EAM Sample Solution

The EAM Sample Solution optionally deploys an ESX agent virtual machine and a vSphere installation bundle (VIB).

Download the vSphere ESX Agent Manager SDK

The ESX Agent Manager SDK is part of the vSphere SDK. The ESX Agent Manager SDK provides a working example of a vSphere solution, the EAM Sample Solution.

The ESX Agent Manager SDK includes the ESX Agent Manager reference implementation, the EAM Sample Solution, ESX Agent Manager API documentation, Java source code for the sample, and the WSDL files that describe the sample solution.

Procedure

1 Download the vSphere Management SDK 6.7 bundle, `VMware-vSphere-SDK-6.7.x-build_number.zip`, from the VMware {code} site at https://code.vmware.com/home.

2 Unzip the vSphere SDK to a convenient location in your development environment.

3 Navigate to the ESX Agent Manager SDK at the following location in the vSphere SDK.

`VMware-vSphere-SDK-6.7.x-build_number\SDK\eam`

4 Copy the contents of `VMware-vSphere-SDK-6.7.x-build_number\SDK\eam` to a folder where you can modify the files.

For example, copy the files to `eam_work_folder`.

You downloaded and located the ESX Agent Manager SDK, and made a copy of its contents to work on.

What to do next

Set up the EAM Sample Solution.

Contents of the ESX Agent Manager SDK

The ESX Agent Manager SDK contains a Web archive that you can use to deploy the EAM Sample Solution and view it in the vSphere Web Client. The SDK also contains the source files of the solution that you can adapt to experiment with solution development.

Table 2‑1. Elements of the ESX Agent Manager SDK Bundle

Element	Description
`eam-sample.war`	Web archive that contains the ready-made version of the sample solution that you can deploy immediately.
`build.xml`	An Ant build file that you can use to rebuild the solution if you adapt it.
`\doc`	ESX Agent Manager API documentation.
`\src`	Java source files of the EAM Sample Solution.

Table 2‑1. Elements of the ESX Agent Manager SDK Bundle (Continued)

Element	Description
\war\vm	An OVF template and VMDK file from which the EAM Sample Solution deploys a simple ESX agent virtual machine.
\war\vib	VIB bundles for ESXi.
\war\WEB‑INF\	Source files of the Web application from which the EAM Sample Solution is deployed. You find ready-made JAX-WS stubs in \war\WEB‑INF\lib\eam+vim25‑wsdl.jar.
\wsdl	Web Service Definition Language (WSDL) files for ESX Agent Manager client stubs.

Set up and Start the EAM Sample Solution

The EAM Sample Solution is a Web-based application. The solution demonstrates how to use OVF and vSphere Installation Bundle (VIB) files to deploy ESX agent virtual machines to ESXi servers that a vCenter Server instance manages.

Prerequisites

- Download the vSphere ESX Agent Manager SDK.

- Verify that you have a running vCenter Server 6.7 instance.

- Verify that you have a running Web application server.

Procedure

1 Copy the eam‑sample.war file from the ESX Agent Manager SDK folder to a work directory.

2 Unzip the eam‑sample.war file.

3 Open the *<work_directory>*\eam‑sample\WEB‑INF\eamri.properties file in a text editor.

4 Provide the IP address of the machine where you run the solution, and the user name and password for your vCenter Server system in the eamri.properties file.

```
# IP of your machine.
self.ip=<solution_IP_address>
# URL prefix. If running on any other port than standard Tomcat port change 8080 to
# the used port.
self.url.prefix=http://*:<port_number>/eam-sample

# vCenter Server settings.
vc.host=<vCenter_Server_IP_address>
```

```
vc.http.proxy.port = 80
vc.https.proxy.port = 443
vc.user = <username>
vc.password = <password>
```

Note   If you run the Web application server on the same machine as vCenter Server, change the default port value of the `self.url.prefix` property from **8080** to **18080** to avoid a port clash between the vCenter Server instance and the Web application server.

5   Leave the `vib.deploy` property value set to **false**.

6   Save and close the file.

7   (Optional) If you change the default port in the `eamri.properties` file, you must replicate this change in the Web application server configuration file `catalina.properties`.

   `bio.http.port=<port_number>`

8   Copy the `eam-sample` directory from your work directory to the deployment location of your Web application server to deploy the EAM Sample solution.

9   In a Web browser, go to `http://<My_Web_App_Server_IP_address>:<port_number>/eam-sample/` to check that the EAM Sample Solution is running.

Note   Make sure that you use the correct port number in the URL in case you changed the default value.

You see the text `The EAM Sample Solution is up and running`.

You have set up and started the EAM Sample Solution.

What to do next

View the EAM Sample Solution in the vSphere Web Client and use it to deploy ESX agent virtual machines.

View the EAM Sample Solution in vCenter Server Extensions

After you start the EAM Sample Solution in an application server, it appears in vCenter Server Extensions in the vSphere Web Client. You can monitor, manage, and configure the EAM Sample Solution and use it to deploy ESX agents.

When you start the EAM Sample Solution, it registers itself as a vCenter Server extension with the vCenter Server instance that you configured in the `eamri.properties` file. The sample solution also registers itself as a vCenter Server solution with ExtensionManager.

Prerequisites

- Verify that you have a running vCenter Server 6.7 instance.

- Verify that you have set up and started the EAM Sample Solution in an application server.

Procedure

1 Open the vSphere Web Client and log in to the vCenter Server instance that you configured in the `eamri.properties` file.

2 Click **Home** in the navigation bar of the vSphere Web Client, and then click **Administration**.

3 Click **EAM Sample Solution** in the list of running vCenter Server solutions.

The general information and the health status of the EAM Sample Solution appear on the **Summary** tab.

4 Click each of the tabs in the EAM Sample Solution.

Option	Description
Summary	Displays an overview of the Solution that includes the associatedvCenter Server, the Solution's status, description, product page, and vendor page. The Solutions portlet in the **Summary** tab, holds a link to the **Configuration** window, that allows you to update a configuration, and disable or uninstall a solution.
Monitor	Displays Events and vServices that the solution provides. The EAM Sample Solution does not provide any vServices, so none are listed.
Configure	Displays one or more ESX Agents, the services they contain, and actions you can perform on them.
VMs	Displays one or more Virtual Machines or vApps that vCenter Server manages. You can deploy ESX agents on any of the listed objects.

The EAM Sample Solution is now available in vCenter Server Extensions.

What to do next

Before you can use the EAM Sample Solution to deploy ESX agents on ESXi hosts, you must configure the hosts and datacenters for ESX agent deployment.

Configure the Virtual Infrastructure for ESX Agent Deployment

Before the EAM Sample Solution can deploy ESX agents on ESXi hosts, you must configure the virtual infrastructure so that vSphere can assign networks and datastores to the ESX agent virtual machines.

You must set the ESX Agent Manager properties on a vCenter Server system to configure the network and datastore on which to deploy ESX agent virtual machines on this vCenter Server instance.

Procedure

1 From the vSphere Web Client Home page, click **Global Inventory Lists**.

2 Under **Resources**, click **Hosts**.

3 Select a host where you want to deploy ESX agent virtual machines.

4 Click the **Configure** tab to display configuration information for the host.

5 Select **Agent VM Settings** and click **Edit** in the top right corner.

6   From the **Datastore** drop-down menu, select a datastore in which to deploy the ESX agent virtual machines.

7   From the **Network** drop-down menu, select a network to connect the ESX agents.

8   Click **OK**.

You configured the network and datastore for the ESXi host where you want to deploy ESX agent virtual machines. You configured the IP address management so that vCenter Server can allocate IP addresses to ESX agent virtual machines.

Deploy ESX Agents from OVF Using the EAM Sample Solution

The EAM Sample Solution uses an OVF template to deploy ESX agent virtual machines from an application server onto ESXi hosts.

The EAM Sample Solution enables an ESX agent, creates a resource pool, creates a virtual machine folder, deploys an OVF template, and powers on the virtual machine.

The ESX agents that the EAM Sample Solution deploys from OVF are empty. The ESX agents do not add functions to the ESXi server on which you deploy them. The EAM Sample Solution demonstrates how a solution deploys ESX agents on compute resources. A solution deploys agents on a standalone host or on a cluster of hosts. If you deploy the EAM Sample Solution on a cluster of hosts, then it deploys ESX agents on all the hosts in the cluster.

The EAM Sample Solution displays information about the virtual machines that it creates, and how you can monitor and manage ESX agents in ESX Agent Manager. The EAM Sample Solution also demonstrates how ESX Agent Manager integrates with vSphere High Availability (HA), maintenance mode, Distributed Resource Scheduler (DRS), and Distributed Power Management (DPM).

Prerequisites

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have configured the virtual infrastructure on which to deploy ESX agents.

  Important   If you have not configured the ESX agent network and datastore settings on the host, ESX Agent Manager sets the status to red and ESX agents do not deploy. See Configure the Virtual Infrastructure for ESX Agent Deployment.

Procedure

1   Click **Home** in the navigation bar of the vSphere Web Client, and then click **Administration**.

2   Click **vCenter Server Extensions**.

3   Click **EAM Sample Solution** in the list of running vCenter Server solutions.

4   From the **Summary** tab, click the link from the Solutions portlet.

    The EAM Sample Solution Configuration page opens in a new window.

5   In the **Compute resources** panel, select the host name of the ESXi host or cluster on which to deploy ESX agents and click **Update Configuration**.

The **Recent Tasks** view at the bottom of the vSphere Web Client shows the progress of the EAM Sample Solution.

6   Click **Home** in the navigation bar of the vSphere Web Client and then click **Hosts and Clusters**.

The EAM Sample Solution created a resource pool named **ESX Agents** and an ESX agent virtual machine named **Sample Service (1)** on the host that you selected. The virtual machine has an icon in the inventory that denotes that it is an ESX agent.

7   Click the **Sample Service (1)** virtual machine icon, then click the **Summary** tab.

vCenter Server added a panel labeled **Managed by** to the **Summary** tab, which shows that ESX Agent Manager manages this ESX agent virtual machine.

8   (Optional) If you deployed the EAM Sample Solution on a cluster of hosts, put one of the hosts into maintenance mode.

ESX Agent Manager powers off the ESX agent that is running on that host.

9   (Optional) If you deployed the EAM Sample Solution on a cluster of hosts and you have configured DRS on the cluster, add a host to the cluster.

a   From **Agent VM Settings**, select a datastore and network on the new host.

ESX Agent Manager deploys an ESX agent on the new host and prevents any other virtual machines from migrating to the host until the ESX agent virtual machine is running.

You used the EAM Sample Solution to deploy an ESX agent virtual machine from an OVF template that the solution defines. The EAM Sample Solution registered that ESX agent with ESX Agent Manager.

What to do next

Use EAM Sample Solution and ESX Agent Manager to monitor and manage the running ESX agent virtual machine.

Monitor and Manage the Sample ESX Agency and Agent

You can monitor the status of the sample ESX agent in ESX Agent Manager. The EAM Sample Solution provides basic management actions that you can perform in the EAM Sample Solution view.

With the EAM Sample Solution, you can uninstall the solution, monitor status and logging information about the ESX agent and resolve issues in ESX Agent Manager.

Prerequisites

Verify that you used the EAM Sample Solution to deploy an ESX agent.

Procedure

1   Click **Home** in the navigation bar of the vSphere Web Client, and then click **Administration**.

2   Click **vCenter Server Extensions**.

3   Use the **vSphere ESX Agent Manager** and **EAM Sample Solution** to view and manage the ESX
     agents that you deployed, as described in the following table.

Option	Action
See events that ESX Agent Manager has logged in the vCenter Server database	Select **Administration > vCenter Server Extensions > vSphere ESX Agent Manager > Monitor**. The logged events appear in the **Tasks & Events** tab.
See the list of virtual machines that ESX Agent Manager manages	Select **Administration > vCenter Server Extensions > vSphere ESX Agent Manager > VMs** and click the **Virtual Machines** tab.
View information about the running ESX agencies	Select **Administration > vCenter Server Extensions > vSphere ESX Agent Manager > Configure** and click the **ESX Agencies** tab. The Sample Service appears in the list of running ESX agencies, with information that ESX Agent Manager has logged about the agency in the vCenter Server database.
View the Sample Service (1) ESX agent virtual machine that the EAM Sample Solution has deployed	Select **Administration > vCenter Server Extensions > EAM Sample Solution > VMs > Virtual Machines** .
View information about the agency that ESX Agent Manager has logged in the vCenter database.	Select **Administration > vCenter Server Extensions > EAM Sample Solution > Configure**. Click the **ESX Agencies** tab.

4   From the **vSphere ESX Agent Manager** window, choose **Virtual Machines** from the navigation
     menu. Power off an ESX agent virtual machine, and click **Yes** to ignore the warning about powering
     off a machine that ESX Agent Manager manages.

5   Select the **Configure** tab.

     The status of the ESX agency that the EAM Sample Solution deployed has changed to red.

6   Select the agency with the red status, and then right-click the agency and select **Resolve All Issues**
     from the pop-up menu. Click the **Refresh** button.

     ESX Agent Manager powers on the ESX agent virtual machine. The status of the ESX agency turns
     yellow while the ESX agent powers on, then turns green when it is running.

7   From the **vSphere ESX Agent Manager** window, choose **Virtual Machines** from the left navigation
     menu. Power off an ESX agent virtual machine, delete it from the disk, and click **Yes** to ignore the
     warning about powering off a machine that ESX Agent Manager manages.

8   Select **vSphere ESX Agent Manager > Configure**.

     The status of the ESX agency that the EAM Sample Solution deployed has changed to red.

9   Select the agency with the red status, and then right-click the agency and select **Resolve All Issues**
     from the pop-up menu. Click the **Refresh** button.

     ESX Agent Manager deploys a clone of the ESX agent that you deleted. The status of the ESX
     agency turns yellow while the ESX agent powers on, then turns green when it is running.

10 Uninstall the solution, unregister the running ESX agency, and delete its ESX agent virtual machines.

    a   Go to the **Administration > vCenter Server Extensions > EAM Sample Solution > Summary** tab and click the link in the Solutions portlet to open the EAM Sample Solution Configuration page.

    b   From the Solution Actions panel, click **Uninstall**.

    c   In the vCenter Server Extensions window, go to the **vSphere ESX Agent Manager > Configure** tab and right-click the Sample Service.

    d   From the pop-up menu, select **Delete Agency**.

The Recent Tasks panel shows the progress as the EAM Sample Solution and ESX Agent Manager uninstalls the ESX agent, powers off the virtual machine, and deletes the virtual machine and resource pool from the vCenter Server inventory.

What to do next

Use the EAM Sample Solution to deploy an ESX agent virtual machine and a vSphere Installation Bundle (VIB).

Deploy ESX Agents and a VIB Using the EAM Sample Solution

The EAM Sample Solution optionally deploys an ESX agent virtual machine and a vSphere installation bundle (VIB).

Prerequisites

- Verify that you uninstalled the simple EAM Sample Solution that you deployed in Deploy ESX Agents from OVF Using the EAM Sample Solution.

- Verify that you have configured the virtual infrastructure on which to deploy ESX agents.

    Important   If you have not configured the ESX agent network and datastore settings on the host, ESX Agent Manager sets the status to red and ESX agents do not deploy. See Configure the Virtual Infrastructure for ESX Agent Deployment.

Procedure

1 Open the `<Web_app_server_deployment_directory>/eam-sample/WEB_INF/eamri.properties` file in a text editor.

2 Change the value of the `vib.deploy` property from `false` to `true`, save and close the file.

3 Restart the EAM Sample application on the Web application server.

4 Click **Home** in the navigation bar of the vSphere Web Client, and then click **Administration**.

5 Click **vCenter Server Extensions**.

6 Click **EAM Sample Solution** in the list of running vCenter Server solutions and click the link in the **Solutions** portlet.

The EAM Sample Solution configuration page loads in a separate browser window.

7   In the **Compute resources** panel, select the host name of the ESXi host or cluster on which to deploy ESX agents and click **Update Configuration**.

The **Recent Tasks** view shows the progress of the EAM Sample Solution.

You used the EAM Sample Solution to deploy an ESX agent from a VIB.

What to do next

Implement the features that the EAM Sample Solution demonstrates, and features that the sample does not demonstrate. You can experiment with solution development by modifying the EAM Sample Solution, rebuilding, and redeploying it.

Modifying the EAM Sample Solution

You can modify the EAM Sample Solution to experiment with solution development.

You can add to the function of the EAM Sample Solution by copying the ESX Agent Manager SDK to a working folder in your development environment and adapting the Java files in `VMware-vSphere-SDK-6.7.0-build_number\SDK\eam\src` and the OVF and vSphere Installation Bundle (VIB) files in `VMware-vSphere-SDK-6.7.0-build_number\SDK\eam\war`.

For example, you can modify the sample solution to perform the following kinds of functions.

- Change the data that the solution stores in the vCenter Server database.

- Modify the OVF or VIB definitions to change the settings of the virtual machines that the solution creates.

- Modify or add new tabs that appear in the Summary page of the EAM Sample Solution.

- Change the goal state or scope of an ESX agency.

Rebuild and Redeploy the EAM Sample Solution

If you modify the sample, you must rebuild it for your modifications to take effect.

The EAM Sample Solution bundle includes an Ant build file that rebuilds the sample WAR file, `eam-sample.war`.

Prerequisites

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have Apache Ant in your development environment. For information about how to install and configure Ant, see http://ant.apache.org/.

Procedure

1   Go to the root folder of your ESX Agent Manager SDK development environment.

For example, *eam_work_folder*

2   Rebuild the EAM Sample Solution WAR file by running the `ant` command.

3 Copy the `eam-sample.war` file from the `ESX Agent Manager` SDK folder to a work directory.

4 Unzip the `eam-sample.war` file.

5 Update the `eamri.properties` file.

6 Deploy the updated EAM Sample solution to your Web application server.

You rebuilt the EAM Sample Solution.

What to do next

View the changes that you made to the EAM Sample Solution in the vSphere Web Client.

Creating vSphere Solutions

3

The vSphere SDK provides a set of APIs that you can use to create extensions for vCenter Server. The key SDK objects for extension development are `ExtensionManager` and `Extension`.

Use the `ExtensionManager` managed object in your vCenter Server instance to register a new extension. Extensions can add new objects to the vCenter inventory. Extensions define tasks, events, and faults that relate to actions that the solution performs on the objects, the events that occur in the extension, and the problems that the objects encounter. You can also use `ExtensionManager` to add user interface elements to the vSphere Web Client to allow users to interact with the objects that your extension provides to vCenter Server.

ExtensionManager adds data objects to the vSphere Extension APIs that integrate extensions as vCenter solutions that you can manage.

For information about how to develop vSphere Web Client extensions, see the *VMware vSphere Web Client Extensions Programming Guide*.

For information about developing vSphere applications, see the *vSphere Web Services SDK Programming Guide* and the *VMware vSphere API Reference* documentation.

- Contents and Structure of an Extension

  A vSphere solution must register with vCenter Server as an extension. If you register a solution with vCenter Server, you can mark virtual machines as belonging to that solution, and integrate it with ExtensionManager and ESX Agent Manager.

- Register an Extension with vCenter Server

  You must provide information about an extension when you register it with vCenter Server.

Contents and Structure of an Extension

A vSphere solution must register with vCenter Server as an extension. If you register a solution with vCenter Server, you can mark virtual machines as belonging to that solution, and integrate it with ExtensionManager and ESX Agent Manager.

vCenter Server extensions can have both a client and a server component, or they can consist of just a client component or just a server component. Each instance of an extension is represented by an `Extension` data object.

An extension can define events, tasks, faults, and privileges for performing operations on the objects that the solution exposes to vCenter Server, or on existing vCenter Server objects.

- Events inform users about occurrences in the solution.

- Tasks are operations that the extension performs.

- Faults signal errors in the extension to the end users.

- Privileges define which users or user groups can access the objects in the extension and perform tasks on them.

Solutions typically include Web pages that appear in vSphere Web Client. You can use the pages to configure the solution and to manage the objects that the solution exposes.

An extension is typically a Web application that runs in an application server or a script that vCenter Server accesses through a URL. You can implement the Web server features by using any of the following programming languages.

- Java Servlets or Java Server Pages (JSP)

- Microsoft Active Server Pages (ASP.NET)

- Common Gateway Interface (CGI) scripting

- Static or dynamic HTML pages

For example, the EAM Sample Solution uses Java, the Spring Framework, and WSDL to implement its Web application.

An extension performs the following types of tasks:

- Establishes the connection to vCenter Server.

- Registers the extension with `ExtensionManager`.

- Deploys virtual machines or vApps by using Open Virtualization Format (OVF).

- Deploys VMkernel modules or ESXi applications as vSphere Installation Bundles (VIB).

- Exposes the functions of the product that the extension integrates into vCenter Server.

- Provides a user interface to access, configure, and use the extension.

To integrate an extension as a solution with ExtensionManager, the solution must be a valid extension and it must implement the `Extension.shownInSolutionsManager` property. The solution can also provide information about itself to vCenter Server in the `ExtSolutionManagerInfo` object.

The requirements for integrating a solution with ESX Agent Manager are slightly more restrictive than for integrating with ExtensionManager. For information about requirements, see Chapter 5 Integrating Solutions with vSphere ESX Agent Manager.

For information about how to develop vSphere Web Client extensions, see the *VMware vSphere Web Client Extensions Programming Guide*.

For information about developing vSphere applications, see the *vSphere Web Services SDK Programming Guide* and the *VMware vSphere API Reference* documentation.

Key Objects in the vSphere API for Extension and Solution Development

The vSphere API contains objects that are key for developing extensions and for integrating the extensions with ExtensionManager.

Table 3-1. Key Objects in the vSphere API for Extension and Solution Development

Object	Description
ExtensionManager	You call `ExtensionManager.registerExtension()` to register an `Extension` instance with `ExtensionManager`.
Extension	Information about extension instances, including properties that you set in an `ExtSolutionManagerInfo` object to integrate an extension as a solution in ExtensionManager.
ExtensionClientInfo	Information about the client side of an extension.
ExtensionEventTypeInfo	Information about the types of events that occur on the objects in the extension.
ExtensionFaultTypeInfo	Information about the types of problems that the objects in the extension encounter.
ExtensionHealthInfo	Information about the health of a solution.
ExtensionPrivilegeInfo	Information about access privileges for the extension.
ExtensionResourceInfo	Information about resource files that contain localizable user interface text and messages that appear in vSphere Web Client in different locales.
ExtensionServerInfo	Information about the server side of an extension.
ExtensionTaskTypeInfo	Information about the types of tasks that the objects perform in the extension.
ExtExtendedProductInfo	Information about the product that the extension exposes to vCenter Server.
ExtManagedEntityInfo	Information about the objects that the extension manages.
ExtSolutionManagerInfo	Information that the solution exposes about itself.
ExtSolutionManagerInfoTabInfo	Information about the tabs that a solution adds to its Summary page.

Register an Extension with vCenter Server

You must provide information about an extension when you register it with vCenter Server.

- A description of the extension

- A unique identifier for the extension

- Version information

- URLs to any server or client interfaces that the extension exposes

- An email address for the extension administrator

- Any additional metadata that the extension defines, for example, URLs to the the company and product Web sites, the managed object reference (MoRef) of the virtual machine or vApp in which the solution is running, and so on

- Definitions of the tasks, events, faults, and privileges that the extension adds to vCenter Server

- Localization data for the task, event, fault, and privilege descriptions

- A public key which vCenter Server uses to register the extension in the registry of the vCenter Server system on which it is running.

Procedure

1 Create the Program that Manages the Extension

An extension communicates with vCenter Server across a network, so you can use any programming language to create the program that manages an extension.

2 Connect the Extension to vCenter Server

You must provide information about the vCenter Server instance to which you connect an extension. Set the details of the connection to vCenter Server in the client-side stub of the extension.

3 Set the Extension Key

Every extension that you register with vCenter Server must have a unique extension key that vCenter Server uses to identify the extension.

4 Set the Extension Product Information

You provide product information about an extension by setting properties when you instantiate the `Extension` object in the program that manages the extension. The product information that you set appears in the vSphere Web Client.

5 Set the Extension Name and Localization Resources

When you develop extensions, the vSphere Web Client can appear in different languages in different locales. You can set information that appears in the vSphere Web Client, for example, the extension name, as resources that can be translated.

6 Identify the Virtual Machines or vApps that an Extension Manages

If an extension manages virtual machines or vApps, you can identify those virtual machines or vApps as being managed by that extension. You can add icons to the objects that the extension manages that appear in the vCenter Server inventory, and warn users if they try to perform actions on those objects.

7 Set the Types of the Virtual Machines or vApps that the Extension Manages

You can set properties in the `ExtManagedEntityInfo` data object to define how the vSphere Web Client displays the different types of virtual machines and vApps that an extension deploys.

8 Set the Description for a Type of Virtual Machine or vApp that a Solution Manages

If a solution deploys different types of virtual machine or vApp, you can provide a description for each type by setting the `ExtManagedEntityInfo description` property.

9 Unregister the Extension from vCenter Server

When you develop an extension, the best practice is to provide a means of unregistering the extension from vCenter Server.

Create the Program that Manages the Extension

An extension communicates with vCenter Server across a network, so you can use any programming language to create the program that manages an extension.

The product that you are exposing as an extension to vCenter Server determines the way you create the program that will manage your extension. The managing program must implement `Extension` to provide the information that vCenter Server requires to register the extension.

The EAM Sample Solution defines the server side of the solution in the `Manager` class.

`Manager.java` performs the following tasks for the EAM Sample Solution.

- Sets up the EAM Sample Solution by obtaining the values for the extension key, vCenter Server connection, IP addresses, and port configuration from the `eamri.properties` file.

- Secures the connection to vCenter Server by using TLS.

- Defines methods to construct the URLs through which to access the server side of the extension.

- Implements the `Extension` object to register the EAM Sample Solution as an extension with vCenter Server.

- Integrates the EAM Sample Solution as a solution with ExtensionManager.

- Connects the solution to the ESX Agent Manager.

- Instantiates the class that defines the ESX agents that the solution deploys.

- Starts the solution in vCenter Server.

- Defines a task to unregister the solution from vCenter Server.

Prerequisites

- Download the vSphere ESX Agent Manager SDK.

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Open *eam_work_folder*\src\com\vmware\eam\sample\solution\Manager.java

Procedure

1 Create a program to manage your extension using the programming language of your choice.

The EAM Sample Solution defines the `Manager` class to manage the extension. `Manager` implements the Spring Framework API.

```
public class Manager implements InitializingBean {
}
```

2 Create an instance of the `Extension` data object with which to register the extension with vCenter Server.

The `Manager` class defines an internal method that instantiates an `Extension`.

```
public class Manager implements InitializingBean {

  [...]
  private Extension createExtensionObject() {
    Extension extension = new Extension();
  [...]
  }
}
```

What to do next

Connect the extension to vCenter Server.

Connect the Extension to vCenter Server

You must provide information about the vCenter Server instance to which you connect an extension. Set the details of the connection to vCenter Server in the client-side stub of the extension.

Note If you deploy your extension using the Open Virtualization Format (OVF), you can integrate it with the vCenter Extension vService. The vCenter Extension vService automates the process of registering extension with vCenter Server, so you do not need to provide any connection parameters. See Chapter 6 Integrating an Extension with the vCenter Extension vService.

To connect an extension to vCenter Server, provide the following information to the client-side stub of the connection.

- A username and password for a vCenter Server administrator account, if you do not use the vCenter Extension vService
- The extension key for the extension
- A reference to the `SessionManager` instance in the vCenter Server

The EAM Sample Solution defines the connection to vCenter Server in the `VimConnection` class.

The `Manager` class performs the following functions.

- Implements the `VimConnection` class to establish the connection to vCenter Server when the EAM Sample Solution starts.

- Uses the Spring framework to obtain the connection information from the `eamri.properties` file that you configure when you set up the EAM Sample Solution.

- Passes the connection property values to `VimConnection`.

Prerequisites

- Download the vSphere ESX Agent Manager SDK.

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Open *eam_work_folder*\src\com\vmware\eam\sample\solution\util\VimConnection.java in an editor.

Procedure

1 Create an instance of the `ManagedObjectReference` data object to define the connection to the extension.

  The `VimConnection` class creates a `ManagedObjectReference` object of type `ServiceInstance`, named `_siRef`.

```
public VimConnection(String host, int port) {
 [...]
 _siRef = new ManagedObjectReference();
 _siRef.setType("ServiceInstance");
 _siRef.setValue("ServiceInstance");
 }
```

2 Define methods to get and set the vCenter Server host, ports, username, password, connection timeout, and session cookie.

  The `VimConnection` constructor defines methods to obtain the host, ports, username, password, connection timeout, and session cookie from the information that you set in the `eamri.properties` file.

3 Connect to vCenter Server by obtaining the `SessionManager` managed object for the vCenter Server.

`VimConnection.java` defines a method named `connect()`. The `connect()` method defines a standard connection to vCenter Server that uses WSDL. The following segment shows the calls to the `SessionManager.login()` and `SessionManager.loginExtensionByCertificate()` methods that establish the connection to vCenter Server. The `_stub` variable is an instance of `VimPortType`, `_sc` is a `ServiceContent` object, and `_siRef` is the `ManagedObjectReference` object of type `ServiceInstance`.

```java
private synchronized void connect() {
  [...]
  try {
      [...]
      VimService locator = new VimService(wsdlURL, new QName("urn:vim25Service",
                                                  "VimService"));

      _stub = locator.getVimPort();
      [...]

      _sc = _stub.retrieveServiceContent(_siRef);

      ManagedObjectReference sessionManager = _sc.getSessionManager();

      if (_extensionKey == null) {
        _stub.login(sessionManager, _username, _password, null);
      }
      else {
        _stub.loginExtensionByCertificate(sessionManager, _extensionKey, null);
      }
  [...]
   _connectionStatus = ConnectionStatus.Connected;
  } catch (Exception e) {
      _logger.error(e, e);
  }
}
```

4 Register the extension by calling the `ExtensionManager.registerExtension()` method.

`VimConnection` defines a `registerExtension()` method that implements `ExtensionManager.registerExtension()`. `Manager` calls `VimConnection.registerExtension()` after it has set the properties for the extension.

```java
public void registerExtension(Extension ex) {
  try {
      Extension findExtension = _stub.findExtension(_sc.getExtensionManager(),
                                          ex.getKey());
      if (findExtension == null) {
          _stub.registerExtension(_sc.getExtensionManager(), ex);
      } else {
          _stub.updateExtension(_sc.getExtensionManager(), ex);
      }
```

```
        _stub.setExtensionCertificate(_sc.getExtensionManager(),
                                  ex.getKey(), null);
    } catch (Exception e) {
        _logger.error(e, e);
    }
}
```

You registered an extension with vCenter Server.

What to do next

Provide an extension key with which to register the extension with vCenter Server.

Set the Extension Key

Every extension that you register with vCenter Server must have a unique extension key that vCenter Server uses to identify the extension.

To ensure uniqueness, you can use the Java package-naming convention for the key value, for example `com.yourcompany.yourextension`. The EAM Sample Solution sets the extension key by defining the `extensionKey=com.vmware.eam.sample.solution` property in the `eamri.properties` file. The `Manager.java` class picks up this property by calling the `Extension.setKey()` method.

If you change the extension key in the `extensionKey` property in `eamri.properties`, you can see the changed name in `C:\Program Files\VMware\Infrastructure\VirtualCenter Server\extensions` on the vCenter Server for Windows, or in `/usr/lib/vmware-vpx/extensions/` in the vCenter Server Appliance.

You can see the extensions that you register with `ExtensionManager` in the vCenter Managed Object Browser at `https://vcenter_server_ip_address/mob/?moid=ExtensionManager`

Prerequisites

- [Download the vSphere ESX Agent Manager SDK](#).

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have opened
  eam_work_folder\src\com\vmware\eam\sample\solution\Manager.java in an editor.

Procedure

◆ Call the `Extension.setKey()` method to set the extension key.

  `Manager.java` sets the extension key to the `extensionKey` variable that the `eamri.properties` file defines.

```
public class Manager implements InitializingBean {
  [...]
  public final String EXTENSION_KEY;
  [...]
    EXTENSION_KEY = extensionKey;
  [...]
```

```
  }
  [...]
  private Extension createExtensionObject() {
    Extension extension = new Extension();
    extension.setKey(EXTENSION_KEY);
  [...]
  }
```

You set the extension key for an extension.

What to do next

Provide information about the extension to vCenter Server.

Set the Extension Product Information

You provide product information about an extension by setting properties when you instantiate the
`Extension` object in the program that manages the extension. The product information that you set
appears in the vSphere Web Client.

The EAM Sample Solution instantiates the `Extension` data object in the `Manager.java` class. The
sample solution sets `Extension` product information properties directly in `Manager.java`, but you can set
the property values in configuration files that the program that manages the extension accesses.

Prerequisites

- Download the vSphere ESX Agent Manager SDK.

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have opened
  `eam_work_folder\src\com\vmware\eam\sample\solution\Manager.java` in an editor.

Procedure

1   Provide a description for the extension by creating an instance of the `Description` data object and
    passing it to `Extension`.

```
    private Extension createExtensionObject() {
      Extension extension = new Extension();
      [...]
      Description description = new Description();
      description.setLabel("EAM Sample Solution");
      description.setSummary("This extension provides the EAM Sample Solution.");
      extension.setDescription(description);
      [...]
    }
```

2 Provide a version number for the extension by calling the `Extension.setVersion()` method.

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    extension.setVersion("0.1");
    [...]
  }
```

3 (Optional) Change the value of the version property in `Manager.java`.

4 Provide information about the vendor of the extension by calling the `Extension.setCompany()` method.

The EAM Sample Solution does not set the `vendor` property. You can set it by adding the following line of code to `Manager.java`.

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    extension.setVersion("0.1");
    extension.setCompany("your_company");
    [...]
  }
```

5 Provide URLs to Web pages for the product and for the vendor of the extension by creating an instance of the `ExtExtendedProductInfo` data object and passing it to `Extension`.

The EAM Sample Solution does not set the `companyUrl` property for `ExtExtendedProductInfo`. You can set it by adding a call to to the `ExtExtendedProductInfo.setCompanyUrl()` method in `Manager.java`:

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    ExtExtendedProductInfo extExtendedProductInfo = new
    ExtExtendedProductInfo();
    extExtendedProductInfo.setCompanyUrl("your_URL");
    extExtendedProductInfo.setProductUrl("your_product_url");
    extension.setExtendedProductInfo(extExtendedProductInfo);
    [...]
  }
```

6 (Optional) If you modified `Manager.java`, save your changes and rebuild and redeploy the EAM Sample Solution.

7 (Optional) View your changes in the vSphere Web Client.

If you changed the values of the `version`, `company`, `companyUrl`, and `productUrl` properties, your changes appear in the **Summary** tab of your solution.

You provided product information about an extension by setting properties in the `Extension` and `ExtExtendedProductInfo` data objects.

What to do next

Set the extension name and localization information.

Set the Extension Name and Localization Resources

When you develop extensions, the vSphere Web Client can appear in different languages in different locales. You can set information that appears in the vSphere Web Client, for example, the extension name, as resources that can be translated.

You provide information that requires translation in an `ExtensionResourceInfo` data object. You can add an `ExtensionResourceInfo` object for every locale that your extension supports. You set an array of `ExtensionResourceInfo` objects in the `resourceList` property of the `Extension` instance that defines your extension.

You provide onscreen messages and labels to `ExtensionResourceInfo` in a key and value pairing that you add to a `KeyValue` array in the `ExtensionResourceInfo data` property. You can set the values for the `KeyValue` pair directly in the `ExtensionResourceInfo` object, or you can refer to entries in resource files that contain the message text in different languages, according to the locale in which vSphere is running. For simplicity, the EAM Sample Solution sets the data directly in the `KeyValue` implementation.

You provide a two-character ISO-639 language ID for the `KeyValue locale` property, and set the `module` property to the type of resource to which this locale applies. For example, you can set the module value to `task`, `event`, `auth`, or `extension`, depending on whether the messages that the resource contains relate to tasks, events, privileges, or extensions.

Prerequisites

- [Download the vSphere ESX Agent Manager SDK](#).

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have opened
  eam_work_folder\src\com\vmware\eam\sample\solution\Manager.java in an editor.

Procedure

1 Create an instance of the `ExtensionResourceInfo` data object.

   `Manager.java` instantiates `ExtensionResourceInfo` in the implementation of `Extension`.

   ```
   private Extension createExtensionObject() {
       Extension extension = new Extension();
       [...]
       ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
       [...]
   }
   ```

2 Set the locale and module properties for the `ExtensionResourceInfo` object.

`Manager.java` sets the default locale to en and applies this locale to the `Extension` instance, `extension`.

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
    extensionResourceInfo.setLocale("en");
    extensionResourceInfo.setModule("extension");
    [...]
}
```

3 Provide the data to the `ExtensionResourceInfo` in the form of a `KeyValue` array.

The `label` property is a property of the `Description` object, that `Extension` implements, and defines the name of the extension as it appears in the vSphere Web Client.

`Manager.java` adds the text `EAM Sample Solution` as the value of the `label` property

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
    extensionResourceInfo.setLocale("en");
    extensionResourceInfo.setModule("extension");
    KeyValue keyValue = new KeyValue();
    keyValue.setKey(EXTENSION_KEY + ".label");
    keyValue.setValue("EAM Sample Solution");
    [...]
}
```

4 Call the `ExtensionResourceInfo.getData()` method to add the `KeyValue` array that contains the localization data to the `data` property of the `ExtensionResourceInfo` object.

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
    extensionResourceInfo.setLocale("en");
    extensionResourceInfo.setModule("extension");
    KeyValue keyValue = new KeyValue();
    keyValue.setKey(EXTENSION_KEY + ".label");
    keyValue.setValue("EAM Sample Solution");
    extensionResourceInfo.getData().add(keyValue);
    [...]
}
```

5 (Optional) Add another `KeyValue` object to the `ExtensionResourceInfo data` property that adds a description of the extension for a given locale.

For example, you can add the following description to `Manager.java` in a `KeyValue` object named `keyValue_summary`.

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
    extensionResourceInfo.setLocale("en");
    extensionResourceInfo.setModule("extension");
    KeyValue keyValue = new KeyValue();
    keyValue.setKey(EXTENSION_KEY + ".label");
    keyValue.setValue("EAM Sample Solution");

    KeyValue keyValue_summary = new KeyValue();
    keyValue_summary.setKey(EXTENSION_KEY + ".summary");
    keyValue_summary.setValue("Brief description");

    extensionResourceInfo.getData().add(keyValue);
    extensionResourceInfo.getData().add(keyValue_summary);
    [...]
}
```

6 Call the `Extension.getResourceList()` method to pass the `ExtensionResourceInfo` object to the `Extension` instance.

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
    extensionResourceInfo.setLocale("en");
    extensionResourceInfo.setModule("extension");
    KeyValue keyValue = new KeyValue();
    keyValue.setKey(EXTENSION_KEY + ".label");
    keyValue.setValue("EAM Sample Solution");

    KeyValue keyValue_summary = new KeyValue();
    keyValue_summary.setKey(EXTENSION_KEY + ".summary");
    keyValue_summary.setValue("Brief description");

    extensionResourceInfo.getData().add(keyValue);
    extensionResourceInfo.getData().add(keyValue_summary);

    extension.getResourceList().add(extensionResourceInfo);
    [...]
}
```

7   (Optional) Add more `ExtensionResourceInfo` instances to provide localized text that displays when
    the extension runs in different locales.

    For example, you can add an `ExtensionResourceInfo` instance to `Manager.java` to provide a
    French translation of the extension name.

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
    extensionResourceInfo.setLocale("en");
    extensionResourceInfo.setModule("extension");
    KeyValue keyValue = new KeyValue();
    keyValue.setKey(EXTENSION_KEY + ".label");
    keyValue.setValue("EAM Sample Solution");

    KeyValue keyValue_summary = new KeyValue();
    keyValue_summary.setKey(EXTENSION_KEY + ".summary");
    keyValue_summary.setValue("Brief description");

    extensionResourceInfo.getData().add(keyValue);
    extensionResourceInfo.getData().add(keyValue_summary);

    ExtensionResourceInfo extensionResourceInfo_FR = new ExtensionResourceInfo();
    extensionResourceInfo_FR.setLocale("fr");
    extensionResourceInfo_FR.setModule("extension");
    KeyValue keyValue_FR = new KeyValue();
    keyValue_FR.setKey(EXTENSION_KEY + ".label");
    keyValue_FR.setValue("Exemple d'une Solution EAM");

    extensionResourceInfo_FR.getData().add(keyValue_FR);

    extension.getResourceList().add(extensionResourceInfo);
    extension.getResourceList().add(extensionResourceInfo_FR);
    [...]
}
```

8   (Optional) If you modified `Manager.java`, save your changes and rebuild and redeploy the EAM
    Sample Solution.

    If you changed the `label` value, the extension appears in vCenter Server Extensions with the new
    name. If you added an `ExtensionResourceInfo` object for a different locale, the localized text that
    you added appears in vCenter Server Extensions when you connect the EAM Sample Solution to a
    vCenter Server instance that runs in that locale.

You added localizable message resources to the extension, so that onscreen messages and labels that
your extension provides can appear in different languages in different locales.

Identify the Virtual Machines or vApps that an Extension Manages

If an extension manages virtual machines or vApps, you can identify those virtual machines or vApps as being managed by that extension. You can add icons to the objects that the extension manages that appear in the vCenter Server inventory, and warn users if they try to perform actions on those objects.

For example, the EAM Sample Solution creates ESX agent virtual machines. The ESX agent virtual machines that it manages appear in the vCenter Server inventory with an icon that identifies them as ESX agents. A panel appears in the **Summary** tab for those virtual machines that identifies ESX Agent Manager as the solution that manages them. If you attempt to perform an operation directly on an ESX agent virtual machine, you see a warning that instructs you to perform the operation by using ESX Agent Manager, rather than by performing it manually on the virtual machine. For the list of operations that trigger warnings when you try to perform them on a virtual machine or vApp that an extension manages, see Operations that Trigger Warnings from Extensions.

You identify a virtual machine or vApp as belonging to an extension by setting the `managedBy` property in the `VirtualMachineConfigSpec` or `VAppConfigSpec` implementations that define the virtual machines or vApps that the extension manages. You set the `managedBy` property to a `ManagedByInfo` object.

You identify the extension that manages a virtual machine or vApp by setting the `extensionKey` property in the `ManagedByInfo` object. You specify different types of virtual machine or vApp that an extension manages by setting the `type` property in `ManagedByInfo`.

If you set the `ManagedByInfo type` property in the virtual machine or vApp definition, you can pass this value to the `ExtManagedEntityInfo` implementation in the extension definition. `ExtManagedEntityInfo` applies descriptions and icons to all the virtual machines or vApps of this type that the extension manages.

For information about configuring and deploying virtual machines programmatically, see the *vSphere Web Services SDK Programming Guide* and the *VMware vSphere API Reference*.

Prerequisites

You have a vCenter Server extension that manages virtual machines or vApps.

Note   The EAM Sample Solution uses ESX Agent Manager to deploy ESX agent virtual machines. ESX Agent Manager sets the `ManagedByInfo` properties on these virtual machines to mark them as belonging to ESX Agent Manager. The EAM Sample Solution does not set the `ManagedByInfo` properties itself. The code extracts in this procedure are not taken from the source files of the EAM Sample Solution.

Procedure

1   In the program that defines the virtual machines or vApps that an extension deploys, create an instance of `VirtualMachineConfigSpec` or `VAppConfigSpec`.

For example, you can instantiate `VirtualMachineConfigSpec`.

```
VirtualMachineConfigSpec configSpec = new VirtualMachineConfigSpec();
```

2 Create an instance of `ManagedByInfo`.

```
ManagedByInfo managedByInfo = new ManagedByInfo();
```

3 Set the `ManagedByInfo` `extensionKey` property to the extension key of the extension that deploys the virtual machines or vApps.

Use the extension key that you define in the implementation of `Extension` in your extension.

```
managedByInfo.setExtensionKey("com.yourcompany.yourextension");
```

4 Set the `ManagedByInfo` `type` property to identify the virtual machine or vApp as being of a certain type.

Set the `type` property to a descriptive name for this type of virtual machine. In the implementation of `Extension` that manages the extension, you can apply icons and descriptions to all virtual machines or vApps of this type that the extension deploys.

```
managedByInfo.setType("your_vm_type"");
```

5 Pass the `ManagedByInfo` instance to the `managedBy` property of the `VirtualMachineConfigSpec` or `VAppConfigSpec` implementation.

```
configSpec.setManagedBy(managedByInfo);
```

You set the `managedBy` properties in a virtual machine or vApp definition, to identify the virtual machines or vApps as being of a certain type and as belonging to an extension.

What to do next

Set the types of virtual machines or vApps that an extension manages by implementing `ExtManagedEntityInfo`.

Operations that Trigger Warnings from Extensions

If an extension deploys virtual machines or vApps, you can configure them as being virtual machines or vApps that the extension manages. If users attempt to perform operations on these virtual machines or vApps, they see a warning.

You configure a virtual machine or vApp as belonging to an extension by implementing `ManagedByInfo` in the configuration of that virtual machine or vApp.

The following operations result in a warning if a user attempts to run them from the vSphere Web Client on a virtual machine or vApp that an extension manages:

- `PowerOn`
- `PowerOff`
- `Suspend`

- Reset

- RebootGuest

- ShutdownGuest

- StandbyGuest

- Edit

- Clone

- Templatize

- ConvertToVirtualMachine

- Deploy

- CloneTemplateToTemplate

- CloneVmToTemplate

- TakeSnapshot

- GotoSnapshot

- StartRecording

- Remove

- CreateSecondaryVm

- DisableFaultTolerance

- TurnFaultToleranceOff

- MigrateHost

- Migrate

- RelocateDatastore

- Rename

Set the Types of the Virtual Machines or vApps that the Extension Manages

You can set properties in the `ExtManagedEntityInfo` data object to define how the vSphere Web Client displays the different types of virtual machines and vApps that an extension deploys.

If you set the `ManagedByInfo type` property in a virtual machine or vApp configuration, you can configure an extension to apply certain properties to all the virtual machines or vApps of that type that it deploys. You can implement the `ExtManagedEntityInfo` data object to apply a common description and an icon that appears in the vCenter inventory for all virtual machines of that type that the extension deploys.

You add an array of `ExtManagedEntityInfo` objects to an extension by setting the `Extension` `managedEntityInfo` property. If your extension deploys different types of virtual machine or vApp, you can add one `ExtManagedEntityInfo` object to the array for each type of virtual machine or vApp. By creating several instances of `ExtManagedEntityInfo` with different properties, you can differentiate the different types of virtual machines or vApps that the extension deploys in the vCenter inventory.

Prerequisites

Verify that you have set the `managedBy` property in the configuration of the virtual machines or vApps that an extension deploys. See Identify the Virtual Machines or vApps that an Extension Manages.

Procedure

1 Create an instance of `ExtManagedEntityInfo` in the implementation of `Extension` in the program that manages the extension.

```
Extension extension = new Extension();
ExtManagedEntityInfo extManagedEntityInfo = new ExtManagedEntityInfo();
```

2 Set the `ExtManagedEntityInfo type` property to the `type` value that you set in the `ManagedByInfo` property in the program that defines the virtual machine or vApp to deploy.

For example, set the `type` property to the same value as the `type` property of the `ManagedByInfo` object.

```
extManagedEntityInfo.setType("your_vm_type");
```

3 Add the `ExtManagedEntityInfo` instance to the array of `ExtManagedEntityInfo` instances in the `Extension managedEntityInfo` property.

```
extension.getManagedEntityInfo().add(extManagedEntityInfo);
```

You set the types of virtual machine or vApp that the extension manages. When you set different types of virtual machine or vApp in an extension, you can change how those types of object appear in the vSphere inventory.

What to do next

Set a description for the types of object that the extension manages.

Set the Description for a Type of Virtual Machine or vApp that a Solution Manages

If a solution deploys different types of virtual machine or vApp, you can provide a description for each type by setting the `ExtManagedEntityInfo description` property.

The text that you set in the `ExtManagedEntityInfo description` property allows you to add a description to the types of virtual machines or vApps that your solution deploys. If a solution deploys different types of virtual machine, you can create several `ExtManagedEntityInfo` instances, each with a different description.

Prerequisites

- Verify that you have set the `managedBy` property in the configuration of the virtual machines or vApps that an extension deploys. See Identify the Virtual Machines or vApps that an Extension Manages.

- Verify that you have created an instance of `ExtManagedEntityInfo` in the program that defines an extension.

Note   The EAM Sample Solution uses ESX Agent Manager to deploy ESX agent virtual machines. ESX Agent Manager sets the `ManagedByInfo` properties on these virtual machines to mark them as belonging to ESX Agent Manager. The EAM Sample Solution does not set the `ManagedByInfo` properties itself. The code extracts in this procedure are not taken from the source files of the EAM Sample Solution.

Procedure

- Call the `ExtManagedEntityInfo.setDescription()` method to set the `ExtManagedEntityInfo description` property for a type of virtual machine or vApp that the solution deploys.

```
Extension extension = new Extension();
ExtManagedEntityInfo extManagedEntityInfo = new ExtManagedEntityInfo();
extManagedEntityInfo.setType("your_vm_type");
extManagedEntityInfo.setDescription("Description of this type of virtual machine or vApp.");
extension.getManagedEntityInfo().add(extManagedEntityInfo);
```

You added a description to all virtual machines or vApps of a certain type that your solution deploys. The description appears in the vSphere Web Client when the solution registers with vCenter Server.

Unregister the Extension from vCenter Server

When you develop an extension, the best practice is to provide a means of unregistering the extension from vCenter Server.

You unregister an extension from vCenter Server by calling the `ExtensionManager.unregisterExtension()` method. To unregister an extension, you pass to `unregisterExtension()` the `ManagedObjectReference` instance that identifies the `ExtensionManager` for the vCenter Server and the extension key for the extension.

Prerequisites

- Download the vSphere ESX Agent Manager SDK.

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have opened
  `eam_work_folder\src\com\vmware\eam\sample\solution\Manager.java` in an editor.

Procedure

◆ Define a method that unregisters the extension by calling
   `ExtensionManager.unregisterExtension()`.

   `Manager.java` defines a method named `cleanup()` that unregisters the EAM Sample Solution from
   vCenter Server. In the EAM Sample Solution, the `VimConnection.java` class defines the connection
   to vCenter Server. `Manager.java` creates an instance of `VimConnection`, `_vimConnection`, with
   which it connects to vCenter Server. The `VimConnection.getStub()` method obtains the port on
   which the vCenter Server is running.

```
public void cleanup()
  throws NotFoundFaultMsg, RuntimeFaultFaultMsg {
    [...]
    _vimConnection.getStub().unregisterExtension(
                            _vimConnection.getExtensionManager(),
                            EXTENSION_KEY);
}
```

You defined a method to unregister an extension from vCenter Server.

Integrating Solutions with vCenter Server Extensions

4

You can integrate a vCenter Server extension with the vCenter Server Extensions by implementing certain data objects and properties from the vSphere Management SDK.

vCenter Server Extensions is the principal location in the vSphere Web Client where a vSphere administrator can find information about the solutions that are running in vCenter Server. vCenter Server Extensions displays the name and version of the solution.

Register with vCenter Server Extensions all of the solutions that you develop. Typically, you create at least one tab to configure and monitor a solution. Solution configuration tasks to perform in a configuration tab include setting up the solution, specifying any database connections, and setting logging levels.

- SDK Objects for Integration with vCenter Server Extensions

  You integrate a solution with vCenter Server Extensions by implementing certain properties and data objects from the vSphere Management SDK in a vCenter Server extension.

- Integrate a Solution with vCenter Server Extensions

  The information in this section examines the classes of the EAM Sample Solution to show you how to add a solution to vCenter Server Extensions. Some information modifies the EAM Sample Solution to add new content to it.

SDK Objects for Integration with vCenter Server Extensions

You integrate a solution with vCenter Server Extensions by implementing certain properties and data objects from the vSphere Management SDK in a vCenter Server extension.

vCenter Server Extensions provides a graphical view in the vSphere Web Client of the list of extensions that the `ExtensionManager` managed object exposes to vCenter Server. To integrate a solution with vCenter Server Extensions, you set properties in the implementation of the `Extension` data object that defines an instance of a vCenter Server extension. Some of the properties that you set in the `Extension` implementation refer to data objects in the vSphere Management SDK that are specific to vCenter Server Extensions.

- Properties of the Extension Data Object That Relate to the vCenter Server Extensions

  Certain properties of the `Extension` data object are specific to vCenter Server Extensions.

- vCenter Server Extensions Data Objects

  You set properties in the `Extension` implementation to integrate a solution with the vCenter Server Extensions. Some of the properties that you set refer to data objects in the vSphere Management SDK that are specific to the vCenter Server Extensions.

Properties of the Extension Data Object That Relate to the vCenter Server Extensions

Certain properties of the `Extension` data object are specific to vCenter Server Extensions.

You can use the properties of the `Extension` data object to integrate a solution with vCenter Server Extensions.

Properties	Type	Description
shownInSolutionManager	Boolean	You must set this property to **true** for a solution to appear in vCenter Server Extensions.
solutionManagerInfo	`ExtSolutionManagerInfo` data object	Provides the names for tabs and URLs to the Web applications that define the contents of the tabs.
extendedProductInfo	`ExtExtendedProductInfo` data object	Provides information about the solution, such as URLs to product and vendor Web pages.
managedEntityInfo	`ExtManagedEntityInfo` data object that represent virtual machines and vApps	Provides information about the objects that the solution manages, such as their type, a description of the objects, and an icon to represent the objects in the vCenter Server inventory.

vCenter Server Extensions Data Objects

You set properties in the `Extension` implementation to integrate a solution with the vCenter Server Extensions. Some of the properties that you set refer to data objects in the vSphere Management SDK that are specific to the vCenter Server Extensions.

For information about the properties and implementation details of each data object, see the VMware vSphere API Reference documentation.

Table 4-1. Data Objects in the vSphere Management SDK that relate to the vCenter Server Extensions

Data Object	Description
ExtSolutionManagerInfo	Provides an array of `ExtSolutionManagerInfoTabInfo` objects that implement the tabs that the solution adds to vCenter Server Extensions.
ExtExtendedProductInfo	Provides URLs to information about the solution, such as the solution vendor's Web site, a management interface for the solution, and a description of the solution. Also provides links to the virtual machine or vApp that runs the solution.

Table 4-1. Data Objects in the vSphere Management SDK that relate to the vCenter Server
Extensions (Continued)

Data Object	Description
ExtManagedEntityInfo	Provides information about the objects that the solution manages, such as a description, an icon to represent the objects that the solution creates in the vCenter Server inventory, and the types of the objects.
ExtSolutionManagerInfoTabInfo	Provides label for the solution in the Solutions portlet on the Summary page. You can also add URLs to the Web application.
ManagedByInfo	Identifies types of virtual machine or vApp as belonging to a solution. You can configure the virtual machines or vApps that a solution deploys so that they appear in the vCenter Server inventory with special icons and descriptions. You can also warn users about performing operations on virtual machines or vApps that a solution manages.

Integrate a Solution with vCenter Server Extensions

The information in this section examines the classes of the EAM Sample Solution to show you how to add
a solution to vCenter Server Extensions. Some information modifies the EAM Sample Solution to add
new content to it.

Procedure

1 Add a Solution to vCenter Server Extensions

   You add a solution to vCenter Server Extensions by setting the shownInSolutionManager property
   in the implementation of the Extension data object that manages the solution.

2 Set the Icon for a Type of Virtual Machine or vApp That an Extension Manages

   You can provide an icon for each type of virtual machine or vApp that an extension manages by
   setting the ExtManagedEntityInfo smallIconUrl property.

3 Add Tabs to the EAM Sample Solution

   When you create a solution, you can add tabs to your solution to allow users to configure the
   solution and to access the functions of the solution in the vSphere Web Client.

4 Set Up Health Monitoring for a Solution

   vCenter Server pulls data from solutions about their health status. Solutions expose health data by
   providing a URL to an XML file that defines the health model of the solution.

Add a Solution to vCenter Server Extensions

You add a solution to vCenter Server Extensions by setting the shownInSolutionManager property in
the implementation of the Extension data object that manages the solution.

The EAM Sample Solution sets the shownInSolutionManager property in the Manager.java class.

Prerequisites

- Download the vSphere ESX Agent Manager SDK.

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have opened
  eam_work_folder\src\com\vmware\eam\sample\solution\Manager.java in an editor.

Procedure

1   Call the ExtSolutionManagerInfo() constructor to create an instance of
    ExtSolutionManagerInfo in the class that implements Extension.

    The following lines in the Manager.java class show how the EAM Sample Solution creates an
    instance of ExtSolutionManagerInfo.

    ```
    private Extension createExtensionObject() {
      Extension extension = new Extension();

      [...]
      ExtSolutionManagerInfo extSolutionManagerInfo =
        new ExtSolutionManagerInfo();
      [...]

    }
    ```

2   Call the setSolutionManagerInfo() method to set the ExtSolutionManagerInfo instance in the
    solution implementation.

    The following lines in the Manager.java class show how the EAM Sample Solution calls the
    setSolutionManagerInfo() method.

    ```
    private Extension createExtensionObject() {
      Extension extension = new Extension();

      [...]
      ExtSolutionManagerInfo extSolutionManagerInfo =
        new ExtSolutionManagerInfo();
      [...]
      extension.setSolutionManagerInfo(extSolutionManagerInfo);

    }
    ```

3   Set the value of the `shownInSolutionManager` property to true by calling the
`setShownInSolutionManager()` method on the implementation of the `Extension` object.

The following lines in the `Manager.java` class show how the EAM Sample Solution sets the
`shownInSolutionManager` property.

```
private Extension createExtensionObject() {
  Extension extension = new Extension();

  [...]
  ExtSolutionManagerInfo extSolutionManagerInfo =
    new ExtSolutionManagerInfo();
  [...]
  extension.setSolutionManagerInfo(extSolutionManagerInfo);
  extension.setShownInSolutionManager(true);
  [...]

}
```

You set the `shownInSolutionManager` property in the implementation of the `Extension` data object that
defines the solution. By setting the `shownInSolutionManager` property to `true`, a solution appears in
vCenter Server Extensions when it registers with vCenter Server.

Set the Icon for a Type of Virtual Machine or vApp That an Extension Manages

You can provide an icon for each type of virtual machine or vApp that an extension manages by setting
the `ExtManagedEntityInfo smallIconUrl` property.

The icon that you set in the `ExtManagedEntityInfo smallIconUrl` property allows you to identify the
types of virtual machines or vApps that your extension deploys. If an extension deploys different types of
virtual machines, you can create several `ExtManagedEntityInfo` instances, with a different icon for each
type. The virtual machines that the extension deploys appear in the vCenter Server inventory with the
icons that you set.

The icon image must be in the PNG format and must measure 16 by 16 pixels. You must save the icon
image to an appropriate location in the Web application that defines the extension. You provide a URL or
path to the image to the `ExtManagedEntityInfo` instance for the type of virtual machine or vApp that this
icon represents. Extensions access the icon image by using HTTP. Extensions do not support HTTPS
URLs to icon images.

Prerequisites

▪   Verify that you have set the `managedBy` property in the configuration of the virtual machines or vApps
that an extension deploys. See Identify the Virtual Machines or vApps that an Extension Manages.

■ Verify that you have created an instance of `ExtManagedEntityInfo` in the program that defines an extension.

Note The EAM Sample Solution uses ESX Agent Manager to deploy ESX agent virtual machines. ESX Agent Manager sets the `ManagedByInfo` properties on these virtual machines to mark them as belonging to ESX Agent Manager. The EAM Sample Solution does not set the `ManagedByInfo` properties itself. The code extracts in this procedure are not taken from the source files of the EAM Sample Solution.

Procedure

1 Create an icon image of type `PNG` and of dimensions 16 by 16 pixels.

2 Save the icon image to an appropriate location in the Web application that defines your an extension.

3 Call `ExtManagedEntityInfo.setSmallIconUrl()` to set the `ExtManagedEntityInfo` `smallIconUrl` property for a type of virtual machine or vApp that the extension deploys.

```
Extension extension = new Extension();
ExtManagedEntityInfo extManagedEntityInfo = new ExtManagedEntityInfo();
extManagedEntityInfo.setType("your_vm_type");
extManagedEntityInfo.setDescription("Description of this type of virtual machine or vApp.");
extManagedEntityInfo.setSmallIconUrl("path_to_PNG_image");
extension.getManagedEntityInfo().add(extManagedEntityInfo);
```

Virtual machines and vApps that the an extension deploys appear in the vCenter Server inventory with the icon that you set.

Add Tabs to the EAM Sample Solution

When you create a solution, you can add tabs to your solution to allow users to configure the solution and to access the functions of the solution in the vSphere Web Client.

You define the content of tabs as dynamic Web pages. For example, you can define tabs that allow you to perform the following kinds of actions.

■ Configure the solution.

■ Show events triggered by the solution events or vCenter Server.

■ Deploy predefined virtual machines or vApps of different types.

■ Monitor the solution application or the virtual machines that it deploys.

■ Uninstall the solution.

You create tabs by implementing the `ExtSolutionManagerInfoTabInfo` data object in the program that manages the solution. You set properties in `ExtSolutionManagerInfoTabInfo` to provide a label for the tabs and a URL to the dynamic Web pages that provide the content of the tabs. You create one `ExtSolutionManagerInfoTabInfo` instance for each tab that you add to your solution. You pass that instance to `ExtSolutionManagerInfo` in an array.

How you define the content of the tabs depends on the function of the application that the solution adds to vCenter Server.

The EAM Sample Solution adds a Configuration portlet to the Summary page of the solution. With the EAM Sample Solution Configuration page, you can deploy ESX agents on the hosts that vCenter Server manages and to uninstall the solution. You find the implementation of the EAM Sample Solution Configuration page in
eam_sample_solution_working_folder\src\com\vmware\eam\sample\solution\ui.

Table 4-2. Elements of the EAM Sample Solution Configuration Page

File	Description
ConfigModel.java	Defines the actions of the EAM Sample Solution Configuration page by calling methods that Manager.java and AgentHandler.java define.
ConfigController.java	Constructs the Web page that appears for the EAM Sample Solution from the *eam_sample_solution_working_folder*\war\WEB-INF\jsp\config.jsp file.

The EAM Sample Solution implements ExtSolutionManagerInfoTabInfo in the Manager.java class.

Prerequisites

- Download the vSphere ESX Agent Manager SDK.

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have opened
  eam_work_folder\src\com\vmware\eam\sample\solution\Manager.java in an editor.

- Examine the code of the ConfigController.java and ConfigModel.java classes and the config.jsp file to see how they expose the functions of the EAM Sample Solution Configuration page.

Procedure

1   Create an instance of ExtSolutionManagerInfo in the program that manages the solution.

    The following lines in the Manager.java class show how the EAM Sample Solution creates an instance of ExtSolutionManagerInfo.

    ```
    private Extension createExtensionObject() {
      Extension extension = new Extension();
      [...]
      ExtSolutionManagerInfo extSolutionManagerInfo =
        new ExtSolutionManagerInfo();
      [...]
    }
    ```

2   Create an instance of `ExtSolutionManagerInfoTabInfo` to contain the name of the page and a link to the Web page that defines its contents.

```
ExtSolutionManagerInfoTabInfo extSolutionManagerInfoTabInfo =
    new ExtSolutionManagerInfoTabInfo();
```

3   Call the `ExtSolutionManagerInfoTabInfo.setLabel()` method to provide a name for the page.

`Manager.java` names the EAM Sample Solution Configuration page.

```
extSolutionManagerInfoTabInfo.setLabel("Configuration");
```

4   Call the `ExtSolutionManagerInfoTabInfo.setUrl()` method to provide a URL to the Web page that defines the contents of the EAM Sample Solution Configuration page.

`Manager.java` uses the `config.html` Web page to define the contents of the configuration page.

```
extSolutionManagerInfoTabInfo.setUrl("/config.html");
```

5   Add the tab to the array of `ExtSolutionManagerInfoTabInfo` instances that define the tabs for the solution in the `ExtSolutionManagerInfo` object.

```
extSolutionManagerInfo.getTab().add(extSolutionManagerInfoTabInfo);
```

You added solution-specific tabs that appear on the Summary page of the EAM Sample Solution.

What to do next

Set up health monitoring for the solution and the objects that it manages.

Set Up Health Monitoring for a Solution

vCenter Server pulls data from solutions about their health status. Solutions expose health data by providing a URL to an XML file that defines the health model of the solution.

All solutions must provide an XML file that specifies their health model. The health specification XML file for your solution must conform to the VMware Health Service extensible schema definition (XSD). See Solution Health XML Schema for the complete VMware solution health schema.

Solutions expose health data about themselves to vCenter Server by publishing XML documents that declare different health statuses, depending on the events that occur in the solution. You provide messages to accompany each health status in the XML document that the solution exposes to vCenter Server. Solutions can provide the following statuses to vCenter Server:

- alert

- warning

- info

When you develop a solution, you must include a program or function that generates the health status XML file for the solution. The health status of a solution can be red, yellow, or green.

You must pass a URL to the XML file that defines the health model for a solution to the `ExtensionHealthInfo` object in the program that manages the solution.

The EAM Sample Solution provides two classes that provide health data about the solution. You find these classes in *eam_work_folder*\src\com\vmware\eam\sample\solution\health.

- `VimHealthProvider.java` defines the health statuses for the EAM Sample Solution.

- `HealthStatusServlet.java` dynamically creates the XML file in which the solution exposes health status data. For an example of the XML file that `HealthStatusServlet.java` creates, see Example: Contents of the EAM Sample Solution Health XML File .

In the EAM Sample Solution, the `Manager.java` class implements `ExtensionHealthInfo`. The `Manager.java` class sets a URL to a health definition XML file in the `ExtensionHealthInfo url` property. The URL that `Manager.java` provides is the path to the `health.xml` file that `HealthStatusServlet.java` creates.

Prerequisites

- Download the vSphere ESX Agent Manager SDK.

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have opened *eam_work_folder*\src\com\vmware\eam\sample\solution\Manager.java in an editor.

- Examine the code of the `VimHealthProvider.java` and `HealthStatusServlet.java` classes to see how they extract health data from the EAM Sample Solution and publish it in an XML file.

Procedure

1 Create a program that defines the health statuses of the solution.

   The EAM Sample Solution defines the health statuses in the `VimHealthProvider.java` class.

2 Create a program that creates an XML file that exposes the health status data, conforming to the VMware health service XSD.

   The EAM Sample Solution creates the health data XML file in the `HealthStatusServlet.java` class. `HealthStatusServlet.java` implements `VimHealthProvider` to extract the health status from the solution. The class creates an XML file, `health.xml`, that exposes the health data about the solution according to the health status that `VimHealthProvider` provides.

3   Create an instance of `ExtensionHealthInfo` in the program that manages the solution.

The EAM Sample Solution implements `ExtensionHealthInfo` in `Manager.java`.

```
private Extension createExtensionObject() {
  Extension extension = new Extension();
  [...]
  ExtensionHealthInfo healthInfo = new ExtensionHealthInfo();
  [...]
}
```

4   Call the `ExtensionHealthInfo.setUrl()` method to set the URL at which the solution publishes its health data XML file.

The EAM Sample Solution publishes an XML file, `health.xml`, that the `HealthStatusServlet.java` class generates.

```
healthInfo.setUrl(_url.toString() + "/health/health.xml");
```

5   Call the `Extension.setHealthInfo()` method to add the `ExtensionHealthInfo` object to the solution

`Manager.java` provides a link to the `health.xml` XML file that the `HealthStatusServlet.java` class generates.

```
private Extension createExtensionObject() {
  Extension extension = new Extension();
  [...]
  ExtensionHealthInfo healthInfo = new ExtensionHealthInfo();
  healthInfo.setUrl(_url.toString() + "/health/health.xml");
  extension.setHealthInfo(healthInfo);
  [...]
}
```

Example: Contents of the EAM Sample Solution Health XML File

The EAM Sample Solution publishes health data at
`http://<solution_ip_address>:<solution_port>/eam-sample/health/health.xml`. The
`HealthStatusServlet.java` class in the EAM Sample Solution generates this file when the solution
starts. This example shows the XML file that `HealthStatusServlet.java` generates when the EAM
Sample Solution is running correctly.

```
<vimhealth schemaVersion="1.0">
  <health id="com.vmware.eam.sample.solution">
    <name>EAM Sample Solution</name>
    <status>green</status>
```

```
    <message id="com.vmware.eam.sample.solution" level="info" time="current date and
time">Running</message>
  </health>
</vimhealth>
```

Solution Health XML Schema

Solutions must provide data about their health status to vCenter Server in XML documents.

The health status XML documents that a solutions pushes to vCenter Server must conform to the VMware health schema.

```xml
<?xml version="1.0" encoding="utf-8"?>
 <xs:schema targetNamespace="http://www.vmware.com/vi/healthservice"
            elementFormDefault="qualified"
            xmlns="http://www.vmware.com/vi/healthservice"
            xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">

   <xs:complexType name="healthType">
    <xs:sequence>
      <xs:element name="name" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="status" minOccurs="1" maxOccurs="1">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="green" />
            <xs:enumeration value="yellow" />
            <xs:enumeration value="red" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="message" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType mixed="true">
          <xs:sequence>
            <xs:element name="param" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="id" type="xs:string" use="required"/>
          <xs:attribute name="level" use="required">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="info" />
                <xs:enumeration value="warning" />
                <xs:enumeration value="alert" />
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="time" type="xs:dateTime" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="health" type="healthType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required"/>
   </xs:complexType>

   <xs:element name="vimhealth">
```

```
    <xs:complexType>
      <xs:sequence>
        <xs:element name="health" type="healthType" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="schemaVersion" type="xs:decimal" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Integrating Solutions with vSphere ESX Agent Manager

5

vSphere ESX Agent Manager is a standard vCenter Server solution that allows other solutions to deploy, monitor, and manage ESX agents on ESXi hosts.

ESX Agent Manager performs the following functions:

- Provisions ESX agent virtual machines for solutions.

- Monitors changes to the ESX agent virtual machines and their scope in vCenter Server.

- Reports configuration issues in the ESX agents to the solution.

- Integrates agent virtual machines with vSphere features such as Distributed Resource Scheduler (DRS), Distributed Power Management (DPM), vSphere High Availability (HA), fault tolerance, maintenance mode, and operations such as adding and removing hosts to and from clusters.

A solution can add functions to an ESXi host by deploying an ESX agent virtual machine and an optional vSphere Installation Bundle (VIB) that provide this function.

Every vCenter Server instance contains a running ESX Agent Manager.

- Benefits of Integrating Solutions with ESX Agent Manager

  Integrating a solution with ESX Agent Manager has several benefits and eliminates potential problems that can occur when you deploy ESX agent virtual machines.

- Overview of vSphere ESX Agent Manager Architecture

  ESX Agent Manager is an intermediary between vCenter Server and a solution. ESX Agent Manager provides a WSDL based SOAP API that you can use to create solutions that deploy and monitor ESX agent virtual machines on ESXi hosts, and integrate the ESX agents with Distributed Resource Scheduler (DRS), Distributed Power Management (DPM), High Availability (HA), and other vSphere features.

- Configuration Components of ESX Agencies

  For a solution to integrate with ESX Agent Manager, it must register with vCenter Server as an extension, and create an ESX agency that defines the configuration of the solution and the ESX agents that the solution deploys.

- Requirements for Integrating a Solution with ESX Agent Manager

  To integrate with ESX Agent Manager, a solution must meet certain requirements.

- Authenticating Against ESX Agent Manager

  vCenter Server handles the authentication of ESX Agent Manager clients, so a solution must first log in to vCenter Server before it can call the methods of the ESX Agent Manager API.

- Availability of ESX Agent Virtual Machines

  You can configure ESX agencies so that they only mark ESX agent virtual machines as available after the client of the agency has performed additional configuration after provisioning or powering on the ESX agent virtual machine.

- Monitoring and Resolving ESX Agent Issues

  With the ESX Agent Manager API, you can obtain status and state data about all the ESX agencies that a solution creates. The status of an ESX agency reflects the status of all the ESX agents that are running in the ESX agency.

- Integrate a Solution with ESX Agent Manager

  The EAM Sample Solution contains classes that you can use to integrate a solution with ESX Agent Manager Manager.

Benefits of Integrating Solutions with ESX Agent Manager

Integrating a solution with ESX Agent Manager has several benefits and eliminates potential problems that can occur when you deploy ESX agent virtual machines.

Monitoring Solutions

When you integrate a solution with ESX Agent Manager, users can monitor the health of agent virtual machines in the vSphere Web Client. The solution can monitor ESX agents by using the ESX Agent Manager API. Integrating a solution with ESX Agent Manager also helps you to add functions to ESXi hosts.

User Privileges

By integrating a solution with ESX Agent Manager, you can limit certain operations on ESX agent virtual machines to certain types of user. For example, you can block all users from powering off an ESX agent unless they have the **EAM.Modify** privilege.

Integration with vSphere Features

When you develop solutions that deploy virtual machines that extend the function of ESXi hosts, integrate the solution with ESX Agent Manager. If you do not integrate the solution and ESX Agent Manager, vSphere does not detect that virtual machines are ESX agent virtual machines, Virtual machines that ESX Agent Manager deploys integrate with vSphere features in ways that other virtual machines do not. For example, if an ESXi host implements vSphere High Availability (VMware HA) and that host stops running unexpectedly, normally all the virtual machines running on that host migrate to another ESXi host. An ESX agent virtual machine should not migrate to another host. Features such as vSphere Distributed

Resource Scheduler (DRS), the Add Host to Cluster operation, vSphere Distributed Power Management (DPM), and maintenance mode can also create problems if they modify or migrate ESX agents. Similarly, if one of these features migrates or modifies an ESX agent that a solution requires, that solution can stop running.

ESX Agent Manager is compatible with the VMware HA, DRS, AddHost, DPM, and maintenance mode features. By integrating a solution with ESX Agent Manager, you can deploy ESX agent virtual machines that modify the function of ESXi hosts and implement VMware HA, DRS, AddHost, DPM, and maintenance mode. ESX Agent Manager ensures that solutions that deploy ESX agents integrate correctly with these vSphere features.

ESX Agent Virtual Machines and High Availability

If you have a configuration with a very large virtual appliance per host, a best practice is to use an ESX Agent VM as a container for the appliance, because vCenter Server High Availability (HA) feature gives special priority to ESX Agent VMs.

The HA feature will power on an ESX Agent VM before any other VMs.

If a failover host policy is in use, the HA feature allows Agent VMs to be powered on with the failover hosts.

The HA feature does not include ESX Agent VMs in admission control calculations. Therefore, if you want to use a VM that is not included in the slot size for the 'host failures to tolerate' admission control policy, you can use an Agent VM.

To take advantage of these special priorities, you must include an ESX Agent VM on each host in a cluster.

Table 5-1. Interaction Between vSphere Features and ESX Agent Manager

vSphere Feature	Interaction with ESX Agent Manager
VMware HA	- If a host stops unexpectedly, VMware HA does not start an ESX agent virtual machine on another host, unless a failover policy is in use. - When a host restarts, VMware HA restarts the ESX agent virtual machines before it restarts other virtual machines. - ESX agent virtual machines are not included in slot size for admission control calculations.
Distributed Resource Scheduler (DRS)	- ESX Agent Manager pins ESX agent virtual machines to the hosts on which they are running. DRS does not move ESX agent virtual machines between hosts in a cluster. - ESX Agent Manager blocks DRS from moving virtual machines that are not ESX agents to hosts in clusters where ESX agents that the virtual machines require are not available.
Add Host to Cluster operation	When a user adds a host to a cluster of hosts that require an ESX agent, ESX Agent Manager deploys the required ESX agents on the new host.

Table 5-1. Interaction Between vSphere Features and ESX Agent Manager (Continued)

vSphere Feature	Interaction with ESX Agent Manager
vSphere Distributed Power Management (DPM)	DPM can put a host into standby mode even when ESX agent virtual machines are present. DPM powers off ESX agent virtual machines only after it has moved all the other virtual machines to another host and put the host into standby mode.
Maintenance mode	When a host enters maintenance mode, ESX Agent Manager powers off any ESX agent virtual machines and restarts the ESX agents when the host exits maintenance mode.

Overview of vSphere ESX Agent Manager Architecture

ESX Agent Manager is an intermediary between vCenter Server and a solution. ESX Agent Manager provides a WSDL based SOAP API that you can use to create solutions that deploy and monitor ESX agent virtual machines on ESXi hosts, and integrate the ESX agents with Distributed Resource Scheduler (DRS), Distributed Power Management (DPM), High Availability (HA), and other vSphere features.

For an overview of the architecture of ESX Agent Manager and how it integrates with vCenter Server, see the introduction to the *ESX Agent Manager API Reference*.

Configuration Components of ESX Agencies

For a solution to integrate with ESX Agent Manager, it must register with vCenter Server as an extension, and create an ESX agency that defines the configuration of the solution and the ESX agents that the solution deploys.

ESX agencies encapsulate the ESX agents that they deploy from a solution. When you create an ESX agency, you must provide configuration information that the solution applies to the individual ESX agents that it deploys.

You must set the scope of the ESX agents that the agency defines and the ESX agency goal state. You provide a URL to an Open Virtualization Format (OVF) file that defines the ESX agent virtual machines. You also provide a URL to an optional vSphere installation bundle (VIB) that provides a function that extends an ESXi host, for example, a VMkernel module or a custom ESXi server application.

- ESX Agency Scope

  When you create an ESX agency, you must define the scope of the agency. You define the scope of an ESX agent in terms of compute resources, namely standalone hosts or clusters.

- ESX Agency Goal State

  When you create an ESX agency, you define a goal state for that ESX agency. The goal state of an ESX agency defines the state in which all the ESX agents in that ESX agency should be when the solution is running correctly.

- ESX Agency Status

  When you create an agency, you define the goal state of the ESX agents that the agency contains. The status of an ESX agency reflects the situation of the ESX agency in relation to its goal state.

- Status of ESX Agency Scope Changes

  When a solution updates its ESX agency scope, ESX Agent Manager provisions new ESX agents on new hosts and removes ESX agents from hosts that are no longer in the scope.

ESX Agency Scope

When you create an ESX agency, you must define the scope of the agency. You define the scope of an ESX agent in terms of compute resources, namely standalone hosts or clusters.

For example, ESX agents can run on standalone ESXi hosts or on clusters of ESXi hosts.

A solution must obtain from vCenter Server the set of compute resource instances on which to deploy ESX agents. For example, you can implement a query in your solution that obtains a list of all the hosts that are running in a vCenter Server instance. Users can then select the hosts on which to deploy ESX agents.

You set the initial ESX agency scope in the ESX agency configuration. As the solution runs, the scope of the ESX agency can change. For example, the EAM Sample Solution runs on compute resources that users select from the EAM Sample Solution Configuration page. As users select different compute resources on which to run the solution, the scope of the ESX agency changes as it adds the MoRefs of the selected compute resources to the scope.

ESX Agency Goal State

When you create an ESX agency, you define a goal state for that ESX agency. The goal state of an ESX agency defines the state in which all the ESX agents in that ESX agency should be when the solution is running correctly.

If the ESX agency is not in its goal state, ESX Agent Manager attempts to remediate the issues that it discovers, so that the agency can achieve or return to its goal state.

Table 5-2. ESX Agent Goal States

Goal State	Description
Enabled	Solution has deployed all ESX agents in the ESX agency successfully, and they are running with a heartbeat. For ESX agent virtual machines without VMware Tools installed, which do not return a heartbeat, ESX Agent Manager considers them to be enabled when they are powered on.
Disabled	Solution has deployed all ESX agents in the ESX agency successfully, but they are not running.
Undeployed	Solution has removed all ESX agents in the ESX agency from all the compute resources in the ESX agent scope.

ESX Agency Status

When you create an agency, you define the goal state of the ESX agents that the agency contains. The status of an ESX agency reflects the situation of the ESX agency in relation to its goal state.

ESX Agent Manager notifies a solution of the status of its ESX agencies. ESX Agent Manager defines three possible statuses for ESX agencies.

Table 5-3. ESX Agent Status

Status	Description
Green	The ESX agency has achieved its goal state.
Yellow	ESX Agent Manager is working to move the ESX agency into its goal state.
Red	The ESX agency cannot achieve its goal state. The solution or the vSphere administrator must actively resolve one or more issues. ESX Agent Manager reports the issues to the solution in the agency runtime. ESX Agent Manager can resolve certain issues if the administrator clicks the **Resolve Issues** button in the ESX Agent Manager tab, for example if an ESX agent is powered off.

ESX Agent Manager sets the ESX agency status to yellow until the ESX agency reaches its goal state. This can either mean that the deployment of the ESX agents is still ongoing, or that ESX Agent Manager is attempting to remediate issues. You define the remediation of issues in the implementation of the solution, or you can leave the remediation of issues to the vSphere administrator, who can resolve the issue by using the ESX Agent Manager user interface.

The criteria on which ESX Agent Manager determines that the status of an ESX agency is green depends on the goal state of that ESX agency.

Table 5-4. Criteria for Meeting the Goal State of an ESX Agency

ESX Agency Goal State	Criteria for Green Status
Enabled	ESX Agent Manager has downloaded the ESX agent binaries from the OVF and VIB URLs, deployed the ESX agent virtual machines, installed the VIB, started the ESX agent virtual machines, and received heartbeats from all of them.
Disabled	ESX Agent Manager has downloaded the ESX agent binaries from the OVF and VIB URLs, deployed the ESX agent virtual machines, and installed the VIB.
Undeployed	ESX Agent Manager has powered off and deleted all the ESX agent virtual machines and uninstalled the VIB.

If the ESX agency cannot reach its goal state, ESX Agent Manager reports the issues to the solution and sets the status to red.

Note   If the scope of an ESX agency is empty, there are no compute resources onto which to deploy ESX agents, so no ESX agents are deployed. In this case, ESX Agent Manager determines that the ESX agency has performed correctly, and sets the status to green.

Status of ESX Agency Scope Changes

When a solution updates its ESX agency scope, ESX Agent Manager provisions new ESX agents on new hosts and removes ESX agents from hosts that are no longer in the scope.

When the scope of an ESX agency changes, vSphere ESX Agent Manager detects the change and displays the status of the ESX agents in the scope and operations that are in progress. You can add functions in the solution definition to remediate the issues, or the solution can report the issues so that administrators can fix them manually.

When a solution adds or removes compute resources from the ESX agency scope, the status of the ESX agency changes as the solution works to achieve the goal state of the ESX agency.

Table 5-5. Status of ESX Agency Scope Changes

Scope Change	Current ESX Agency State	ESX Agency Status
Add compute resource.	Undeployed	Unchanged. New ESX agents always start in the undeployed state.
Remove compute resource.	Undeployed	Unchanged. Old ESX agents that the solution has not removed yet are pending undeployment.
Add compute resource.	Disabled	Yellow, until the solution deploys the ESX agents on the new compute resource.
Remove compute resource.	Disabled	Yellow, until the solution undeploys the ESX agents from the compute resource.
Add compute resource.	Enabled	Yellow, until the solution deploys the ESX agents on the new compute resource, starts the ESX agents, and receives heartbeats.
Remove compute resource.	Enabled	Yellow, until the solution stops and undeploys the ESX agents from the compute resource.

Requirements for Integrating a Solution with ESX Agent Manager

To integrate with ESX Agent Manager, a solution must meet certain requirements.

To integrate a solution with ESX Agent Manager, you must develop the solution according to the following requirements:

- The solution must be a vCenter Server extension that implements the `Extension` data object and registers with `ExtensionManager`.

- Use Open Virtualization Format (OVF) to package ESX agent virtual machines or vApps. ESX Agent Manager only supports the deployment of virtual machines using OVF.

- Use HTTP or HTTPS to publish OVF files to ESX Agent Manager.

- Use vSphere installation bundles (VIB) to add functions to ESXi hosts, for example to add VMkernel modules or custom ESX Server applications to ESXi hosts.

- Use HTTP or HTTPS to publish VIB files to ESX Agent Manager.

- Use vCenter Server Compute Resources to define the ESX agent scope.

Authenticating Against ESX Agent Manager

vCenter Server handles the authentication of ESX Agent Manager clients, so a solution must first log in to vCenter Server before it can call the methods of the ESX Agent Manager API.

ESX Agent Manager registers itself with the vCenter Server reverse proxy in the `eam` namespace. You find the ESX Agent Manager API under `eam/sdk`. You must direct all ESX Agent Manager API calls in your solution to *vcenter_server_ip_address*/eam/sdk.

ESX Agent Manager grants access to clients that are also clients of the vCenter Server instance in which ESX Agent Manager is an extension. All vCenter Server extensions that have an active vCenter Server session have access to ESX Agent Manager. These extensions can create ESX agencies and monitor existing ESX agencies in ESX Agent Manager. The user name that the extension provides is the extension key that the extension sets in the vCenter Server `ExtensionManager` instance.

Users can access the ESX Agent Manager user interface only if they have the **Eam.View** and **Eam.Modify** privileges. Solutions by definition have all vCenter Server privileges, but the **Eam.View** and **Eam.Modify** privileges limit what users can do in the vSphere Web Client.

Table 5-6. ESX Agent Manager Privileges

Privilege	Permitted Actions
Eam.View	Users can monitor all running ESX agencies in ESX Agent Manager.
Eam.Modify	Users can modify the ESX agencies, for example by powering off ESX agent virtual machines.

When you make an HTTP request to call an object in the ESX Agent Manager API for the first time, you must set the VMware SOAP session cookie. You set it to the value of the VMware SOAP session cookie of the vCenter Server HTTP connection. You must add a function in your solution to obtain the SOAP session cookie from vCenter Server when the solution establishes the connection to vCenter Serverr. The cookie remains set for the duration of the ESX Agent Manager session.

Availability of ESX Agent Virtual Machines

You can configure ESX agencies so that they only mark ESX agent virtual machines as available after the client of the agency has performed additional configuration after provisioning or powering on the ESX agent virtual machine.

You configure an ESX agency in the `Agency.ConfigInfo` object. You can set two options in `Agency.ConfigInfo` that set when ESX agents can be marked as available.

Table 5-7. ESX Agent Availability Options

Option	Description
Boolean `manuallyMarkAgentVmAvailableAfterProvisioning`	If set to true, the client of this ESX agency must manually mark the agent as ready after the ESX agent virtual machine has been provisioned. This is useful if the client of this solution performs some extra reconfiguration of the ESX agent virtual machine before it is powered on.
Boolean `manuallyMarkAgentVmAvailableAfterPowerOn`	If set to true, the client of this ESX agency must manually mark the agent as ready after the ESX agent virtual machine has been powered on. In this case, DRS will not regard the ESX agent virtual machine as ready until the client has marked the agent as ready.

Monitoring and Resolving ESX Agent Issues

With the ESX Agent Manager API, you can obtain status and state data about all the ESX agencies that a solution creates. The status of an ESX agency reflects the status of all the ESX agents that are running in the ESX agency.

Solutions can only access data about ESX agencies that they create. A solution cannot access the data of ESX agencies that other solutions create on the same host.

You can use ESX Agent Manager to monitor and track many of the issues that can occur when you deploy ESX agent virtual machines. ESX Agent Manager handles updates to the deployment specifications of ESX agent virtual machines. ESX Agent Manager responds to changes in the vCenter Server instance in which it runs. If ESX Agent Manager encounters a conflict and cannot satisfy the requirements of an update to a deployment specification, it reports the conflict to the solution as an issue and shows the issue in the ESX Agent Manager user interface.

The issues that the ESX Agent Manager monitors relate to the goal state and status of ESX agencies and agents. If ESX Agent Manager encounters an issue, it sets the status of the ESX agency or agent to red and alerts the solution about the issue. ESX Agent Manager can resolve certain issues if the administrator clicks the **Resolve Issues** button in the ESX Agent Manager user interface.

Remediation of Issues by ESX Agent Manager

ESX Agent Manager reports to the solution or to the vSphere administrator issues that can occur during the deployment of ESX agent virtual machines. ESX Agent Manager can resolve certain issues, if the solution or the administrator requests that it does so.

ESX Agent Manager defines standard issues that can occur during the deployment of ESX agent virtual machines. For example, deployment of ESX agents can fail because the network, datastore, or resource pool settings are incorrect.

Certain errors occur only in the certain goal states. Other errors can occur in any ESX agency goal state.

Table 5-8. Remediation of Issues by ESX Agent Manager (Partial List)

Issue	Goal State	Resolved by ESX Agent Manager?
AgentVmPoweredOffIssue	Enabled	Yes
AgentVmWrongDatastoreIssue	Any	No
AgentVmWrongNetworkIssue	Any	No
AgentVmWrongResourcePoolIssue	Any	Yes
AgentVmWrongFolderIssue	Any	Yes
InsufficientSpaceOnDatastoreIssue	Enabled	No

Integrate a Solution with ESX Agent Manager

The EAM Sample Solution contains classes that you can use to integrate a solution with ESX Agent Manager Manager.

For descriptions of the objects that the EAM Sample Solution implements, see the *vSphere ESX Agent Manager API Reference*.

Procedure

1   Connect a Solution to ESX Agent Manager

    To connect a solution to ESX Agent Manager, you must obtain connection information from the vCenter Server in which you run the solution. You pass a reference to the `EsxAgentManager` instance that is running in vCenter Server to the solution.

2   Configure an ESX Agency and ESX Agents

    You must define the configuration of the ESX agencies in the implementation of the solution. You set the configuration for ESX agencies and ESX agents in the `AgencyConfigInfo` and `AgentConfigInfo` data objects.

3   Create an ESX Agency

    You create an ESX agency by calling the `EsxAgentManager.createAgency()` method. You must specify ESX agent configurations for each version of ESX Server on which you deploy ESX agents.

4   Update the Agency Scope of a Solution

    You define the ESX agency scope of a solution by passing the managed object references (MoRefs) of the vSphere compute resources to the solution.

5   Change the Goal State of an Agency

    A solution can change the goal state of its ESX agencies while the solution is running. For example, when a solution starts, the goal state of its ESX agencies can be `ENABLED`. If the solution includes a function to remove ESX agencies, when this function runs, the goal state of the ESX agency changes to `UNDEPLOYED`.

6   Delete an ESX Agency

    A solution can delete an ESX agency by calling the `Agency.destroyAgency()` method on the `Agency` object.

7 Resolve ESX Agent Issues

ESX Agent Manager can detect issues in the ESX agents that solutions deploy. Solutions can try to resolve issues when the status of an ESX agency or an ESX agent is set to red.

Connect a Solution to ESX Agent Manager

To connect a solution to ESX Agent Manager, you must obtain connection information from the vCenter Server in which you run the solution. You pass a reference to the `EsxAgentManager` instance that is running in vCenter Server to the solution.

You pass the connection information to a solution by calling the appropriate methods from the vSphere API.

Note If you integrate your solution with the vCenter Extension vService, the vService obtains the vCenter Server connection information for you. See Chapter 6 Integrating an Extension with the vCenter Extension vService.

The EAM Sample Solution defines the connection to vCenter Server in the `VimConnection.java` class and the connection to ESX Agent Manager in the `EamConnection.java` class.

The `VimConnection` class defines methods to obtain the vCenter Server host ID, HTTPS proxy, and the session cookie. The `Manager.java` class implements `VimConnection` to connect to the vCenter Server on which the EAM Sample Solution is running. See Connect the Extension to vCenter Server.

Prerequisites

- Download the vSphere ESX Agent Manager SDK.

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Open *eam_work_folder*\src\com\vmware\eam\sample\solution\util\VimConnection.java in an editor.

- Open *eam_work_folder*\src\com\vmware\eam\sample\solution\util\EamConnection.java in an editor.

- Open *eam_work_folder*\src\com\vmware\eam\sample\solution\Manager.java in an editor.

- Obtain the following information from the vCenter Server instance on which you run a solution:

  - The vCenter Server host ID.

  - The HTTPS proxy port for vCenter Server.

  - The session cookie for the current user session in which the solution is logged in as an extension.

  - The managed object reference (MoRef) of the ESX Agent Manager instance that is running in the vCenter Server instance.

Procedure

1 Establish a connection to the vCenter Server instance on which the solution runs.

The EAM Sample Solution establishes the connection to vCenter Server in the `VimConnection.java` class.

2 Create an instance of the `ManagedObjectReference` object of type `EsxAgentManager`.

The `EamConnection.java` class defines a constructor that creates a MoRef named `_eamRef` and a method named `getEsxAgentManager()` that returns the MoRef to the solution.

```
public EamConnection(String vcHost, int vcHttpsProxyPort, String sessionCookie) {
 [...]

 _eamRef = new ManagedObjectReference();
 _eamRef.setType("EsxAgentManager");
 _eamRef.setValue("EsxAgentManager");
}

 [...]

 public ManagedObjectReference getEsxAgentManager() {
    return _eamRef;
}
```

3 Connect to the ESX Agent Manager with the URL to the ESX Agent Manager service and the session cookie for the current session in which the solution is logged as an extension.

The `EamConnection` class defines a `connect()` method that connects to the ESX Agent Manager service.

```
public void connect() {
  if (_isConnected) {
    return;
  }

  try {
    [...]

    String eamUrl = "https://" + _vcHost + ":" + _vcHttpsProxyPort + "/eam/sdk/";
    [...]
    Map<String, List<String>> map = new HashMap<String, List<String>>();
    map.put("Cookie", Collections.singletonList(_sessionCookie));
    ((BindingProvider) _stub).getRequestContext()
            .put(MessageContext.HTTP_REQUEST_HEADERS, map);
    _isConnected = true;
  }
}
```

You obtained the connection details for vCenter Server and connected a solution to ESX Agent Manager.

What to do next

Configure ESX agencies and agents to deploy from the solution.

Configure an ESX Agency and ESX Agents

You must define the configuration of the ESX agencies in the implementation of the solution. You set the configuration for ESX agencies and ESX agents in the `AgencyConfigInfo` and `AgentConfigInfo` data objects.

When you create ESX agencies, you provide the `AgencyConfigInfo` object with an array of `AgentConfigInfo` objects for each version of ESX Server on which the agency deploys ESX agents. You also define the name of the ESX agency and of the ESX agents and the scope of the ESX agency in the `AgencyConfigInfo` object.

You define the deployment of the ESX agent virtual machines in the `AgentConfigInfo` object. You set the following information in the `AgentConfigInfo` object.

- A URL to the Open Virtualization Format (OVF) file from which to deploy the ESX agent.

- A URL to an optional vSphere Installation Bundle (VIB) that adds function to the ESX Server, for example a VMkernel module or a custom ESX application that you developed.

The URL to the ESX agent virtual machine OVF and the URL to an optional VIB must lead to a server that ESX Agent Manager can access. ESX Agent Manager downloads the ESX agent virtual machine from the URLs that you provide and deploys the virtual machines on the ESX hosts. ESX Agent Manager installs one ESX agent instance per agency per host.

Note   To install VIBS, all ESXi hosts must have configured the firewall so that they can access the HTTP port on the vCenter Server.

Setting the `ovfEnvironment` property allows a solution to provide OVF properties specific to the ESX agent virtual machine. ESX Agent Manager sets the OVF properties when it deploys an ESX agent. A typical use of the `ovfEnvironment` field is to specify the IP address and credentials of the solution so that ESX agents can connect back to the solution when they are running.

Prerequisites

- Download the vSphere ESX Agent Manager SDK.

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Open *eam_work_folder*\src\com\vmware\eam\sample\solution\AgentHandler.java in a text editor.

Procedure

1   Create a program to configure and create ESX agencies and agents.

The EAM Sample Solution defines the configuration and creation of ESX agencies and agents in the
`AgentHandler.java` class.

```
public AgentHandler(String selfUrl,
                    String selfIp,
                    String ovfUrl4x,
                    String ovfUrl50,
                    String vibUrl4x,
                    String vibUrl50,
                    boolean deployVibs,
                    Map<String, String> ovfEnvironment,
                    VcUtils vcUtils) {
    _vcUtils = vcUtils;

    _agentConfigInfo4x = new AgentConfigInfo();
    _agentConfigInfo50 = new AgentConfigInfo();

    [...]
}
```

2   Create `AgentConfigInfo` instances for each type of ESX agent that the ESX agency deploys.

The EAM Sample Solution defines ESX agents for ESX Server 4.x and for ESX Server 5.0.

```
public AgentHandler([...]) {
    [...]
    _agentConfigInfo4x = new AgentConfigInfo();
    _agentConfigInfo50 = new AgentConfigInfo();
    [...]
}
```

3   Set the URLs to the OVF files from which the solution deploys ESX agent virtual machines by calling
the `AgentConfigInfo.setOvfPackageUrl()` method.

The EAM Sample Solution constructs the URLs to OVF files from information that you set in the
`eamri.properties` file.

```
public AgentHandler([...]) {
    [...]

    _agentConfigInfo4x.setOvfPackageUrl(urlPrefix + ovfUrl4x);
    _agentConfigInfo50.setOvfPackageUrl(urlPrefix + ovfUrl50);

    [...]
    }
```

4   (Optional) Set the URLs to the optional VIB files from which the solution adds functions to the ESX
     Server by calling the `AgentConfigInfo.setVibUrl()` method.

     The EAM Sample Solution constructs the URLs to VIB files from information that you set in the
     `eamri.properties` file.

```
public AgentHandler([...]) {
   [...]

   _agentConfigInfo4x.setOvfPackageUrl(urlPrefix + ovfUrl4x);
   _agentConfigInfo50.setOvfPackageUrl(urlPrefix + ovfUrl50);

   if (deployVibs) {
      _agentConfigInfo4x.setVibUrl(urlPrefix + vibUrl4x);
      _agentConfigInfo50.setVibUrl(urlPrefix + vibUrl50);

      [...]
   }
}
```

5   Set any OVF environment properties that the solution requires by creating an instance of the
     `AgentOvfEnvironmentInfo` object and passing it to the `AgentConfigInfo` object for each ESX
     agent.

     The EAM Sample Solution sets some dummy properties in the `eamri-webapp.xml` file.

```
public AgentHandler([...]) {
   [...]
   AgentOvfEnvironmentInfo ovfEnv = new AgentOvfEnvironmentInfo();
   for (final Map.Entry<String, String> entry : ovfEnvironment.entrySet()) {
      ovfEnv.getOvfProperty().add(new AgentOvfEnvironmentInfoOvfProperty() {
         {
            setKey(entry.getKey());
            setValue(entry.getValue());
         }
      });
   }
   _agentConfigInfo4x.setOvfEnvironment(ovfEnv);
   _agentConfigInfo50.setOvfEnvironment(ovfEnv);
   [...]
}
```

6   Create an instance of `AgencyConfigInfo` to define the ESX agency that the solution deploys.

```
public AgentHandler([...]) {
   [...]

   _agencyConfigInfo = new AgencyConfigInfo();
   [...]
}
```

7 Provide names for the ESX agency and the ESX agents by calling the `AgencyConfigInfo.setAgencyName()` and `setAgentName()` methods.

The EAM Sample Solution names the ESX agency and the ESX agents Sample Service.

```
public AgentHandler([...]) {
    [...]

    _agencyConfigInfo = new AgencyConfigInfo();
    _agencyConfigInfo.setAgencyName("Sample Service");
    _agencyConfigInfo.setAgentName("Sample Service");
    [...]
}
```

8 Add an array of ESX agent configurations to the ESX agency configuration by calling the `AgencyConfigInfo.getAgentConfig()` method.

```
public AgentHandler([...]) {
    [...]

    _agencyConfigInfo = new AgencyConfigInfo();
    _agencyConfigInfo.setAgencyName("Sample Service");
    _agencyConfigInfo.setAgentName("Sample Service");
    _agencyConfigInfo.getAgentConfig().add(_agentConfigInfo4x);
    _agencyConfigInfo.getAgentConfig().add(_agentConfigInfo50);
    [...]
}
```

9 Set the scope of the ESX Agency by calling the `AgencyConfigInfo.setScope()` method.

Users of the EAM Sample Solution set the scope of the ESX agency by selecting ESXi hosts from the EAM Sample Solution Configuration page. Consequently, the scope is empty until the EAM Sample Solution updates it according to the user interaction.

```
public AgentHandler([...]) {
    [...]
    _agencyConfigInfo.setScope(null);
    [...]
}
```

You set the configuration properties for an ESX agency and the ESX agents that it contains.

What to do next

Call the `createAgency()` method to create the ESX agency.

Create an ESX Agency

You create an ESX agency by calling the `EsxAgentManager.createAgency()` method. You must specify ESX agent configurations for each version of ESX Server on which you deploy ESX agents.

When you call the `EsxAgentManager.createAgency()` method you must pass it an `AgencyConfigInfo` object and a string to define the initial goal state of the ESX agents that the agency deploys. The `initialGoalState` property informs ESX Agent Manager of the state in which to deploy ESX agent virtual machines when the solution first runs.

The EAM Sample Solution defines a function that creates an ESX agency in the `AgentHandler.java` class.

Prerequisites

- [Download the vSphere ESX Agent Manager SDK](#).

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have opened
  eam_work_folder\src\com\vmware\eam\sample\solution\AgentHandler.java in an editor.

Procedure

1 Establish a connection to the ESX Agent Manager running in vCenter Server.

   The `AgentHandler.java` implements the `EamConnection.java` class from the EAM Sample Solution to connect to vCenter Server and ESX Agent Manager.

   ```
   public void setup(EamConnection eamConnection) {
     assert eamConnection != null;
     _eamConnection = eamConnection;

     ManagedObjectReference eamRef = _eamConnection.getEsxAgentManager();
     EamPortType stub = _eamConnection.getStub();

     [...]
   }
   ```

2 Create an ESX agency by calling the `EsxAgentManager.createAgency()` method.

   The `AgentHandler.java` class checks whether any ESX agencies are already running, and if not calls the `EsxAgentManager.createAgency()` method. `AgentHandler.java` passes to the `EsxAgentManager.createAgency()` method the `ManagedObjectReference` object for the ESX Agent Manager instance running in vCenter Server, `eamRef`. `AgentHandler.java` also passes to `createAgency()` the `AgencyConfigInfo` object that defines the configuration of the ESX agency.. The `AgentHandler.java` class sets the initial goal state of the ESX agency to `ENABLED`.

   ```
   public void setup(EamConnection eamConnection) {
     assert eamConnection != null;
     _eamConnection = eamConnection;

     ManagedObjectReference eamRef = _eamConnection.getEsxAgentManager();
     EamPortType stub = _eamConnection.getStub();
     try {
       List<ManagedObjectReference> agencyRefs = stub.queryAgency(eamRef);
       if (agencyRefs != null && agencyRefs.size() > 0) {
             _agency = agencyRefs.get(0);
   ```

```
    } else {
      _agency = stub.createAgency(eamRef,
                                  _agencyConfigInfo,
                                  EamObjectRuntimeInfoGoalState.ENABLED.toString()
                                  .toLowerCase());
    }
  [...]
  }
```

3   Call the `Agency.queryConfig()` method to verify the configuration of the ESX agency and report any issues with the configuration.

```
  public void setup(EamConnection eamConnection) {
      [...]
      _agencyConfigInfo = stub.queryConfig(_agency);
      _isSetup = true;
    } catch (RuntimeFaultFaultMsg e) {
      _log.error(e, e);
    } catch (InvalidAgencyScopeFaultMsg e) {
      _log.error(e, e);
    } catch (InvalidAgentConfigurationFaultMsg e) {
      _log.error(e, e);
    } catch (InvalidUrlFaultMsg e) {
      _log.error(e, e);
    }
  }
```

4   Call the `Agency.agencyQueryRuntime()` method to return the status of the ESX agency.

The `Agency.agencyQueryRuntime()` method returns an `EamObjectRuntimeInfo` object that contains the goal state of the agency, its current status, and a list of any problems that the agency has encountered.

```
  public EamObjectRuntimeInfo getRuntime() throws RuntimeFaultFaultMsg {
      waitForSetup();
      return _eamConnection.getStub().agencyQueryRuntime(_agency);
  }
```

You created an ESX agency that a solution can deploy on ESXi hosts.

Update the Agency Scope of a Solution

You define the ESX agency scope of a solution by passing the managed object references (MoRefs) of the vSphere compute resources to the solution.

You set the initial ESX agency scope in the `scope` property of the `AgencyConfigInfo` object. You can change the scope when a solution runs by calling the `Agency.update()` method. For example, in the EAM Sample Solution, users select the ESXi hosts on which to run the solution from a list on the EAM Sample Solution Configuration page. The EAM Sample Solution updates the scope of the sample ESX agency according to the hosts that the user selects.

The EAM Sample Solution defines a function to update the scope of the ESX agency in the
`AgentHandler.java` class.

Prerequisites

- [Download the vSphere ESX Agent Manager SDK](#).

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have opened
  eam_work_folder\src\com\vmware\eam\sample\solution\AgentHandler.java in an editor.

- Open *eam_work_folder*\src\com\vmware\eam\sample\solution\utils\VcUtilsjava in an
  editor.

Procedure

1 Write a function that implements the vSphere Web Services API to detect compute resources on
   which to run the solution.

   The EAM Sample Solution provides a helper class, `VcUtils.java`, that defines functions to obtain
   the compute resources on which to run the solution. `AgentHandler.java` calls the
   `VcUtils.getComputeResources()` method to obtain a list of `ManagedObjectReference` objects for
   the ESXi hosts running in vCenter Server.

   ```
   public void updateConfig(String[] updates) throws RuntimeFaultFaultMsg {
       waitForSetup();

       boolean changed = false;
       Map<String, ManagedObjectReference> crs = _vcUtils.getComputeResources();
   ```

2 Add the `ManagedObjectReference` objects for the compute resources to a `HashSet` that defines the
   ESX agency scope.

   The `AgentHandler.java` class adds the list of `ManagedObjectReference` objects that the
   `VcUtils.getComputeResources()` method returns to the existing scope and updates the list if
   additional compute resources are present.

   ```
       Set<ManagedObjectReference> newScope = new HashSet<ManagedObjectReference>();

       for (String update : updates) {
           String[] kv = update.split("=", 2);
           if (kv[0].equals("scope")) {
             try {
                ManagedObjectReference cr = crs.get(kv[1]);
                if (cr == null) {
                  continue;
                }
                ManagedObjectReference moRef = new ManagedObjectReference();
                moRef.setType(cr.getType());
                moRef.setValue(cr.getValue());
                newScope.add(moRef);
             }
   ```

```
        catch (NullPointerException e) {
            // ignore
        }
      }
    }
  }
```

3   Create an `AgencyComputeResourceScope` instance to contain the scope `HashSet`.

```
    AgencyComputeResourceScope scopeDO = (AgencyComputeResourceScope) _agencyConfigInfo.getScope();
    Set<ManagedObjectReference> oldScope = new
  HashSet<ManagedObjectReference>(scopeDO.getComputeResource());
```

4   Compare the old scope to the new scope to establish whether any compute resources have been added or removed.

The `AgentHandler.java` class compares the size of the new scope to the initial scope and adds any new compute resources to the `HashSet` of `ManagedObjectReference` objects.

```
    if (!oldScope.containsAll(newScope) || oldScope.size() != newScope.size()) {
        AgencyComputeResourceScope scope = new AgencyComputeResourceScope();
        scope.getComputeResource().addAll(newScope);
        agencyConfigInfo.setScope(scope);
        changed = true;
    }
```

5   If the new scope differs from the old scope, call `Agency.update()` to add the new scope to the ESX agency.

```
    if (changed) {
      assert _agency != null;
      try {
          _eamConnection.getStub().update(_agency, agencyConfigInfo);
      } catch (Exception e) {
       _log.error("Failed to update agency. Reason: " + e.getMessage());
      }
      updateConfiguration();
    }
  }
```

You defined a function in a solution to detect changes of scope and to update an ESX agency.

Change the Goal State of an Agency

A solution can change the goal state of its ESX agencies while the solution is running. For example, when a solution starts, the goal state of its ESX agencies can be `ENABLED`. If the solution includes a function to remove ESX agencies, when this function runs, the goal state of the ESX agency changes to `UNDEPLOYED`.

You call the `Agency.enable()`, `Agency.disable()`, and `Agency.uninstall()` methods to change the goal state of an ESX agency. Calling these methods changes the status of the ESX agency to yellow until the agency reaches the desired state, in which case the status changes to green. If the ESX agency cannot achieve the goal state, the status changes to red.

The EAM Sample Solution defines a function to change the goal state of its ESX agencies in the `AgentHandler.java` class. The EAM Sample Solution changes the goal state of its ESX agency when users select ESXi hosts on which to run the solution, and when they uninstall the solution.

Prerequisites

- [Download the vSphere ESX Agent Manager SDK](#).

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have opened
  eam_work_folder\src\com\vmware\eam\sample\solution\AgentHandler.java in an editor.

Procedure

1 Get the current goal state of the ESX agency by calling the
   `EamObjectRuntimeInfo.getGoalState()` method.

   The EAM Sample Solution defines a function in the `AgentHandler.java` class to obtain the current goal state from the solution.

   ```
   public void updateGoalState(String params) throws RuntimeFaultFaultMsg,
     NotFoundFaultMsg {
     String[] kv = params.split("=", 2);
     assert kv[0].equals("goalState");

     String goalState = kv[1];
     String currentGoalState = getRuntime().getGoalState().toString();
   ```

2 Call the `Agency.enable()`, `Agency.disable()`, and `Agency.uninstall()` methods to set the ESX agency in the new goal state.

   The EAM Sample Solution calls the appropriate methods to set the ESX agencies to the goal state. If the goal state is UNINSTALLED, the solution calls the `cleanup()` method that `Manager.java` defines to remove the ESX agencies and uninstall the solution.

   ```
   if (goalState.equals(currentGoalState)) {
       return;
   }

   if (goalState.equals(EamObjectRuntimeInfoGoalState.ENABLED.toString()
                                               .toLowerCase())) {
       enable();
   } else if (goalState.equals(EamObjectRuntimeInfoGoalState.DISABLED.toString()
                                               .toLowerCase())) {
       disable();
   } else {
       assert goalState.equals(EamObjectRuntimeInfoGoalState.UNINSTALLED.toString()
   ```

```
                                                              .toLowerCase());
        _unregistered = true;
        Manager.getInstance().cleanup();
  }
 }
```

You changed the goal state of an ESX agency while the solution is running.

Delete an ESX Agency

A solution can delete an ESX agency by calling the `Agency.destroyAgency()` method on the `Agency` object.

Typically, before deleting an ESX agency, a solution firsts call the `EsxAgentManager.uninstall()` method to put the agency in the uninstalled state. The solution tracks the progress of `EsxAgentManager.uninstall()` and only calls `destroyAgency()` to remove the ESX agency when the status of the ESX agency is green.

If your solution does not need to track the removal of the ESX agency and its ESX agents, you can call `destroyAgency()` without first calling `uninstall()`. ESX Agent Manager removes the ESX agency and all of the ESX agents without tracking the status of the uninstallation process.

Alternatively, disconnecting the solution from vCenter Server by calling the `ExtensionManager.unregisterExtension()` method removes all ESX agencies and ESX agents.

The EAM Sample Solution does not call the `Agency.destroyAgency()` method directly. The EAM Sample Solution calls the `Agency.disable()` and `Agency.uninstall()` methods to set the ESX agents in the uninstalled state, then calls `ExtensionManager.unregisterExtension()` to unregister the solution from vCenter Server, which removes the ESX agency and the ESX agents.

Prerequisites

- Download the vSphere ESX Agent Manager SDK.

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have opened
  eam_work_folder\src\com\vmware\eam\sample\solution\AgentHandler.java in an editor.

- Verify that you have opened
  eam_work_folder\src\com\vmware\eam\sample\solution\Manager.java in an editor.

Procedure

1 Call the `Agency.disable()` method to disable the ESX agency.

Calling the `Agency.disable()` method powers off the ESX agent virtual machines, but does not undeploy them.

The EAM Sample Solution calls the `Agency.disable()` method in the `AgentHandler.java` class.

```
public void disable() throws RuntimeFaultFaultMsg {
    waitForSetup();
        _eamConnection.getStub().disable(_agency);
}
```

2 Call the `Agency.uninstall()` method to put the ESX agency in the uninstalled state.

Calling the `Agency.uninstall()` method uninstalls all the ESX agents in the ESX agency.

The EAM Sample Solution calls the `Agency.uninstall()` method in the `AgentHandler.java` class.

```
public void uninstall() throws RuntimeFaultFaultMsg {
    waitForSetup();
        _eamConnection.getStub().uninstall(_agency);
}
```

3 Delete the ESX agency and ESX agents.

Option	Description
Call the `Agency.destroyAgency()` method on the `Agency` object.	Deletes the agency and its ESX agents, but the solution remains registered with `ExtensionManager`.
Call the `ExtensionManager.unregisterExtension()` method on the `Extension` object.	Unregisters the solution from vCenter Server, which uninstalls the solution and deletes the ESX agency and its ESX agents.

The EAM Sample Solution deletes the ESX agency by calling the `ExtensionManager.unregisterExtension()` method in the `Manager.java` class.

```
public void cleanup() throws NotFoundFaultMsg, RuntimeFaultFaultMsg {
    if (_eamConnection != null) {
        _eamConnection.disconnect();
    }

     _vimConnection.getStub().unregisterExtension(_vimConnection.getExtensionManager(),
                                        EXTENSION_KEY);
}
```

You defined a function to delete an ESX agency from a solution.

Resolve ESX Agent Issues

ESX Agent Manager can detect issues in the ESX agents that solutions deploy. Solutions can try to resolve issues when the status of an ESX agency or an ESX agent is set to red.

You obtain the issues that affect an ESX agency or agent by using the `EventManager` to listen for issues.

A solution can try to resolve issues by calling the `EamObject.resolve(Issue[])` method on an individual ESX agent, in which case ESX Agent Manager tries to resolve the issues. The solution can also call `EamObject.resolveAll()` on an ESX agency, in which case ESX Agent Manager attempts to resolve all the issues on all the ESX agents that the ESX agency deploys.

Prerequisites

- [Download the vSphere ESX Agent Manager SDK](#).

- Verify that you have set up and started the EAM Sample Solution in an application server.

- Verify that you have opened
  eam_work_folder\src\com\vmware\eam\sample\solution\AgentHandler.java in an editor.

Procedure

1   Obtain the unique identifiers of issues by calling the `Issue.getKey()` method.

    The EAM Sample Solution defines a method that calls the `Issue.getKey()` method in the
    `AgentHandler.java` class.

    ```
    public String getIssueId(Issue issue) {
        return Integer.toString(issue.getKey());
    }
    ```

2   Call the `Agent.getRuntime()` method to obtain an `EamObjectRuntimeInfo` object for a running
    ESX agent.

    The EAM Sample Solution implements `Agency.getRuntime()` in a method that checks for a running
    ESX agency and obtains the runtime information for that ESX agency.

    ```
    public Issue getIssue(String issueId) throws RuntimeFaultFaultMsg {
        waitForSetup();
        EamObjectRuntimeInfo runtime = getRuntime();
        assert runtime != null;
        [...]
    }
    ```

3   Call the `EamObject.queryIssue()` method to obtain the list of issues affecting an ESX agency from
    the runtime of that ESX agency.

    The EAM Sample Solution adds any issues that it discovers for the ESX agency to a `List` object, and
    returns the issues with their issue identifiers.

    ```
    public Issue getIssue(String issueId) throws RuntimeFaultFaultMsg {
        waitForSetup();
        EamObjectRuntimeInfo runtime = getRuntime();
        assert runtime != null;
        List<Issue> issues = _eamConnection.getStub().queryIssue(_agency, null);
        if (issues == null) {
            return null;
    ```

```
    }
    for (Issue issue : issues) {
       if (getIssueId(issue).equals(issueId)) {
          return issue;
       }
    }
    return null;
}
```

4   Call the `EamObject.resolveAll()` method to resolve all issues with ESX agents running in an ESX agency.

The EAM Sample Solution calls the `resolveAll()` method on the `Agency` object that the `AgentHandler.java` class creates.

```
public void resolveAll() throws RuntimeFaultFaultMsg {
    waitForSetup();
    _eamConnection.getStub().resolveAll(_agency);
}
```

5   Call the `EamObject.resolve()` method to resolve a specific issue with an ESX agent.

The EAM Sample Solution calls the `resolve()` method on the issue identifier that the `getIssue()` method returns, and generates a list of unknown issues that the solution cannot resolve.

```
public void resolve(String issueId) throws NumberFormatException,
    RuntimeFaultFaultMsg {
    waitForSetup();
    List<Integer> unknownIssueIds = _eamConnection.getStub()
                                          .resolve(_agency,

Collections.singletonList(Integer.parseInt(issueId)));
    if (unknownIssueIds != null) {
       _log.error("Failed to resolve issue:" + issueId);
    }
}
```

You called the methods from the ESX Agent Manager API to obtain issues and attempt to resolve them.

Integrating an Extension with the vCenter Extension vService

6

The vCenter Extension vService simplifies the installation and deployment of extensions. By integrating an extension with the vCenter Extension vService you can deploy extensions from the vSphere Web Client without having to enter the connection parameters of the vCenter Server on which you install the extension, or provide the login credentials for that vCenter Server instance.

The vCenter Extension vService performs the following functions for solutions that you integrate with it.

- Provides the extension with the connection parameters of the vCenter Server instance.

- Registers the extension certificate with vCenter Server.

For example, to deploy the EAM Sample Solution you provide the IP address and login credentials in the `eamri.properties` file, which requires you to update the solution for every vCenter Server instance on which you install it. By integrating an extension with the vCenter Extension vService, you avoid this manual step.

- Integrate a Virtual Machine with the vCenter Extension vService

  To integrate a virtual machine with the vCenter Extension vService, you must deliver the extension in an Open Virtualization Format (OVF) package, add sections to the OVF descriptor file, and provide in the guest operating system a script that connects to the vCenter Extension vService.

- Deploy an Extension in the vSphere Web Client By Using the vCenter Extension vService

  If you configure the virtual machine that runs an extension to use the vCenter Extension vService, you can deploy the extension directly in the vSphere Web Client.

- vCenter Extension vService XML Schema

  The vCenter Extension vService XML Schema defines the `<vServiceEnvironmentSection>` in the OVF environment of the virtual machine that contains the extension. It also defines the `RegisterExtension` function that registers the extension with vCenter Server.

Integrate a Virtual Machine with the vCenter Extension vService

To integrate a virtual machine with the vCenter Extension vService, you must deliver the extension in an Open Virtualization Format (OVF) package, add sections to the OVF descriptor file, and provide in the guest operating system a script that connects to the vCenter Extension vService.

Procedure

1 Configure the OVF Descriptor File

   To integrate an extension with the vCenter Extension vService, you must add a section to the Open
   Virtualization Format (OVF) descriptor file that contains the virtual machine.

2 Provide a Script in the Extension Virtual Machine to Register as a vCenter Extension

   You must write a script that you run in the virtual machine that contains the extension.

Configure the OVF Descriptor File

To integrate an extension with the vCenter Extension vService, you must add a section to the Open
Virtualization Format (OVF) descriptor file that contains the virtual machine.

Including a `<vServiceDependencySection>` element in the OVF descriptor file of the virtual machine
informs vCenter Server that this virtual machine depends on the vCenter Extension vService.

When you deploy a virtual machine by using OVF, vCenter Server generates an OVF environment XML
document for the virtual machine. The OVF environment is a secure one-way communication channel
between vCenter Server and the guest OS of the virtual machine. To use this feature you must enable
OVF environment transport in the OVF descriptor of your virtual machine. The vCenter Extension
vService makes the OVF environment available to the virtual machine in an ISO image that it locates in
the first CD-ROM drive of the virtual machine.

The vCenter Extension vService uses the OVF environment to pass connection parameters to the guest
operating system in the virtual machine. The vCenter Extension vService adds a
`<vServiceEnvironmentSection>` element to the OVF environment of the virtual machine. The
`<vServiceEnvironmentSection>` element contains the following connection parameters that a script
running within the guest operating system requires to register with vCenter Server as an extension.

- Communication parameters that allow the virtual machine to make a secure connection to the
  vCenter Extension vService Guest API:

  - HTTPS URL to the vCenter Extension vService Guest API.

  - Authentication token that authenticates the virtual machine with the vCenter Extension vService.

  - SSL thumbprint of an X509 certificate that the vCenter Extension vService uses to establish a
    secure HTTPS connection with the virtual machine.

- Communication parameters that allow the guest operating system to make a secure connection to
  vCenter Server:

  - IP address of the vCenter Server instance

  - SSL thumbprint of the X509 certificate that vCenter Server uses to make a secure HTTPS
    connection to the virtual machine

  - HTTP port on which the vCenter Server is reachable

  - HTTPS port on which the vCenter Server is reachable

■ Managed object reference of the virtual machine in vCenter Server

Important The `<vServiceEnvironmentSection>` element contains sensitive data. Take special care within the guest operating system to prevent other users from accessing it. When the connection to vCenter Server is established, the guest operating system no longer needs the OVF environment so you should eject the CD-ROM. Ejecting the CD-ROM deletes the OVF environment from the datastore.

Prerequisites

You have developed an extension running in a virtual machine that you deliver using OVF.

Procedure

1 Add a `<vServiceDependencySection>` element to the OVF descriptor file for the virtual machine.

 You nest the `<vServiceDependencySection>` element in the `<VirtualSystem>` element.

```
<Envelope>
 [...]
 <VirtualSystem>
  [...]
  <vmw:vServiceDependencySection>
  </vmw:vServiceDependencySection>
  [...]
 </VirtualSystem>
 [...]
</Envelope>
```

2 Add the URLs to the standard OVF and VMware OVF schemas.

```
<vmw:vServiceDependencySection xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
                               xmlns:vmw="http://www.vmware.com/schema/ovf">
</vmw:vServiceDependencySection>
```

3 Set the `ovf:required` attribute.

Option	Description
`ovf:required="true"`	Users can only deploy the OVF on vCenter Server 5.x. Users cannot power on the virtual machine if it is not bound to the vCenter Extension vService.
`ovf:required="false"`	Users can deploy the OVF vCenter Server 4.x and 5.x, but the virtual machine only integrates with the vCenter Extension vService in vCenter Server 5.x.

```
<vmw:vServiceDependencySection xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
                               xmlns:vmw="http://www.vmware.com/schema/ovf"
                               ovf:required="true"
                               vmw:id="installation" >
</vmw:vServiceDependencySection>
```

4 Set the type of the dependency to `com.vmware.vservice.extension`, to bind the extension to the vCenter Extension vService.

```
<vmw:vServiceDependencySection xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
                                 xmlns:vmw="http://www.vmware.com/schema/ovf"
                                 ovf:required="true"
                                 vmw:id="installation" >
    <Info>A vService dependency</ovf:Info>
    <vmw:Type>com.vmware.vservice.extension</vmw:Type>
</vmw:vServiceDependencySection>
```

5 Provide a name and a description for the vService dependency, that appears in vService Manager.

```
<vmw:vServiceDependencySection xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
                                 xmlns:vmw="http://www.vmware.com/schema/ovf"
                                 ovf:required="true"
                                 vmw:id="installation" >
    <ovf:Info>A vService dependency</ovf:Info>
    <vmw:Type>com.vmware.vservice.extension</vmw:Type>
    <vmw:Name>dependency_name</vmw:Name>
    <vmw:Description>dependency_description</vmw:Description>
    <vmw:Configuration />
</vmw:vServiceDependencySection>
```

6 Enable OVF environment transport by setting the `ovf:transport` attribute of the `<VirtualHardwareSection>` element to `iso`.

```
<Envelope>
 [...]
 <VirtualSystem>
  [...]
  <VirtualHardwareSection ovf:transport="iso">
    [...]
  </VirtualHardwareSection>
  [...]
 </VirtualSystem>
 [...]
</Envelope>
```

You configured the OVF file for a extension to bind the extension to the vCenter Extension vService and to make the connection information for the vCenter Server instance available over OVF transport.

Example: vService Dependency Section in the OVF Descriptor File

The following code extract shows an example of a `<vServiceDependencySection>` element in an OVF descriptor file.

```
<vmw:vServiceDependencySection xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
                                 xmlns:vmw="http://www.vmware.com/schema/ovf"
                                 ovf:required="true"
                                 vmw:id="installation" >
```

```
    <Info>A vService dependency</Info>
    <vmw:Type>com.vmware.vservice.extension</vmw:Type>
    <vmw:Name>vCenter Extension Installation</vmw:Name>
    <vmw:Description>
        This appliance requires a binding to the vCenter Extension vService,
        which allows it to register automatically as a vCenter Extension at runtime.
    </vmw:Description>
    <vmw:Configuration />
  </vmw:vServiceDependencySection>
```

Provide a Script in the Extension Virtual Machine to Register as a vCenter Extension

You must write a script that you run in the virtual machine that contains the extension.

The script reads the URL of the vCenter Extension vService Guest API, SSL thumbprint, and authentication token in the `<vServiceEnvironmentSection>` element of the Open Virtualization Format (OVF) environment XML file. The script authenticates itself with the vCenter Extension vService using the authentication token from the vService environment, generates a self-signed certificate, and sends the extension key and certificate to the `RegisterExtension` function from the vCenter Extension vService Guest API.

The script that you run in the guest operating system of the virtual machine must perform the following functions.

Prerequisites

- You have developed an extension running in a virtual machine that you deliver using OVF.

- You have configured the OVF descriptor file of the extension to depend on the vCenter Extension vService and enabled OVF transport.

Procedure

1 Parse the `<vServiceEnvironmentSection>` section in the OVF environment file.

2 Create a self-signed X509 certificate.

   For example, create the certificate by using OpenSSL.

3 Send an `HTTP POST` request to the vCenter Extension vService.

   The script obtains the URL to the vCenter Extension vService from the `<vServiceEnvironmentSection>`.

4 Include in the `HTTP` header the authentication token value from the `<vServiceEnvironmentSection>` element of the OVF environment file.

   "evs-token: *Token value*"

5 Verify that the SSL thumbprint of the server-side certificate of the HTTPS connection is the same as that of the `X509Thumbprint` value from the `<vServiceEnvironmentSection>` element of the OVF environment file.

6   In the body of the `HTTP POST` request, pass the extension key of the extension and the PEM encoding of the certificate to the `RegisterExtension` function from the vCenter Extension vService Guest API.

```
<RegisterExtension xmlns="http://www.vmware.com/schema/vservice/ExtensionVService">
   <Key>com.mycompany.extensionkey</Key>
   <Certificate>
   -----BEGIN CERTIFICATE-----
   MIICRTCCAa4CCQDC/hX5KA9rSzANBgkqhkiG9w0BAQUFADBnMQswCQYDVQQGEwIu
   LjEKMAgGA1UECBMBLjEKMAgGA1UEBxMBLjEPMA0GA1UEChMGVk13YXJlMQowCAYD
   VQQLEwEuMREwDwYDVQQDEwhFVlMgZGVtbzEQMA4GCSqGSIb3DQEJARYBLjAeFw0x
   MDA3MDYwNzUwNTVaFw0xMDA4MDUwNzUwNTVaMGcxCzAJBgNVBAYTAi4uMQowCAYD
   VQQIEwEuMQowCAYDVQQHEwEuMQ8wDQYDVQQKEwZWTXdhcmUxCjAIBgNVBAsTAS4x
   ETAPBgNVBAMTCEVWUyBkZW1vMRAwDgYJKoZIhvcNAQkBFgEuMIGfMA0GCSqGSIb3
   DQEBAQUAA4GNADCBiQKBgQC4/XVcMhvNixk35iWl3nn1KHVSgUE18TuQBj7spNUc
   y506RmV8BR847jg9fHl7aErShOQ8RT/EuEEUGey4U1dB2pSocoYldtp2r4g/Lcew
   ZuuyQh2+MC0YzeFe+nyxBDHa0BGUId0dQH9nrjyboW/kNIrWfDkXnxxtq6pQAmFw
   +QIDAQABMA0GCSqGSIb3DQEBBQUAA4GBAElC68z59fuicYUa4fGWBuXNxzb+uqWF
   +cnf78lctBY1pr1DcEedhyww2SYbaGh/xGCc1zqO5kqYhIexQbN/2Vxaol9lJc/n
   vRfQRCp+HaIFTJMu4mVZ2GsYSp/tZSGgiBBQAUXqCLxFQr0eQ29b9rj4Q3/1N+7i
   hbVOln67TOBZ
   -----END CERTIFICATE-----
   </Certificate>
</RegisterExtension>
```

The vCenter Extension vService calls the `RegisterExtension` function and registers the extension with vCenter Server.

Example: `vServiceEnvironmentSection` in the OVF Environment XML File

If you add a vService dependency in an OVF descriptor from which you deploy an extension, the vCenter Extension vService generates and adds a `<vServiceEnvironmentSection>` element to the OVF environment of the virtual machine that it deploys.

```
<ve:vServiceEnvironmentSection xmlns:ve="http://www.vmware.com/schema/ovfenv"

xmlns:evs="http://www.vmware.com/schema/vservice/ExtensionVService"
                               ve:bound="true" ve:id="installation"
                               ve:type="com.vmware.vservice.extension">
   <evs:GuestApi>
        <evs:URL>https://192.168.1.42/vsm/extensionService</evs:URL>
        <evs:Token>1efc34a14232f81a245b9e8172f7a383fdeab312</evs:Token>
        <evs:X509Thumbprint>2b:04:e6:7d:8c:7b:73:70:d4:29:32:ed:
96:11:2b:ae:b4:a0:28:78</evs:X509Thumbprint>
   </evs:GuestApi>
   <evs:VCenterApi>
        <evs:IP>192.168.1.42</evs:IP>
        <evs:X509Thumbprint>2b:04:e6:7d:8c:7b:73:70:d4:29:32:ed:
96:11:2b:ae:b4:a0:28:78</evs:X509Thumbprint>
        <evs:HttpPort>80</evs:HttpPort>
```

```
        <evs:HttpsPort>443</evs:HttpsPort>
        <evs:SelfMoRef>VirtualMachine:vm-246</evs:SelfMoRef>
    </evs:VCenterApi>
 </ve:vServiceEnvironmentSection>
```

Deploy an Extension in the vSphere Web Client By Using the vCenter Extension vService

If you configure the virtual machine that runs an extension to use the vCenter Extension vService, you can deploy the extension directly in the vSphere Web Client.

Prerequisites

- You have developed an extension running in a virtual machine that you deliver using OVF.

- You have configured the OVF descriptor file of the extension to depend on the vCenter Extension vService and enabled OVF transport.

- You have provided a script in the virtual machine that runs the extension that reads the OVF environment file to obtain the connection parameters from the vCenter Extension vService and uses them to register as an extension with vCenter Server.

Procedure

1 Log in to the vSphere Web Client.

2 Select **Home > Hosts & Clusters > Hosts** tab.

3 Select the host.

4 From the **Actions** pull-down menu, select **Deploy OVF Template**.

5 (Optional) If your vCenter Server version is earlier than 6.7, verify that the VMware Client Integration Plug-in is installed for your browser.

6 Type the URL to the OVF file from which to deploy the extension.

7 Select a host, datastore, disk format, and network on which to run the extension.

8 Under **Service Bindings**, select **vCenter Extension vService**.

   You must have the **Extension > Register extension** privilege to do this.

9 Select **Power on after deployment** and select **Finish**.

   When the OVF has deployed and the virtual machine has powered on, a script in the guest operating system connects to the vCenter Extension vService and registers the virtual machine as a vCenter Server extension.

You deployed an extension from OVF by using the vCenter Extension vService.

vCenter Extension vService XML Schema

The vCenter Extension vService XML Schema defines the `<vServiceEnvironmentSection>` in the OVF environment of the virtual machine that contains the extension. It also defines the `RegisterExtension` function that registers the extension with vCenter Server.

```xml
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:evs="http://www.vmware.com/schema/vservice/ExtensionVService"
        targetNamespace="http://www.vmware.com/schema/vservice/ExtensionVService"
        elementFormDefault="qualified" attributeFormDefault="qualified">

    <!-- RegisterExtension_Type:
        The command type used in the guest API when registering as a
        vCenter extension.
    -->
    <complexType name="RegisterExtension_Type">
      <sequence>
        <!-- The key of the extension to register -->
        <element name="Key" type="string" />
        <!-- The PEM-encoded certificate of the extension to register -->
        <element name="Certificate" type="string" />
        <any processContents="lax" minOccurs="0" maxOccurs="unbounded" />
      </sequence>
      <anyAttribute namespace="##any" processContents="lax"/>
    </complexType>

    <element name="RegisterExtension" type="evs:RegisterExtension_Type" />

    <!-- GuestApi: Contains information for the guest software about how to connect to the
        Extension vService.
    -->
    <element name="GuestApi">
      <complexType>
       <sequence>
          <!-- The URL of the Extension vService guest API -->
          <element name="URL" type="anyURI" />
          <!-- Authentication token for the Extension vService guest API -->
          <element name="Token" type="string" />
          <!-- The X509 thumbprint of the Extension vService server
            certificate.
          -->
          <element name="X509Thumbprint" type="string" />
          <any processContents="lax" minOccurs="0" maxOccurs="unbounded" />
      </sequence>
      <anyAttribute namespace="##any" processContents="lax"/>
     </complexType>
    </element>

    <!-- VCenterApi:
        Contains information for the guest software about how to connect
        to vCenter Server.
```

```
    -->

    <element name="VCenterApi">
      <complexType>
       <sequence>
          <!-- The IP address of the vCenter server -->
          <element name="IP" type="string" />
          <!-- The X509 thumbprint of the vCenter server certificate -->
          <element name="X509Thumbprint" type="string" />
          <!-- The HTTP port of the vCenter server -->
          <element name="HttpPort" type="int" />
          <!-- The HTTPS port of the vCenter server -->
          <element name="HttpsPort" type="int" />
          <!--
              The managed object reference of this virtual machine
              in the vCenter server
          -->
          <element name="SelfMoRef" type="string" />
          <any processContents="lax" minOccurs="0" maxOccurs="unbounded" />
       </sequence>
       <anyAttribute namespace="##any" processContents="lax"/>
      </complexType>
    </element>
  </schema>
```