vSphere SDK for .NET Developer's Guide

VMware vSphere SDK for .NET 4.1

EN-000365-00

vmware®

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

http://www.vmware.com/support/

The VMware Web site also provides the latest product updates.

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

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

About This Book

The *vSphere SDK for .NET Developer's Guide* provides information about setting up the development environment and developing applications using the vSphere SDK for .NET 4.1. VMware provides several different SDK products, each of which targets different developer communities and target platforms. This guide is intended for developers who are creating applications for managing VMware vSphere components.

Intended Audience

This book is intended for anyone who needs to set up the development environment to develop applications using the vSphere SDK for .NET 4.1. vSphere SDK for .NET users typically include software developers creating .NET applications using MS Visual Studio .NET.

VMware Technical Publications Glossary

VMware Technical Publications provides a glossary of terms that might be unfamiliar to you. For definitions of terms as they are used in VMware technical documentation go to http://www.vmware.com/support/pubs.

Document Feedback

VMware welcomes your suggestions for improving our documentation. If you have comments, send your feedback to docfeedback@vmware.com.

Technical Support and Education Resources

The following sections describe the technical support resources available to you. To access the current version of this book and other books, go to http://www.vmware.com/support/pubs.

Online and Telephone Support

To use online support to submit technical support requests, view your product and contract information, and register your products, go to http://www.vmware.com/support.

Customers with appropriate support contracts should use telephone support for the fastest response on priority 1 issues. Go to http://www.vmware.com/support/phone_support.html.

Support Offerings

To find out how VMware support offerings can help meet your business needs, go to http://www.vmware.com/support/services.

VMware Professional Services

VMware Education Services courses offer extensive hands-on labs, case study examples, and course materials designed to be used as on-the-job reference tools. Courses are available onsite, in the classroom, and live online. For onsite pilot programs and implementation best practices, VMware Consulting Services provides offerings to help you assess, plan, build, and manage your virtual environment. To access information about education classes, certification programs, and consulting services, go to http://www.vmware.com/services.

Getting Started with vSphere SDK for .NET 1

This chapter provides general information on VMware vSphere SDK for .NET 4.1. It discusses the following topics:

Application Scope of vSphere SDK for .NET

The VMware vSphere SDK for .NET is a client-side framework from VMware that simplifies the programming effort associated with the vSphere API and server-side object model. It is a part of VMware vSphere PowerCLI, which provides easy-to-use C# and PowerShell interface to vSphere APIs. Using vSphere SDK for .NET you can create, customize, or manage vSphere inventory objects using vSphere APIs calls. For more information on vSphere PowerCLI, visit www.vmware.com/go/powercli.

To find a general description of the server-side vSphere object model and information about how to access and modify server-side objects using vSphere SDK for .NET, see Chapter 2, "Programming with .NET Assemblies," on page 9.

Setting Up the Development Environment

vSphere SDK for .NET is intended for use with Visual Studio 2005 .NET or later.

To get started writing and running vSphere .NET applications

1 Launch VIsual Studio 2005 .NET or later.

2 Create a new project or open an existing project.

3 Reference the vSphere API .NET Library (`VMware.Vim.dll`) from the GAC.

4 Use `vClient` and other `VMware.Vim` namespace classes to manage your vSphere inventory.

Programming with .NET Assemblies 2

vSphere SDK for .NET allows you to find objects, access and modify their properties, and invoke methods on the server. This chapter illustrates how to work with server-side objects, vSphere .NET objects, and view objects in the following topics:

vSphere SDK for .NET Basics

This section explores the basics of the vSphere SDK for .NET and the vSphere API model.

Understanding Server-Side Objects

When you run a vSphere SDK for .NET application, your goal is always to access and potentially analyze or modify server-side objects. You need the name of the vSphere API objects and often their properties and method names. For example, if you want to power down a virtual machine, you must know how to find the corresponding object, what the name of the power down method is, and how to call that method.

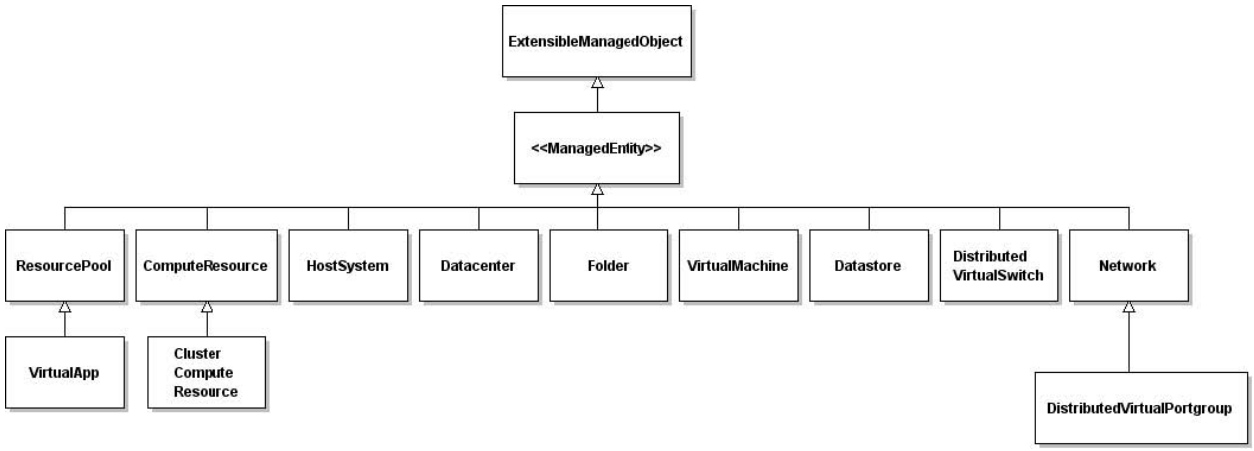
NOTE The *vSphere API Reference Guide* gives reference documentation for all vSphere API objects. Some users might also find the vSphere SDK Programmer's Guide helpful. It is available from the SDK download site at http://www.vmware.com/download/sdk/index.html.

A *managed object* is the primary type of object in the vSphere object model. A managed object is a data type available on the server that consists of properties and operations. Each managed object has properties and provides various services (operations or methods). Figure 2-1 shows the `ExtensibleManagedEntity` hierarchy as an example.

The different managed objects define the entities in the inventory as well as common administrative and management services such as managing performance (`PerformanceManager`), finding entities that exist in the inventory (`SearchIndex`), disseminating and controlling licenses (`LicenseManager`), and configuring alarms to respond to certain events (`AlarmManager`). See the *vSphere API Reference* for a detailed discussion.

A managed object reference (represented by a `ManagedObjectReference`) identifies a specific managed object on the server, encapsulates the state and methods of that server-side objects, and makes the state and methods available to client applications. Clients invoke methods (operations) on the server by passing the appropriate managed object reference to the server as part of the method invocation.

Figure 2-1. ExtensibleManagedEntity Hierarchy



The different managed objects define the entities in the inventory as well as common administrative and management services such as managing performance (`PerformanceManager`), finding entities that exist in the inventory (`SearchIndex`), disseminating and controlling licenses (`LicenseManager`), and configuring alarms to respond to certain events (`AlarmManager`). See the *vSphere API Reference* for a detailed discussion.

A managed object reference (represented by a `ManagedObjectReference`) identifies a specific managed object on the server, encapsulates the state and methods of that server-side objects, and makes the state and methods available to client applications. Clients invoke methods (operations) on the server by passing the appropriate managed object reference to the server as part of the method invocation.

Example 2-1. Accessing Server-Side Inventory Objects

This example illustrates accessing server-side inventory objects.

```
/* ************************************************************
 * Copyright (C) 2009 VMware, Inc.
 * All Rights Reserved
 * ************************************************************/
using System;
using System.Collections.Specialized;
using VMware.Vim;
namespace Samples {
   /// <summary>
   /// This example gets ServiceContent data object
   /// and creates a reference to the diagnosticManager.
   /// The application shows the last five lines of the log.
   /// </summary>
   public class Example2_3 {
      private string serviceUrl = null;
      private string userName = null;
      private string password = null;
      public Example2_3(string[] args) {
         //Parse the input arguments.
         ParseArguments(args);
      }
      public void PrintDiagnosticLog() {
         VimClient client = new VimClient();
         // Connect to the vSphere web service.
         client.Connect(serviceUrl);
         // Login with username/password credentials.
         client.Login(userName, password);
         // Get DiagnosticManager.
         DiagnosticManager diagMgr = (DiagnosticManager) client.GetView(
            client.ServiceContent.DiagnosticManager, null);
         // Obtain the last line of the logfile by setting an arbitrarily large
         // line number as a starting point.
```

```
DiagnosticManagerLogHeader log =
            diagMgr.BrowseDiagnosticLog(null, "hostd", 999999999, null);
         int lineEnd = log.LineEnd;
         // Get the last five lines of the log.
         int start = lineEnd - 5;
         log = diagMgr.BrowseDiagnosticLog(null, "hostd", start, null);
         foreach (string line in log.LineText) {
            Console.WriteLine(line);
         }
         // Logout from the vSphere server.
client.Disconnect();
      }
      /// <summary>
      /// Define the main entry point for the application.
      /// </summary>
      public static void Main(string[] args) {
         // Check the input parameter count.
         if (args.Length < 3) {
            Console.Write("Usage: Example2_3 <serviceUrl> <username> <password>");
            return;
         }
         Example2_3 example2_3 = new Example2_3(args);
         // Perform the required operation.
         try {
            example2_3.PrintDiagnosticLog();
         } catch (VimException ex) {
            // Handle VimException to determine vSphere server faults.
            Console.WriteLine(
               "A server fault of type {0} with message '{1}' occured while performing requested
               operation.",
               ex.MethodFault.GetType().Name,
               ex.Message);
         } catch (Exception ex) {
            // Handle a user code exception.
            Console.WriteLine(
               "An exception of type {0} with message '{1}' occured while performing requested
               operation.",
               ex.GetType(),
               ex.Message);
         }
      }
      private void ParseArguments(string[] args) {
         serviceUrl = args[0];
         userName = args[1];
         password = args[2];
      }
   }
}
```

Understanding vSphere .NET Objects

A vSphere SDK for .NET view object is a .NET object with the following characteristics:

- It includes properties and methods that correspond to the properties and operations of the server-side managed object.
- It is a static copy of a server-side managed object and is not automatically updated when the object on the server changes. See "Updating View Objects" on page 12.
- It includes additional methods (beyond the operations offered in the server-side managed object), specifically:
  - A blocking and a non-blocking method for each (non-blocking) operation provided by the server-side managed object.
  - A method that updates the state of any client-side view object with current data from the server. See "Updating View Objects" on page 12.

The vSphere SDK for .NET maps server-side operations to client-side .NET view object methods. For each operation defined on a server managed object, the vSphere SDK for .NET creates a corresponding view method when it creates the view object.

By default, all server-side operations available in the vSphere API are non-blocking operations listed in the *vSphere API Reference Guide* (`<opname>_Task()` method). The vSphere SDK for .NET also provides a blocking (synchronous) method (`<opname>()` method) that provides the same functionality as `<opname>_Task()`, but does not return a reference to a task object. If the operation in the *vSphere API Reference Guide* is described as `<opname>_Task()`, then you can use both non-blocking and blocking `<opname>()` methods in your vSphere .NET application.

To find all available operations for each managed object, see the *vSphere API Reference Guide*.

Updating View Objects

The properties values of a view object represent the state of the server-side object at the time the view was created. In a production environment, the state of managed objects on the server is likely to be changing constantly. The property values, however, are not updated automatically. You can refresh the values of client-side view objects with the corresponding server-side objects values using the vSphere SDK for .NET `UpdateViewData()` method

Example 2-2. Using the `UpdateViewData()` Method to Refresh a View Object Data.

```
using VMware.Vim;
using System.Collections.Specialized;
namespace Samples {
   public class Example2_2 {
      public void PowerOffVM() {
         VimClient client = new VimClient();

   ...
         IList<EntityViewBase> vmList =
            client.FindEntityViews(typeof(VirtualMachine), null, filter, null);
         // Power off the virtual machines.
         foreach (VirtualMachine vm in vmList) {
            // Refresh the state of each view.
            vm.UpdateViewData();
            if (vm.Runtime.PowerState == VirtualMachinePowerState.poweredOn) {
               vm.PowerOffVM();
               Console.WriteLine("Stopped virtual machine: {0}", vm.Name);
            } else {
               Console.WriteLine("Virtual machine {0} power state is: {1}",
                  vm.Name, vm.Runtime.PowerState);
            }
         }
...
```

Versioning Support

Three major versions of the vSphere API are available - VI API 2 (ESX 3.0.x/VirtualCenter 2.0.x), VI API 2.5 (ESX 3.5.x/ VirtualCenter 2.5.x), and vSphere API 4.0 (ESX 4.0/ vCenter 4.0). VI API 2 .NET applications can connect to VI API 2.5 and VI API 3.0 servers, but can not use the features that are new in VI API 3.0.x and vSphere API 4.0. The vSphere SDK for .NET eliminates this restriction and allows the applications built using it, to access the full functionality of the ESX host or vCenter version they are connecting to.

Writing vSphere .NET Applications

This section illustrates how to write .NET applications for managing vSphere using vSphere SDK for .NET.

Creating and Using Filters

Filters are used to reduce large sets of output data by retrieving only the objects that correspond to the specified filter criteria. vSphere SDK for .NET allows you to define and use filters to select specific objects based on property values.

Using Filters with VimClient.FindEntityView() or VimClient.FindEntityViews()

To save time when calling `VimClient.FindEntityView()` or `VimClient.FindEntityViews()`, you can apply one or more filters to select a subset of objects based on property values. For example, instead of retrieving all virtual machine objects in a datacenter, you can obtain only those, whose names begin with a certain prefix.

To apply a filter to the results of `VimClient.FindEntityView()` or `VimClient.FindEntityViews()`, you can supply an optional filter parameter. The value of the parameter is a `NameValueCollection` object containing one or more pairs of filter criteria. Each of the criteria consists of a property path and a match value. The match value can be either a string, or a regular expression object. If the match value is a string, the property value of the target objects must be exactly the same as the string.

Example 2-3. Filtering Virtual Machines by Power State

This example retrieves all virtual machines, whose power state is `PoweredOff`.

```
NameValueCollection filter = new NameValueCollection();

filter.Add("Runtime.PowerState", "PoweredOff")
```

You can also match objects using regular expressions. In this case, the property value must contain the regular expression, specified in the filter.

Example 2-4. Filtering Objects by Name

This example retrieves all virtual machines, whose names start with “Test“.

```
NameValueCollection filter = new NameValueCollection();

filter.Add("name", "^Test");
```

Example 2-5. A Filter for Creating Views of Windows-Based Virtual Machines Only

This example illustrates using `VimClient.FindEntityViews()` in combination with a filter. It retrieves a list of virtual machine objects, whose guest operating system names contain the string Windows.

```
NameValueCollection filter = new NameValueCollection();
filter.Add("Config.GuestFullName", "Windows");

IList<EntityViewBase> vmList =
   client1.FindEntityViews(typeof(VirtualMachine), null, filter, null);

// Print VM names
foreach (VirtualMachine vm in vmList) {
   Console.WriteLine(vm.Name);
```

Example 2-6. A Multiple Criteria Filter

This example demonstrates using a filter with multiple criteria. It retrieves the virtual machines that correspond to the following requuirements:

- The guest operating system is Windows.
- The virtual machineis powered on.

```
NameValueCollection filter = new NameValueCollection();
filter.Add("Runtime.PowerState", "PoweredOn");
filter.Add("Config.GuestFullName", "Windows");

IList<EntityViewBase> vmList =
   client1.FindEntityViews(typeof(VirtualMachine), null, filter, null);

// Print VM names
foreach (VirtualMachine vm in vmList) {
   Console.WriteLine(vm.Name);
```

Example 2-7. A Basic Pattern for Error Handling

This example illustrates a basic pattern implementation of error handling in the vSphere SDK for .NET.

```
try {
            // call operations
         } catch (VimException ex) {
            if (ex.MethodFault is InvalidLogin) {
               // Handle Invalid Login error
            } else {
               // Handle other server errors
            }
         } catch (Exception e) {
            // Handle user code errors
         }
```

Handling Server Errors

Because the vSphere API is hosted as a Web service, server errors are reported as SOAP exception that contains a vSphere API `SoapFault` object. The *vSphere API Reference Guide* lists a SOAP fault for each task inside each managed object. The vSphere SDK for .NET runtime performs additional error handling by translating the vSphere API `SoapFault` object from `SoapException.Detail` property into a `MethodFault` descendant object and throwing a `VimException` exception. The vSphere API `SoapFault` is located in the `VimException.MethodFault` property.

Saving and Using Sessions

The vSphere SDK for .NET `VimClient` class includes several methods for saving and restoring sessions. This enables you to maintain sessions across applications. Instead of storing passwords in applications, you can call the `LoadSession()` method with the name of the session file. The session file does not expose password information, and this enhanced security.

To save a session to a file, call `SaveSession()`with the file name:

```
VimClient client1 = new VimClient();
client1.Connect("https://<hostname>/sdk");
client1.Login("user", "pass");
client1.SaveSession("VimSession.txt");
```

To use the session in another application, call `LoadSession()` with the name of the session file:

```
VimClient client2 = new VimClient();
client2.Connect("https://<hostname>/sdk");
client2.LoadSession("VimSession.txt");
client2.FindEntityView(typeof(VirtualMachine), null, null, null);
```

Sample 3

The following sample .NET application demonstrates applying the `VirtualMachine` class methods on a virtual machine. The virtual machine is retrieved by `FindEntityView` using a filter by name, and the virtual machine host is retrieved by the `GetView` method. The sample also illustrates the use of error handling described in the section "Handling Server Errors" on page 14.

```
/* ************************************************************
 * Copyright (C) 2008 VMware, Inc.
 * All Rights Reserved
 * ************************************************************/
using System;
using System.Collections.Generic;
using System.Text;
using VMware.Vim;
using System.Collections.Specialized;

namespace Samples {

   /// <summary>
   /// Performs poweron, poweroff, suspend and reset
   /// operations on the VM and reboot, shutdown and
   /// standby operations on the Guest OS.
   /// </summary>
   public class VmPowerOps {

      private string serviceUrl = null;
      private string userName = null;
      private string password = null;
      private string vmName = null;
      private string operation = null;

      public VmPowerOps(string[] args) {
         //parse input arguments
         ParseArguments(args);
      }

      public void DoPowerOps() {

         VimClient client = new VimClient();
         // connect to the vSphere web service
         client.Connect(serviceUrl);
         // Login using username/password credentials
         client.Login(userName, password);

         // create filter by VM name
         NameValueCollection filter = new NameValueCollection();
         filter.Add("name", "^" + vmName + "$");
```

```
            // get the VirtualMachine view object
            VirtualMachine vm =
               (VirtualMachine) client.FindEntityView(typeof(VirtualMachine), null, filter, null);

            if (vm != null) {
               // get the host view of the virtual machine
               HostSystem host = (HostSystem) client.GetView(vm.Runtime.Host, null);

               switch (operation) {
                  case "on":
                     vm.PowerOnVM(vm.Runtime.Host);
                     Console.WriteLine(
                        "Virtual Machine '{0}' under host '{1}' powered on successfully.", vm.Name,
host.Name);
                     break;
                  case "off":
                     vm.PowerOffVM();
                     Console.WriteLine(
                        "Virtual Machine '{0}' under host '{1}' powered off successfully.", vm.Name,
host.Name);
                     break;
                  case "suspend":
                     vm.SuspendVM();
                     Console.WriteLine(
                        "Virtual Machine '{0}' under host '{1}' suspended successfully.", vm.Name,
host.Name);
                     break;
                  case "reset":
                     vm.ResetVM();
                     Console.WriteLine(
                        "Virtual Machine '{0}' under host '{1}' reset successfully", vm.Name,
host.Name);
                     break;
                  case "rebootGuest":
                     vm.RebootGuest();
                     Console.WriteLine(
                        "Virtual Machine '{0}' under host '{1}' reboot successfully.", vm.Name,
host.Name);
                     break;
                  case "shutdownGuest":
                     vm.ShutdownGuest();
                     Console.WriteLine(
                        "Virtual Machine '{0}' under host '{1}' shutdown successfully.", vm.Name,
host.Name);
                     break;
                  case "standbyGuest":
                     vm.StandbyGuest();
                     Console.WriteLine(
                        "Virtual Machine '{0}' under host '{1}' put into standby mode.", vm.Name,
host.Name);
                     break;
                  default:
                     Console.WriteLine("Invalid operation: {0}", operation);
                     break;
               }
            } else {
               Console.WriteLine("Unable to find VirtualMachine named '{0}' in Inventory", vmName);
            }

            // logout from the vSphere server
            client.Disconnect();
         }

         /// <summary>
         /// define the main entry point for the application
         /// </summary>
```

```
      public static void Main(string[] args) {
         // check the input parameter count
         if (args.Length < 5) {
            Console.Write("Usage: VmPowerOps <serviceUrl> <username> <password> <vmname> ");
            Console.WriteLine("<on|off|suspend|reset|rebootGuest|shutdownGuest|standbyGuest>\n");

            return;
         }

         VmPowerOps vmPowerOps = new VmPowerOps(args);

         // perform the required operation
         try {
            vmPowerOps.DoPowerOps();
         } catch (VimException ex) {
            // handle VimException to determine vSphere server faults
            Console.WriteLine(

               "A server fault of type {0} with message '{1}' occured while performing requested
operation.",
               ex.MethodFault.GetType().Name,
               ex.Message);
         } catch (Exception ex) {
            // Handle user code exception
            Console.WriteLine(
               "An exception of type {0} with message '{1}' occured while performing requested
operation.",
               ex.GetType(),
               ex.Message);
         }
      }

      private void ParseArguments(string[] args) {
         serviceUrl = args[0];
         userName = args[1];
         password = args[2];
         vmName = args[3];
         operation = args[4];
      }
   }
}
```