Developer's Guide

VMware Infrastructure .NET Toolkit 1.5

VI .NET Toolkit Developer's Guide
Item: EN-000133-00

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

This book, the *Developer's Guide,* provides information about setting up the development environment and developing applications using the VMware Infrastructure .NET Toolkit 1.5. 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 Infrastructure components.

Intended Audience

This book is intended for anyone who needs to set up the development environment to develop applications using the VMware Infrastructure .NET Toolkit 1.5. VI .NET Toolkit users typically include software developers creating .NET applications using MS Visual Studio .NET.

Document Feedback

VMware welcomes your suggestions for improving our documentation. 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 versions of other VMware manuals, go to:

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

Online Support

You can submit questions or post comments to the Developer Community: SDKs and APIs forum, which is monitored by VMware technical support and product teams. To access the forum, go to:

http://communities.vmware.com/community/developer

Support Offerings

Find out how VMware support offerings can help meet your business needs. Go to:

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

VMware Education Services

VMware courses offer extensive hands-on labs, case study examples, and course materials designed to be used as on-the-job reference tools. For more information about VMware Education Services, go to:

http://mylearn1.vmware.com/mgrreg/index.cfm

Getting Started with VI .NET Toolkit 1

This chapter provides general information on the VMware Infrastructure .NET Toolkit 1.5. It discusses the following topics:

Application Scope of VI .NET Toolkit

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

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

Setting Up the Development Environment

VI .NET Toolkit is intended for use with Visual Studio 2005 .NET or later.

To get started writing and running VI .NET applications:

1 Launch VIsual Studio 2005 .NET or later.

2 Create a new project or open an existing project.

3 Reference the VI API .NET Library by locating the `VMware.Vim` assembly in the VI Toolkit (for Windows) installation directory. By default, VI Toolkit (for Windows) is installed in `C:\Program Files\VMware\Infrastructure\VIToolkitForWindows`.

4 Use `VIClient` and other `VMware.Vim` namespace classes to manage your VMware Infrastructure inventory.

Programming with .NET Assemblies 2

VI .NET Toolkit 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, VI. NET objects, and view objects in the following topics:

VI .NET Toolkit Basics

This section explores the basics of the VI .NET Toolkit and the VI API model.

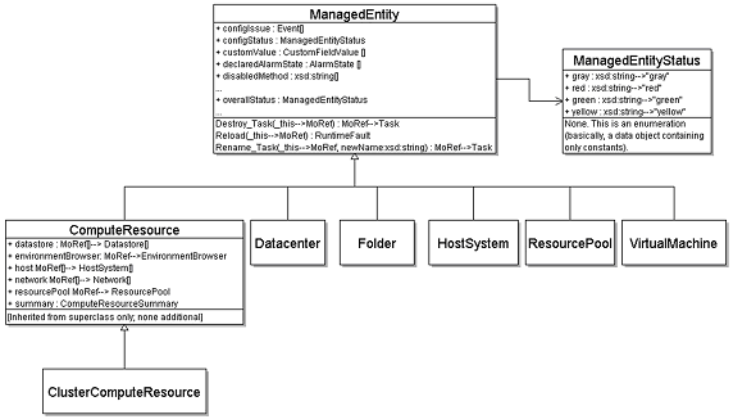
Understanding Server-Side Objects

When you run a VI .NET Toolkit application, your goal is always to access and potentially analyze or modify server-side objects. You need the name of the VI 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 *VI API Reference Guide* gives reference documentation for all VI API objects. Some users might also find the VMware Infrastructure 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 VMware Infrastructure 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 ManagedEntity hierarchy as an example.

Figure 2-1. ManagedEntity 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 *VI 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) 2008 VMware, Inc.
 * All Rights Reserved
 * ************************************************************/
using System;
using System.Collections.Specialized;
using VMware.Vim;
namespace Samples {
   /// <summary>
   /// This example gets ServiceContent data
   /// object, and then creates a reference to the diagnosticManager.
   /// The application shows last 5 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 input arguments
         ParseArguments(args);
      }
      public void PrintDiagnosticLog() {
         VimClient client = new VimClient();
         // connect to VI web service
         client.Connect(serviceUrl);
         // Login using 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 the starting point
         DiagnosticManagerLogHeader log =
```

```
            diagMgr.BrowseDiagnosticLog(null, "hostd", 999999999, null);
         int lineEnd = log.LineEnd;
         // Get the last 5 lines of the log first
         int start = lineEnd - 5;
         log = diagMgr.BrowseDiagnosticLog(null, "hostd", start, null);
         foreach (string line in log.LineText) {
            Console.WriteLine(line);
         }
         // Logout from the VI server
client.Disconnect();
      }
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      public static void Main(string[] args) {
         // check 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 required operation
         try {
            example2_3.PrintDiagnosticLog();
         } catch (VimException ex) {
            // Handle VimException to determine VI 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];
      }
   }
}
```

Understanding VI .NET Objects

A VI. NET Toolkit 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 VI. NET Toolkit maps server-side operations to client-side .NET view object methods. For each operation defined on a server managed object, the VI.NET Toolkit creates a corresponding view method when it creates the view object.

By default, all server-side operations available in the VI API are non-blocking operations listed in the *VI API Reference Guide* (`<opname>_Task()` method). The VI.NET Toolkit 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 *VI API Reference Guide* is described as `<opname>_Task()`, then you can use both non-blocking and blocking `<opname>()` methods in your VI .NET application.

To find all available operations for each managed object, see the *VI 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 VI.NET Toolkit `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 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

Two major versions of the VMware Infrastructure API are available - VI API 2 (ESX 3.0.x/VirtualCenter 2.0.x) and VI API 2.5 (ESX 3.5.x/ VirtualCenter 2.5.x). VI API 2 .NET applications can connect to a VI API 2.5 server, but can not use the features that are new in VI API 2.5. The VI .NET Toolkit eliminates this restriction and allows the applications built using it, to access the full functionality of the ESX host or VirtualCenter version they are connecting to.

Writing VI .NET Toolkit Applications

This section illustrates how to write .NET applications for managing the VMware Infrastructure using the VI .NET Toolkit.

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. The VI. NET Toolkit 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 VI .NET Toolkit.

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

Saving and Using Sessions

The VI.NET Toolkit `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 VI 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 VirtualMachine view object
         VirtualMachine vm =
            (VirtualMachine) client.FindEntityView(typeof(VirtualMachine), null, filter, null);
```

```
            if (vm != null) {
               // Get host view of the VM
               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 VI server
            client.Disconnect();
         }

         /// <summary>
         /// The main entry point for the application.
         /// </summary>
         public static void Main(string[] args) {
            // check 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 required operation
         try {
            vmPowerOps.DoPowerOps();
         } catch (VimException ex) {
            // Handle VimException to determine VI 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];
      }
   }
}
```

Glossary

D **data object**

Complex data type that consists of properties and values only (no operations or methods). Data objects are used throughout the VI API to capture or reflect the state of various properties of managed objects. As implemented in the VI API, data objects are analogous to structures (structs) in C, C++, and other programming languages.

H **hash**

One or more key-value pairs that define the attributes and their values.

I **inventory**

The collection of all managed entities on the server, that is, of all instances of `HostSystem`, `Datacenter`, `VirtualMachine`, `ResourcePool`, `ComputeResource`, `ClusterComputeResource`, and `Folder`.

M **managed entity**

One of the seven managed object types that extend the `ManagedEntity` managed object. The `ManagedEntity` managed object type is an abstract class that defines the base properties and methods for VMware Infrastructure objects, the same kinds of manageable components found in a physical IT infrastructure, such as datacenters and hosts.

managed object

A server-side type that encapsulates properties and operations available on the server. Different objects offer different services (operations, methods). The various managed objects types on the server define common administrative and management services running on a typical datacenter. Such services are 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`).

managed object reference

A type of data object that enables distributed computing for the VMware Infrastructure environment. A managed object reference identifies a specific managed object on the server, and encapsulates the state as well as methods of server-side objects, making them available to client applications. Clients invoke methods (operations) on the server by passing the appropriate managed object reference (mo_ref) to the server, in the method invocation.

V **VI API**

A set of Web services, hosted on ESX and VirtualCenter host systems for providing interfaces to VMware Infrastructure components such as hosts, virtual machines, and data centers, and for operations on these components.

view

A client-side .NET object that the VI .NET Toolkit has populated with the state of one or more server-side managed objects. Client applications and scripts work with view objects rather than with the managed entities that exist on the server. To create a view, call the appropriate `VimClient` method (`Get-View`, `Get-Views`, and so on) with the managed object reference for the entity of interest.