VMware Mirage API Programming Guide

VMware Mirage 5.9

vmware®

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

https://docs.vmware.com/

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

VMware Mirage API Programming Guide 5

1 Setting Up a Development Environment 7
- Enable the WCF HTTP Activation Feature in Windows Server 2008 R2 7
- Enable the WCF HTTP Activation Feature in Windows Server 2012 8
- Set Up a Windows C# Development Environment to Work with .NET 4.5 8
- Set Up a Windows C# Development Environment to Work with .NET 4.0 or Earlier 9
- Set Up a Java Development Environment to Work with .NET 4.5 9
- Set Up a Java Development Environment to Work with .NET 4.0 or Earlier 10

2 Methods 13

3 Performance 47

4 Permissions, Configuration and Logging 49
- Permissions 49
- Configuration 50
- Logging 50

5 Sample Applications 51
- Sample C# Application 51
- Sample Java Application 64

Index 77

VMware Mirage API Programming Guide

The *VMware Mirage API Programming Guide* provides information about developing applications using the VMware Mirage API.

Intended Audience

This book is intended for anyone who needs to develop applications using the Mirage API. Developers typically create client applications using Java or C# (in the Microsoft .NET environment) targeting VMware Mirage. An understanding of Web Services technology and some programming background in one of the stub languages (C# or Java) is required.

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.

Setting Up a Development Environment 1

Mirage API is hosted on Microsoft Internet Information Services (IIS) and is installed with the Mirage Web Manager. For information on installing the Mirage Web Manager, see the *VMware Mirage Installation Guide*.

The URL of the Mirage API is https://*server-address*:7443/mirageapi/MitService.svc.

This chapter includes the following topics:

- "Enable the WCF HTTP Activation Feature in Windows Server 2008 R2," on page 7
- "Enable the WCF HTTP Activation Feature in Windows Server 2012," on page 8
- "Set Up a Windows C# Development Environment to Work with .NET 4.5," on page 8
- "Set Up a Windows C# Development Environment to Work with .NET 4.0 or Earlier," on page 9
- "Set Up a Java Development Environment to Work with .NET 4.5," on page 9
- "Set Up a Java Development Environment to Work with .NET 4.0 or Earlier," on page 10

Enable the WCF HTTP Activation Feature in Windows Server 2008 R2

If the Mirage Web Manager is installed on a Windows Server 2008 R2 machine, you must install the .NET Framework 3.5.1 WCF HTTP Activation feature.

Prerequisites

Verify that Mirage Web Manager is installed on the Windows Server 2008 R2 machine.

Procedure

1 Log in as an administrator.

2 Click **Start > Administrative Tools > Server Manager**.

3 When the Server Manager window displays, click **Features > Add Features**.

4 Expand **.NET Framework 3.5.1 Features**.

5 Expand **WCF Activation**.

6 Select **HTTP Activation** and click **Install**.

7 Follow the prompts and finish the installation.

8 Open a Command Prompt window.

9 Run the `%WINDIR%\Microsoft.NET\Framework\v4.0.30319\aspnet_regiis.exe –iru` command.

Enable the WCF HTTP Activation Feature in Windows Server 2012

If the Mirage Web Manager is installed on a Windows Server 2012 machine, you must install the .NET Framework 4.5 WCF HTTP Activation feature.

Prerequisites

Verify that the Mirage Web manager is installed on the Windows Server 2012 machine.

Procedure

1 Log in as an administrator.

2 Click **Start > Control Panel > Turn Windows features on or off**.

3 Click **Next** until the Select Features window appears.

4 Expand **.NET Framework 4.5 Features**.

5 Expand **WCF Services**.

6 Select **HTTP Activation** and click **Install**.

7 Follow the prompts and finish the installation.

Set Up a Windows C# Development Environment to Work with .NET 4.5

To use the Microsoft C# programming language to develop your applications and the .NET version is 4.5 on the Mirage Server, you must perform certain tasks to set up a C# development environment.

Procedure

1 Get the WSDL file `MitService.wsdl` from https://server-address:7443/mirageapi/MitService.svc?singleWsdl.

2 Open Visual Studio Command Prompt and go to the `wsdl` file directory.

3 Run the command `Svcutil MitService.wsdl`.

Svcutil.exe is the ServiceModel Metadata Utility tool. This command generates a client.

4 Add the generated `MitService.cs` to the C# project.

With the generated client, you can login to Mirage API. For example:

```
ServicePointManager.ServerCertificateValidationCallback = ((sender, certificate, chain,
sslPolicyErrors) => true);
BasicHttpBinding binding = new BasicHttpBinding
{
    AllowCookies = true,
    Security =
    {
        Mode = BasicHttpSecurityMode.Transport
    }
};

//Connect to Mirage Web Management Server
EndpointAddress endpoint = new EndpointAddress(string.Format("https://<server-address>:
7443/mirageapi/MitService.svc", address));
Client = new MitServiceClient(binding, endpoint);
```

```
Client.Login("<username>", "<password>");
Console.WriteLine("Login success!");

//Perform tasks
```

Set Up a Windows C# Development Environment to Work with .NET 4.0 or Earlier

To use the Microsoft C# programming language to develop your applications and the .NET version is 4.0 or earlier on the Mirage Server, you must perform certain tasks to set up a C# development environment.

Procedure

1 Open Visual Studio Command Prompt and go to the `dll` directory.

2 Run the command `svcUtil Web.management.MirageApi.dll`.

By default, the file `Web.Management.MirageApi.dll` is located in `C:\Program Files\Wanova\Mirage API\bin` after you install Mirage.

3 Run the command `svcUtil *.wsdl *.xsd /o:MitService.cs`.

4 Add the generated MitService.cs to the C# project.

With the generated client, you can login to Mirage API. For example:

```
ServicePointManager.ServerCertificateValidationCallback = ((sender, certificate, chain,
sslPolicyErrors) => true);
BasicHttpBinding binding = new BasicHttpBinding
{
    AllowCookies = true,
    Security =
    {
        Mode = BasicHttpSecurityMode.Transport
    }
};

//Connect to Mirage Web Management Server
EndpointAddress endpoint = new EndpointAddress(string.Format("https://<server-address>:
7443/mirageapi/MitService.svc", address));
Client = new MitServiceClient(binding, endpoint);
Client.Login("<username>", "<password>");
Console.WriteLine("Login success!");

//Perform tasks
```

Set Up a Java Development Environment to Work with .NET 4.5

If you plan to use the Java programming language to develop your applications and the .NET version is 4.5 on the Mirage Server, you must perform certain tasks to set up a Java development environment.

Procedure

1 Download Java SOAP library Axis2 from http://axis.apache.org/axis2/java/core.

2 Unzip the file `axis2-1.x.y` to a folder and set the environment variable *%AXIS2_HOME%* to the path of this folder.

3 Get the WSDL file `MitService.wsdl` from https://server-address:7443/mirageapi/MitService.svc?singleWsdl.

4 Generate Java classes with the following command:

```
%AXIS2_HOME%\bin\wsdl2java –uri MitService.wsdl –p com.vmware.mirage.mit
```

With the generated classes, you can login to Mirage API. For example:

```
client = new MitServiceStub("https://server–address:7443/mirageapi/MitService.svc");
final SSLContext sslCtx = SSLContext.getInstance("TLS");
sslCtx.init(null, new TrustManager[] { new TrustAllTrustManager() }, null);
client._getServiceClient()
    .getOptions()
    .setProperty(HTTPConstants.CUSTOM_PROTOCOL_HANDLER, new Protocol("https",
(ProtocolSocketFactory) new SSLProtocolSocketFactory(sslCtx), 7443));
final HttpState httpState = new HttpState();
client._getServiceClient().getOptions()
    .setProperty(org.apache.axis2.transport.http.HTTPConstants.CACHED_HTTP_STATE, httpState);
final Login login = new Login();
login.setUsername(“username”);
login.setPassword(“password”);
client.login(login);

//Perform tasks
```

Set Up a Java Development Environment to Work with .NET 4.0 or Earlier

To use the Java programming language to develop your applications and the .NET version is 4.0 or earlier on the Mirage Server, you must perform certain tasks to set up a Java development environment.

Procedure

1 Open https://server-address:7443/mirageapi/MitService.svc?wsdl and export the certificate `it.atco.com.cer`.

2 Run the command `keytool –import –trustcacerts –keystore $JAVA_HOME/jre/lib/security/cacerts –storepass changeit –noprompt –file it.atco.com.cer` to import the certificate to the trusted store.

3 Run the command `wsdl2java.sh –uri https:// server–address:7443/MirageApi/MitService.svc?wsdl –p vmware.mirage.mit`

The source code folder `src` is created.

4 Copy the source code to your Java project.

With the generated classes, you can login to Mirage API. For example:

```
client = new MitServiceStub("https://server–address:7443/mirageapi/MitService.svc");
final SSLContext sslCtx = SSLContext.getInstance("TLS");
sslCtx.init(null, new TrustManager[] { new TrustAllTrustManager() }, null);
client._getServiceClient()
    .getOptions()
    .setProperty(HTTPConstants.CUSTOM_PROTOCOL_HANDLER, new Protocol("https",
(ProtocolSocketFactory) new SSLProtocolSocketFactory(sslCtx), 7443));
final HttpState httpState = new HttpState();
client._getServiceClient().getOptions()
    .setProperty(org.apache.axis2.transport.http.HTTPConstants.CACHED_HTTP_STATE, httpState);
final Login login = new Login();
login.setUsername(“username”);
```

```
login.setPassword(“password”);
client.login(login);

//Perform tasks
```

Methods

2

Mirage API has 33 methods.

Table 2-1. Mirage API Methods

Method	Description
Login	Authenticates the user.
Logout	Logs out and terminates the current session.
Policy_Query	Queries all the policies in the Mirage Management Server.
Volume_Query	Queries the volumes in the Mirage Management Server.
PendingDevice_Query	Queries the devices that are in the pending assignment state.
Cvd_Query	Queries the CVDs that are in the Mirage Management Server.
Cvd_Get	Gets a CVD by Id.
BaseLayer_Query	Queries all the base layer images in the Mirage Management Server.
AppLayer_Query	Queries all the application layer images in Mirage Management Server.
PendingDevice_CreateNewCvd	Creates a new CVD for the pending devices.
OsMigration_Begin	Starts migration for the migration targets.
OsMigration_BeginDownloadOnly	Starts to download the base layer and app layers for the migration targets.
OsMigration_ApplyDownloadOnlyMigration	Starts to apply the base layer and app layers for the migration targets.
OsMigration_QueryDownloadOnlyInProgress (Deprecated)	Queries the CVDs which are downloading the base layer or app layers.
OsMigration_QueryDownloadOnlyCompleted (Deprecated)	Queries the CVDs which are finished downloading the base layer or app layers.
Cvd_Archive	Archives the CVDs in the Mirage Management system.
Cvd_Delete	Deletes the CVDs in the Mirage Management system.
Cvd_Sync	Synchronizes device information for the CVDs in the Mirage Management system.
Cvd_ApplyPolicy	Applies a policy to the specified CVDs.
PendingDevice_Provision	Provisions the specified devices with a base layer and app layers.
Collection_Query	Queries all CVD collections in the Mirage Management system.
CollectionCvd_Query	Queries all CVDs in the specific collection in the Mirage Management system.
OsMigrationCvd_QueryDownloadOnly	Queries the CVD Ids for download only migration.

Table 2-1. Mirage API Methods (Continued)

Method	Description
Assignment_ApplyDownloadOnly	Applies the downloaded assignments in the Mirage Management system.
Assignment_Query	Queries the assignments in the Mirage Management system.
Cvd_AssignBaseLayer	Assigns a base layer to a CVD.
Cvd_AssignBaseLayerDownloadOnly	Downloads a base layer for a CVD.
Cvd_UpdateAppLayer	Updates app layers on a CVD.
Cvd_UpdateAppLayerDownloadOnly	Downloads app layers for a CVD.
PendingDevice_ProvisionWithProvisionTarget	Provision the pending devices. It provide the ability to change each device's machine name and join domain.
Cvd_EnforceAllLayers	Enforce all the layers on the CVDs.
Cvd_AllowNetworkOperations	Suspend/resume device network operations.
Cvd_Restart	Reboot the device.

For more information about each method, including input parameters, return value, and faults, see the Mirage API Reference.

Login

Login logs a user in to Mirage. You must call this before calling any other method. Otherwise, Mirage returns the NotAuthenticated fault. The client must not log in again if its session is still valid. Otherwise, it will get the InvalidLogin fault. The user must have the Administrator, Web Help Desk, or Web Protection Manager role to log in successfully. For more information about roles, see the Managing Role-Based Access Control and Active Directory Groups section in the *VMware Mirage Administrator's Guide*.

Input:

- username
- password

Return:

- ServerInformation

Note If you update the Mirage API server to version 5.3 or later, you must update the client proxy.

Logout

Logout logs off the current user and invalidates the session.

Input:

None

Return:

None

Policy_Query

Policy_Query queries policies.

Input:

- queryDefinition
  - Filter

Field	Type	Description
POLICY_ID	Id	Id of the policy.
POLICY_NAME	string	The name of the policy.
POLICY_IMAGEID	ImageId	ImageId of the policy.

  - Page

    Starts at 1.

Return:

- QueryResult

  The type of element is PolicyDetails.

Example:

```
QueryDefinition queryDefinition = new QueryDefinition
{
    Filter = new QueryFilterBeginsWith
    {
        Field = FilterField.POLICY_NAME,
        Value = "VMware Mirage default CVD policy"
    },
    Page = 1 // Page starts from 1, not 0
};
QueryResult result = Client.Policy_Query(queryDefinition);
```

Volume_Query

Volume_Query queries a volume.

Input:

- queryDefinition
  - Filter

Field	Type	Description
VOLUME_ID	Id	Id of the volume.
VOLUME_NAME	string	The name of the volume.
VOLUME_PATH	string	The path of the volume.

  - Page

    Starts at 1.

Return:

- QueryResult

  The type of element is VolumeDetails.

Example:

```
QueryDefinition queryDefinition = new QueryDefinition
{
    Filter = new QueryFilterEquals
    {
        Field = FilterField.VOLUME_PATH,
        Value = @"C:\MirageStorage"
    },
    Page = 1 // Page starts from 1, not 0
};
QueryResult result = Client.Volume_Query(queryDefinition);
```

PendingDevice_Query

PendingDevice_Query queries pending devices.

Input:

- queryDefinition
  - Filter

Field	Type	Description
DEVICE_CONNECTION_STATE	bool	The connection state of the device.
DEVICE_ID	Id	The Id of the device.
DEVICE_MODEL_NAME	string	The model name of the device.
DEVICE_NAME	string	The name of the device.
DEVICE_OS	string	The OS version of the device.
DEVICE_OS_VERSION (Deprecated)	OsVersion	The OS version of the device.
DEVICE_USER_NAME	string	The user name of the device.
DEVICE_VENDOR_NAME	string	The vendor name of the device.

The DEVICE_OS filter field supports the following string filter values:

Value	Description
"XP"	Windows XP
"XP_EMBEDDED"	Windows XP Embedded
"VISTA"	Windows Vista x86
"VISTAX64"	Windows Vista x64
"WIN7"	Windows 7 x86
"WIN7_EMBEDDED"	Windows 7 x86 Embedded
"WIN7X64"	Windows 7 x64
"WIN7_EMBEDDEDX64"	Windows 7 x64 Embedded
"WIN8_0"	Windows 8 x86
"WIN8_0X64"	Windows 8 x64
"WIN8_1"	Windows 8.1 x86
"WIN8_1X64"	Windows 8.1 x64

Value	Description
"WINPE"	Windows PE 5
"Win10 x64"	Windows 10 x64
"Win10"	Windows 10
"Win Server 2012 R2"	Windows Server 2012 R2

- Page

  Starts at 1.

Return:

- QueryResult

  The type of element is DeviceDetails.

Example:

```
QueryDefinition queryDefinition = new QueryDefinition
{
    Filter = new QueryFilterEquals
    {
        Field = FilterField.DEVICE_OS_VERSION,
        Value = OsVersion.WIN7X64
    },
    Page = 1 // Page starts from 1, not 0
};
QueryResult queryResult = Client.PendingDevice_Query(queryDefinition);
```

Cvd_Query

Cvd_Query queries CVDs.

Input:

- queryDefinition
  - Filter

Field	Type	Description
CVD_CLIENT_STATE	string	The client state of the CVD.
CVD_DEVICE_CLIENT_STATUS (Deprecated)	ClientState	To filter by client state, see CVD_CLIENT_STATE.
CVD_DEVICE_CONNECTION_STATE	bool	The connection state of the CVD.
CVD_DEVICE_ID	Id	The device Id of the CVD.
CVD_DEVICE_OS	string	The device OS version of the CVD.
CVD_ID	Id	The Id of the CVD.
CVD_MACHINE_VERSION	ImageVersion	The machine version of the CVD.
CVD_NAME	string	The name of the CVD.
CVD_POLICY_ID	Id	The policy Id of the CVD.
CVD_POLICY_IMAGEID	ImageId	The policy ImageId of the CVD.
CVD_POLICY_NAME	string	The policy name of the CVD.

Field	Type	Description
CVD_PROGRESS	long	The operation progress of the CVD.
CVD_USER_NAME	string	The user name of the CVD.

The CVD_CLIENT_STATE filter field supports the following string filter values:

Value	Description
"Idle"	Idle state.
"PendingRebbot"	Pending reboot state.
"ForceReboot"	Force reboot state.
"UploadInitializing"	Upload initializing state.
"Uploading"	Uploading state.
"RestorePrefetch"	Restore pre-fetching states.
"RestoreStreaming"	Restore streaming states.
"UpdateLayers"	Downloading image states.
"RebasePrefetching"	Restore and base layer update pre-fetching states.
"Migration"	Migration states.
"DriverLibraryUpdate"	Driver library updating states.
"RestoreUserData"	Restore and profile pre-fetching states.
"DeviceProvisioning"	Device provisioning states.
"AppLayerRecording"	App layer recording states.
"AppLayerCapture"	App layer capture states.
"Synchronizing"	Synchronizing states.
"PendingUpgrade"	Pending upgrade states.
"PendingRestore"	Pending restore states.
"PendingAssignment"	Pending assignment states.
"Suspended"	Suspended states.
"WaitingForService"	Waiting for service states.
"Throttled"	Throttled states.
"PendingUserAuthentication"	Pending user authentication states.
"Snoozed"	Snoozed states.
"BranchReflectorIsBusy"	Branch reflector is busy states.
"WaitingDeviceProvisioning"	Waiting for device provisioning states.
"RecordingAppLayer"	Recording app ;ayer states.
"NoPotentialBranchReflectors"	No Potential Branch Reflectors states.
"BranchReflectorIsCaching"	Branch Reflector Is Caching states.
"PendingGatewayLogin"	Pending authentication from Gateway states.
"PendingRebootStreaming"	Pending reboot for restore streaming states.
"RestoreInitializing"	Restore initializing states.
"UpdateLayersInitializing"	Download image initializing states.
"RebasePrefetchInitializing"	Restore and base layer update initializing states.

Value	Description
"MigrationInitializing"	Migration initializing states.
"DriverLibraryUpdateInitializing"	Driver library update initializing states.
"RestoreUserDataInitializing"	Restore user data initializing states.
"DeviceProvisoningInitializing"	Device provisioning initializing states.
"AppLayerRecordingInitializing"	App layer recording initializing states.
"AppLayerCaptureInitializing"	App layer capture initializing states.
"SynchronizingInitializing"	Synchronize initializing states.
"RecordingAppLayerInitializing"	Record app layer initializing states.
"RestoreStreamingInitializing"	Restore streaming initializing states.
"UploadChangesFinalizing"	Upload changes finalizing states.
"RestorePrefetchFinalizing"	Restore prefetching finalizing states.
"UpdateLayersFinalizing"	Download image finalizing states.
"RebasePrefetchFinalizing"	Restore and base layer update prefetching finalizing states.
"MigrationFinalizing"	Migration finalizing states.
"DriverLibraryUpdateFinalizing"	Driver library update finalizing states.
"RestoreUserDataFinalizing"	Restore user data finalizing states.
"DeviceProvisioningFinalizing"	Device provisioning finalizing states.
"AppLayerRecordingFinalizing"	App layer recording finalizing states.
"AppLayerCaptureFinalizing"	App layer capture finalizing states.
"RecordingAppLayerFinalizing"	Recording App Layer finalizing states.
"RestoreStreamingFinalizing"	Restore streaming finalizing states.
"SynchronizingFinalizing"	Synchronizing finalizing states.

The CVD_DEVICE_OS filter field supports the following string filter values:

Value	Description
"XP"	Windows XP
"XP_EMBEDDED"	Windows XP Embedded
"VISTA"	Windows Vista x86
"VISTAX64"	Windows Vista x64
"WIN7"	Windows 7 x86
"WIN7_EMBEDDED"	Windows 7 x86 Embedded
"WIN7X64"	Windows 7 x64
"WIN7X64_EMBEDDED"	Windows 7 x64 Embedded
"WIN8_0"	Windows 8 x86
"WIN8_0X64"	Windows 8 x64
"WIN8_1"	Windows 8.1 x86
"WIN8_1X64"	Windows 8.1 x64
"WINPE"	Windows PE 5
"Win10 x64"	Windows 10 x64

Value	Description
"Win10"	Windows 10
"Win Server 2012 R2"	Windows Server 2012 R2

- Page

  Starts at 1.

Return:

- QueryResult

  The type of element is CvdDetails.

Example:

```
QueryDefinition queryDefinition = new QueryDefinition
{
    Filter = new QueryFilterEquals
    {
        Field = FilterField.CVD_DEVICE_CONNECTION_STATE,
        Value = true
    },
    Page = 1 // Page starts from 1, not 0
};
QueryResult queryResult = Client.Cvd_Query(queryDefinition);
```

Cvd_Get

Cvd_Get gets a CVD by Id.

Input:

- Id

  The Id of the CVD.

Return:

- CvdDetails

  The details of the CVD.

Example:

```
Id id = new Id
{
    IdValue = 10024
};
CvdDetails cvdDetails = Client.Cvd_Get(id);
```

Note: If the queried Id does not exist, Cvd_Get will return an InvalidArgument fault.

BaseLayer_Query

BaseLayer_Query queries base layers from Mirage management server.

Input:

- queryDefinition
  - Filter

Field	Type	Description
BASE_IMAGE_LAYER_NAME	string	The name of the base layer.
BASE_IMAGE_LAYER_ID	Id	The Id of the ImageId of the base layer.
BASE_IMAGE_LAYER_IMAGEID	ImageId	The ImageId of the base layer.

  - Page

    Starts at 1.

Return:

- QueryResult

  The type of element is LayerDetails.

Example:

```
QueryDefinition queryDefinition = new QueryDefinition
{
    Filter = new QueryFilterEquals
    {
        Field = FilterField.BASE_IMAGE_LAYER_NAME,
        Value = "Windows 7 x64"
    },
    Page = 1 // Page starts from 1, not 0
};
QueryResult result = Client.BaseLayer_Query(queryDefinition);
```

AppLayer_Query

AppLayer_Query queries app layers from Mirage management server.

Input:

- queryDefinition
  - Filter

Field	Type	Description
BASE_IMAGE_LAYER_NAME	string	The name of the app layer.
BASE_IMAGE_LAYER_ID	Id	The Id of the ImageId of the app layer.
BASE_IMAGE_LAYER_IMAGEID	ImageId	The ImageId of the app layer.

  - Page

    Starts at 1.

Return:

- QueryResult

  The type of element is LayerDetails.

Example:

```
QueryDefinition queryDefinition = new QueryDefinition
{
    Filter = new QueryFilterEquals
    {
       Field = FilterField.BASE_IMAGE_LAYER_NAME,
       Value = "VMware View Agent-5.3.0-EN"
    },
    Page = 1 // Page starts from 1, not 0
};
QueryResult result = Client.AppLayer_Query(queryDefinition);
```

PendingDevice_CreateNewCvd

PendingDevice_CreateNewCvd creates new CVDs from existing pending devices.

Input:

- deviceIds

  A list of pending device Ids. These Ids can be queried from PendingDevice_Query. If there are invalid Ids in the list, they will be skipped and faults will be set in BatchResult. New CVDs will be created for devices whose Id is valid.

- policyId

  The imageId of a policy.

- volumeId

  The Id of the volume that is used to store the CVDs. If this parameter is null, the system automatically selects a volume to store the CVD.

Return:

- BatchResult

  For each CVD, BatchResult has an OperationResult, which presents the result of creating the CVD. If OperationResult's Success is true, it means that Mirage starts to create the CVD, and Result is the IdValue of the Id of the new CVD. Otherwise, check OperationResult's MethodFault to get the error message that indicates why Mirage failed to create the CVD.

Example:

```
DeviceDetails[] deviceDetailsArr = GetDeviceDetailsArr();
Id[] deviceIds = deviceDetailsArr.Select(deviceDetails => deviceDetails.Id).ToArray();
PolicyDetails policyDetails = GetPolicyDetails();
VolumeDetails volumeDetails = GetVolumeDetails();
BatchResult result = client.PendingDevice_CreateNewCvd(cvdIds, policyDetails.ImageId,
volumeDetails.Id);
```

OsMigrationBegin

OsMigration_Begin starts a migration.

Input:

- migrationTargets

An array of MigrationTarget, which contains information for migration, such as Id of CVD, domain name, user, password, and so on. This method validates migration targets first, then starts to download the base layer and app layers. then it migrates. If some migration target fail to validate for some reasons, such as incorrect domain name, invalid CVD Id, and so on, they will be skipped, and other migration targets will continue to be processed.

- baseLayerId

  The imageId of the base layer.

- appLayerIds

  The ImageIds of app layers.

- ignoreWarnings

  Whether to ignore validation warnings. When ignoreWarnings is true, migration will start even if there are validation warnings. Otherwise, migration will not start.

Return:

- BatchResult

  For each CVD, BatchResult has an OperationResult, which presents the result of calling this method. If OperationResult's Success is true, it means that the migration is started. Otherwise, check OperationResult's MethodFault to get the error message.

Example:

```
LayerDetails baseLayer = GetBaseLayer();
LayerDetails[] appLayers = GetAppLayers();
MigrationTarget migrationTarget = new MigrationTarget
{
    CvdId = cvd.Id,
    IdentityInfo = new MachineIdentityInfo
    {
        DomainMember = true,
        DomainOrWorkgroupName = "mydomain.com",
        User = "bob",
        Password = "password"
    }
};
client.OsMigration_Begin(new MigrationTarget[] { migrationTarget }, baseLayer.ImageId,
appLayers.Select(appLayer => appLayer.ImageId).ToArray(), true /* ignore warnings */);
```

OsMigration_BeginDownloadOnly

OsMigration_BeginDownloadOnly starts to download the base layer and the app layers for a migration.

Input:

- migrationTargets

  An array of MigrationTargets, which contains information for migration, such as the Id of CVD, domain name, user, password, and so on. This method validates the migration targets first, and then starts to download the base layer and app layers. Validation includes domain name, CVD Id, and so on. If some migration targets fail to validate, they will be skipped, and other migration targets will continue to be processed.

- baseLayerId

  The imageId of the base layer.

- appLayerIds

  The ImageIds of app layers.

- ignoreWarnings

  Whether to ignore validation warnings. When ignoreWarnings is true, migration will start even if there are validation warnings. Otherwise, migration will not start.

Return:

- BatchResult

  For each CVD, BatchResult has an OperationResult, which presents the result of calling this method. When OperationResult's Success is true, it means that Mirage starts to download the base layer and app layers. Otherwise, check OperationResult's MethodFault to get the error message.

Example:

```
LayerDetails baseLayer = GetBaseLayer();
LayerDetails[] appLayers = GetAppLayers();
MigrationTarget migrationTarget = new MigrationTarget
{
    CvdId = cvd.Id,
    IdentityInfo = new MachineIdentityInfo
    {
        DomainMember = true,
        DomainOrWorkgroupName = "mydomain.com",
        User = "bob",
        Password = "password"
    }
};
client.OsMigration_BeginDownloadOnly(new MigrationTarget[] { migrationTarget },
baseLayer.ImageId, appLayers.Select(appLayer => appLayer.ImageId).ToArray(), true /* ignore
warnings */);
```

OsMigration_ApplyDownloadOnlyMigration

OsMigration_ApplyDownloadOnlyMigration applies the downloaded base layer and app layers.

Input:

- CvdIds

  A list of CVD Ids that are to be migrated. This method validates the cvdIds first and then starts to apply migration. If some CVDs are not valid for applying migration, they are skipped while other CVDs will migrate.

Return:

- BatchResult

  For each CVD, BatchResult has an OperationResult that presents the result of calling this method. When OperationResult's Success is true, it means that migration starts successfully. Otherwise, it means that migration fails.

Example:

```
Id[] cvdIds = GetCvdIds();
BatchResult results = client.OsMigration_ApplyDownloadOnlyMigration(cvdIds);
```

OsMigration_QueryDownloadOnlyInProgress (Deprecated)

OsMigration_QueryDownloadOnlyInProgress queries CVDs which are downloading base and app layers.

Input:

- queryDefinition
  - Filter

    Must be null.
  - Page

    Starts at 1.

Return:

- QueryResult

  The type of element is CvdDetails.

Example:

```
QueryDefinition queryDefinition = new QueryDefinition
{
    Filter = null,
    Page = 1 //Page starts from 1, not 0
};
QueryResult queryResult = Client.OsMigration_QueryDownloadOnlyInProgress(queryDefinition);
```

Note: If the filter is not null, OsMigration_QueryDownloadOnlyInProgress will throw the NotSupportedFault exception.

OsMigration_QueryDownloadOnlyCompleted (Deprecated)

OsMigration_QueryDownloadOnlyCompleted queries CVDs which have finished downloading base and app layers.

Input:

- queryDefinition
  - Filter

    Must be null.
  - Page

    Starts at 1.

Return:

- QueryResult

  The type of element is CvdDetails.

Example:

```
QueryDefinition queryDefinition = new QueryDefinition
{
    Filter = null,
    Page = 1 //Page starts from 1, not 0
};
QueryResult queryResult = Client.OsMigration_QueryDownloadOnlyCompleted(queryDefinition);
```

Note: If the filter is not null, OsMigration_QueryDownloadOnlyCompleted will throw the NotSupportedFault exception.

Cvd_Archive

Cvd_Archive archives the CVDs in the Mirage Management system.

Input:

- cvdIds

The ids of the requested CVDs, The list should not contain more than 500 entries. When there are invalid CVDs in the list, this method only archives the valid CVD Ids. Invalid CVD Ids are skipped.

Return:

- BatchResult

  For each CVD, BatchResult has an OperationResult, which presents the result of archiving the CVD. When OperationResult's Success is true, it means the Mirage starts to archive the CVD, and the Result is the IdValue of Id of the CVD. Otherwise, please check OperationResult's MethodFault to get message why it is failed to archive the CVD.

Example:

```
QueryDefinition queryDefinition = new QueryDefinition
        {
              Filter = new QueryFilterEquals
              {
                    Field = FilterField.CVD_USER_NAME,
                    Value = "ObsoleteUser"
              },
              Page = 1
        };

        QueryResult cvdQuery = client.Cvd_Query(queryDefinition);

        int length = cvdQuery.Elements.Length;
        if (length <= 0)
            return;

        Id[] ids = new Id[length];
        for (int i = 0; i < length; ++i)
        {
            ids[i] = ((CvdDetails)cvdQuery.Elements[i]).Id;
        }

        BatchResult batchResult = client.Cvd_Archive(ids);
        for(int i = 0; i < length; ++i)
        {
            OperationResult opResult = batchResult.results[i];
```

```
           if (!opResult.Success)
                Console.WriteLine("failed to archive CVD {0}, error: {1}", ids[i].IdValue,
opResult.Fault.Message);
        }
```

Cvd_Delete

Cvd_Delete deletes the CVDs in the Mirage Management system.

Input:

- cvdIds

  The ids of the requested CVDs. The list should not contain more than 500 entries. When there are invalid CVDs in the list, this method only deletes the valid CVD Ids. Invalid CVD Ids are skipped.

Return:

- BatchResult

  For each CVD, BatchResult has an OperationResult, which presents the result of deleting the CVD. When OperationResult's Success is true, it means the Mirage starts to delete the CVD, and the Result is the IdValue of Id of the CVD. Otherwise, please check OperationResult's MethodFault to get message why it is failed to delete the CVD.

Example:

```
QueryDefinition queryDefinition = new QueryDefinition
        {
              Filter = new QueryFilterEquals
              {
                    Field = FilterField.CVD_USER_NAME,
                    Value = "ObsoleteUser"
              },
              Page = 1
        };

        QueryResult cvdQuery = client.Cvd_Query(queryDefinition);

        int length = cvdQuery.Elements.Length;
        if (length <= 0)
            return;

        Id[] ids = new Id[length];
        for (int i = 0; i < length; ++i)
        {
            ids[i] = ((CvdDetails)cvdQuery.Elements[i]).Id;
        }

        BatchResult batchResult = client.Cvd_Delete(ids);
        for(int i = 0; i < length; ++i)
        {
            OperationResult opResult = batchResult.results[i];
           if (!opResult.Success)
                Console.WriteLine("failed to delete CVD {0}, error: {1}", ids[i].IdValue,
opResult.Fault.Message);
        }
```

Cvd_Sync

Cvd_Sync synchronizes the device information of the CVDs in the Mirage Management system.

Input:

- cvdIds

  The ids of the requested CVDs. The list should not contain more than 500 entries. When there are invalid CVDs in the list, this method only synchronizes the valid CVD Ids. Invalid CVD Ids are skipped.

Return:

- BatchResult

  For each CVD, BatchResult has an OperationResult, which presents the result of synchronizing the CVD. When OperationResult's Success is true, it means the Mirage starts to synchronize the CVD, and the Result is the IdValue of Id of the CVD. Otherwise, please check OperationResult's MethodFault to get message why it is failed to synchronize the CVD.

Example:

```
QueryDefinition queryDefinition = new QueryDefinition
        {
              Filter = new QueryFilterEquals
              {
                    Field = FilterField.CVD_USER_NAME,
                    Value = "John"
              },
              Page = 1
        };

        QueryResult cvdQuery = client.Cvd_Query(queryDefinition);

        int length = cvdQuery.Elements.Length;
        if (length <= 0)
            return;

        Id[] ids = new Id[length];
        for (int i = 0; i < length; ++i)
        {
            ids[i] = ((CvdDetails)cvdQuery.Elements[i]).Id;
        }

        BatchResult batchResult = client.Cvd_Sync(ids);
        for(int i = 0; i < length; ++i)
        {
            OperationResult opResult = batchResult.results[i];
           if (!opResult.Success)
                Console.WriteLine("failed to sync CVD {0}, error: {1}", ids[i].IdValue,
opResult.Fault.Message);
        }
```

Cvd_ApplyPolicy

Cvd_ApplyPolicy applies a policy to the given CVDs.

Input:

- cvdIds When there

  The ids of the requested CVDs. The list should not contain more than 500 entries. When there are invalid CVDs in the list, this method only applies a policy for valid CVD Ids. Invalid CVD Ids are skipped.

- policyId

  The ImageId of the target policy.

Return Value:

- BatchResult

  For each CVD, BatchResult has an OperationResult, which presents the result of applying the CVD policies. When OperationResult's Success is true, it means the policy of CVD has been applied, and the Result is the IdValue of Id of the CVD. Otherwise, please check OperationResult's MethodFault to get message why it is failed to apply the CVD policies.

Example:

```
QueryDefinition queryDefinition = new QueryDefinition
        {
            Page = 1
        };
        QueryResult queryResult = Client.Cvd_Query(queryDefinition);
        CvdDetails[] cvds = queryResult.Elements.Cast<CvdDetails>().ToArray();
        if (!cvds.Any())
        {
            return;
        }
        Id[] ids = cvds.Select(cvd => cvd.Id).ToArray();
        int length = ids.Length;

        queryDefinition = new QueryDefinition
        {
            Filter = new QueryFilterEquals
            {
                Field = FilterField.POLICY_NAME,
                Value = "Win7 Policy"
            },
            Page = 1
        };
        queryResult = Client.Policy_Query(queryDefinition);
        PolicyDetails[] policies = queryResult.Elements.Cast<PolicyDetails>().ToArray();
        if (!policies.Any())
        {
            return;
        }

        BatchResult result = Client.Cvd_ApplyPolicy(ids, policies[0].ImageId);

        for (int i = 0; i < length; i++)
        {
            OperationResult opResult = result.results[i];
            if (opResult.Success == false)
            {
```

```
                Console.WriteLine("failed to apply policy for Cvd {0}, error: {1}",
ids[i].IdValue, opResult.Fault.Message);
            }
        }
```

PendingDevice_Provision

PendingDevice_Provision provisions the devices with a base layer and app layers.

Input:

- pendingDevices

  pendindDevices is a list of Ids of pending devices. The list should not contain more than 500 entries. When there are invalid devices in the list, this method only provisions the valid device Ids. Invalid device Ids are skipped.

- policyImageId

  The policy ImageId that is applied.

- baseLayerImageId

  The base layer ImageId that is used to provision the devices.

- appLayerImageIds

  The app layer ImageId that is used to provision the devices.

- identityInfo

  The machine identity used to join domain or workgroup.

- volumeId

  The volume Id in which the CVD is stored. If this parameter is null, the system automatically selects a volume to store the CVD.

- ignoreWarnings

  Ignore validation warnings or not. When ignoreWarnings is true, provisioning will start even if there are warnings of validation, otherwise provisioning will not start when there is a warning.

Return Value:

- BatchResult

  For each CVD, BatchResult has an OperationResult, which presents the result of provision the device. When OperationResult's Success is true, it means the Mirage starts to provision the device, and the Result is the IdValue of Id of the new CVD. Otherwise, please check OperationResult's MethodFault to get message why it is failed to provision the device.

Example:

```
QueryResult pendingDeviceQuery = client.PendingDevice_Query(new QueryDefinition { Page = 1 });

        int length = pendingDeviceQuery.Elements.Length;
        if (length <= 0)
            return;

        QueryResult baseLayerQuery = client.BaseLayer_Query(new QueryDefinition
        {
            Filter = new QueryFilterEquals
            {
                Field = FilterField.BASE_IMAGE_LAYER_IMAGEID,
```

```
                Value = new ImageId
                {
                    Id = new Id
                    {
                        IdValue = 1
                    },
                    Version = new ImageVersion
                    {
                        Major = 1,
                        Minor = 0
                    }
                }
                },
            Page = 1
        });

        if (baseLayerQuery.Elements.Length <= 0)
            return;

        LayerDetails baseLayer = (LayerDetails)baseLayerQuery.Elements[0];

        QueryResult appLayerQuery = client.AppLayer_Query(new QueryDefinition {Page = 1});
        ImageId[] appLayers = null;
        if (appLayerQuery.Elements.Length > 0)
            appLayers = appLayerQuery.Elements.Select(layer =>
((LayerDetails)layer).ImageId).ToArray();

        QueryResult policyQuery = client.Policy_Query(new QueryDefinition { Page = 1 });

        if (policyQuery.Elements.Length <= 0)
            return;

        PolicyDetails policy = (PolicyDetails)policyQuery.Elements[0];

        QueryResult volumeQuery = client.Volume_Query(new QueryDefinition { Page = 1 });

        if (volumeQuery.Elements.Length <= 0)
            return;

        VolumeDetails volume = (VolumeDetails)volumeQuery.Elements[0];

        Id[] ids = new Id[length];
        for (int i = 0; i < length; ++i)
            ids[i] = ((DeviceDetails) pendingDeviceQuery.Elements[i]).Id;

        MachineIdentityInfo identityInfo = new MachineIdentityInfo
        {
            DomainMember = true,
            DomainOrWorkgroupName = "domainName",
            User = "user",
            Password = "password"
        };
        BatchResult batchResult = client.PendingDevice_Provision(ids, policy.ImageId,
baseLayer.ImageId, appLayers, identityInfo, volume.Id, false);
        for(int i = 0; i < length; ++i)
```

```
        {
            OperationResult opResult = batchResult.results[i];
            if (!opResult.Success)
            {
                Console.WriteLine("failed to provision device {0}, error: {1}", ids[i].IdValue,
opResult.Fault.Message);
            }
        }
```

Collection_Query

Collection_Query queries all CVD collections in the Mirage Management system.

Input:

- queryDefinition
  - Filter

Filter	Type	Description
COLLECTION_DESCRIPTION	string	The description of the collection.
COLLECTION_ID	Id	The Id of the collection.
COLLECTION_NAME	string	The name of the collection.

  - Page

    Starts at 1.

Return Value:

- QueryResult

  The type of element is CollectionDetails.

Example:

```
QueryResult queryResult = Client.Collection_Query(new QueryDefinition{Filter = null, Page = 1});
    if (!queryResult.Elements.Any())
    {
        return;
    }
    CollectionDetails[] collections = queryResult.Elements.Cast<CollectionDetails>().ToArray();
    // do work with collections
```

CollectionCvd_Query

Queries all CVDs in the specific collection in the Mirage Management system.

Input:

- collectionId

  The Id of the collection.

- queryDefinition
  - Filter

Field	Type	Description
CVD_CLIENT_STATE	string	The client state of the CVD.
CVD_DEVICE_CLIENT_STATUS (Deprecated)	ClientState	To filter with client state, see CVD_CLIENT_STATE.
CVD_DEVICE_CONNECTION_STATE	bool	The connection state of the CVD.
CVD_DEVICE_ID	Id	The device Id of the CVD.
CVD_DEVICE_OS	string	The device OS version of the CVD.
CVD_ID	Id	The Id of the CVD.
CVD_MACHINE_VERSION	ImageVersion	The machine version of the CVD.
CVD_NAME	string	The name of the CVD.
CVD_POLICY_ID	Id	The policy Id of the CVD.
CVD_POLICY_IMAGEID	ImageId	The policy ImageId of the CVD.
CVD_POLICY_NAME	string	The policy name of the CVD.
CVD_PROGRESS	long	The operation progress of the CVD.
CVD_USER_NAME	string	The user name of the CVD.

The CVD_CLIENT_STATE filter field supports the following string filter values:

Value	Description
"Idle"	Idle state.
"PendingReboot"	Pending reboot state.
"ForceReboot"	Force reboot state.
"UploadInitializing"	Upload initializing state.
"RestorePrefetch"	Restore prefetching states.
"RestoreStreaming"	Restore streaming states.
"UpdateLayers"	Downloading image states.
"RebasePrefetching"	Restore and base layer update prefetching states.
"Migration"	Migration states.
"DriverLibraryUpdate"	Driver library updating states.
"RestoreUserData"	Restore and profile prefetching states.
"DeviceProvisioning"	Device provisioning states.
"AppLayerCapture"	App layer capture states.
"Synchronizing"	Synchronizing states.
"PendingUpgrade"	Pending upgrade states.
"PendingRestore"	Pending restore states.
"PendingAssignment"	Pending assignment states.
"Suspended"	Suspended states.
"WaitingForService"	Waiting for service states.
"Throttled"	Throttled states.

Value	Description
"PendingUserAuthentication"	Pending user authentication states.
"Snoozed"	Snoozed states.
"BranchReflectorIsBusy"	Branch reflector is busy states.
"WaitingDeviceProvisioning"	Waiting for device provision states.
"RecordingAppLayer"	Recording app layer states.
"NoPotentialBranchReflectors"	No potential branch reflectors states.
"BranchReflectorIsCaching"	Branch reflector is caching states.
"PendingGatewayLogin"	Pending authentication from Gateway states.
"PendingRebootStreaming"	Pending reboot for restore streaming states.
"RestoreInitializing"	Restore initializing states.
"UpdateLayersInitializing"	Download image initializing states.
"RebasePrefetchInitializing"	Restore and base layer update initializing states.
"MigrationInitializing"	Migration initializing states.
"DriverLibraryUpdateInitializing"	Driver library update initializing states.
"RestoreUserDataInitializing"	Restore user data initializing states.
"DeviceProvisoningInitializing"	Device provisioning initializing states.
"AppLayerRecordingInitializing"	App layer recording initializing states.
"AppLayerCaptureInitializing"	App layer capture initializing states.
"SynchronizingInitializing"	Synchronizing initializing states.
"RecordingAppLayerInitializing"	Record app layer initializing states.
"RestoreStreamingInitializing"	Restore streaming initializing states.
"UploadChangesFinalizing"	Upload changes finalizing states.
"RestorePrefetchFinalizing"	Restore prefetch finalizing states.
"UpdateLayersFinalizing"	Download image finalizing states.
"RebasePrefetchFinalizing"	Restore and base layer update prefetch finalizing states.
"MigrationFinalizing"	Migration finalizing states.
"DriverLibraryUpdateFinalizing"	Driver library update finalizing states.
"RestoreUserDataFinalizing"	Restore user data finalizing states.
"DeviceProvisioningFinalizing"	Device provisioning finalizing states.
"AppLayerRecordingFinalizing"	App layer recording finalizing states.
"AppLayerCaptureFinalizing"	App layer capture finalizing states.
"RecordingAppLayerFinalizing"	Recording app layer finalizing states.
"RestoreStreamingFinalizing"	Restore streaming finalizing states.
"SynchronizingFinalizing"	Synchronizing finalizing states.

The CVD_DEVICE_OS filter field supports the following string filter values:

Value	Description
"XP"	Windows XP
"XP_EMBEDDED"	Windows XP Embedded

Value	Description
"VISTA"	Windows Vista x86
"VISTAX64"	Windows Vista x64
"WIN7"	Windows 7 x86
"WIN7_EMBEDDED"	Windows 7 x86 Embedded
"WIN7X64"	Windows 7 x64
"WIN7_EMBEDDEDX64"	Windows 7 x64 Embedded
"WIN8_0"	Windows 8 x86
"WIN8_0X64"	Windows 8 x64
"WIN8_1"	Windows 8.1 x86
"WIN8_1X64"	Windows 8.1 x64
"WINPE"	Windows PE 5
"Win10 x64"	Windows 10 x64
"Win10"	Windows 10
"Win Server 2012 R2"	Windows Server 2012 R2

- Page

  Starts at 1.

Return:

- QueryResult

  The type of element is CvdDetails.

Example:

```
QueryResult queryResult = Client.Collection_Query(new QueryDefinition
{
    Filter = new QueryFilterEquals
    {
        Field = FilterField.COLLECTION_NAME,
        Value = "static collection"
    },
    Page = 1
});
if (queryResult == null || queryResult.Elements == null || !queryResult.Elements.Any())
{
    return;
}
CollectionDetails collection = queryResult.Elements.Cast<CollectionDetails>().ToArray()[0];

queryResult = Client.CollectionCvd_Query(collection.Id, new QueryDefinition { Page = 1 });
if (queryResult != null && queryResult.Elements != null)
{
    if (!queryResult.Elements.Any())
    {
        return;
    }

    CvdDetails[] cvds = queryResult.Elements.Cast<CvdDetails>().ToArray();
    foreach (CvdDetails cvdDetails in cvds)
```

```
        {
            Console.WriteLine("ID: {0}, device ID: {1}, policy ID: {2}, machine version: {3}.
{4}",
                cvdDetails.Id.IdValue, cvdDetails.DeviceId.IdValue, cvdDetails.PolicyId.IdValue,
                cvdDetails.MachineVersion.Major, cvdDetails.MachineVersion.Minor);
        }
    }
```

OsMigrationCvd_QueryDownloadOnly

OsMigrationCvd_QueryDownloadOnly queries the CVD Ids for download only migration.

Input:

- QueryDefinition
  - Filter

Field	Type	Description
CVD_ID	Id	The Id of the CVD.
DOWNLOAD_ONLY_MIGRATION_STATUS	string	The status of the download only migration

  - Page

    Starts at 1.

Return:

- QueryResult

  The type of element is CvdDownloadOnlyMigrationDetails.

Example:

```
int page = 1;
int total = 0;

while (true)
{
    QueryDefinition query = new QueryDefinition
    {
        Filter = new QueryFilterEquals
        {
            Field = FilterField.DOWNLOAD_ONLY_MIGRATION_STATUS,
            Value = "Failed"
        },
        Page = page
    };
    QueryResult result = Client.MigrationCvd_QueryDownloadOnly(query);

    if (result != null && result.Elements != null)
    {
        if (result.Elements.Length == 0)
        {
            return;
        }

        foreach (CvdDownloadOnlyMigrationDetails migration in result.Elements)
```

```
            {
                Console.WriteLine("ID: {0}, Status: {1}", migration.CvdId.IdValue,
migration.DownloadOnlyMigrationStatus);
            }
            total += result.Elements.Length;
            page ++;
            continue;
        }
        break;
    }
    Console.WriteLine("Total number: {0}", total);
```

Assignment_ApplyDownloadOnly

Assignment_ApplyDownloadOnly applies the downloaded assignments.

Input:

- assignmentIds
  - The id(s) of the assignment(s). The list should not contain more than 500 entries.
- onlyCompleted
  - Only apply the assignments that completed downloading or not.

Return:

- BatchResult
  - For each assignment, BatchResult has an OperationResult, which displays the result of applying assignment. If OperationResult's Success is true, Mirage applies the assignment, and the Result is the IdValue of Id of the assignment. If the Success is false, refer to the MethodFault of OperationResult. Use Assignment_Query to get detailed information of new generated assignments.

Example:

```
Id[] assignmentIds = GetDownloadOnlyAssignmentIds();
BatchResult results = Assignment_ApplyDownloadOnly(assignmentIds, false);
QueryFilter filter = new QueryFilterEquals
{
    Field = FilterField.ASSIGNMENT_TASK_NAME,
    Value = "ApplyLayers"
};
QueryResult queryResult = Client.Assignment_Query(new QueryDefinition { Filter = filter, Page =
1 });
//check detailed status of apply layers assignment.
```

Assignment_Query

Assignment_Query queries assignments in the Mirage Management system.

Input:

- queryDefinition
  - Filter

Field	Type	Description
ASSIGNMENT_CVD_ID	Id	The CVD Id of the assignment.
ASSIGNMENT_CVD_NAME	string	The CVD Name of the assignment.
ASSIGNMENT_ID	Id	The Id of the assignment.
ASSIGNMENT_LAYER_NAME	string	The layer name of the assignment.
ASSIGNMENT_STATUS	string	The status of the assignment.
ASSIGNMENT_TASK_NAME	string	The task name of the assignment.
ASSIGNMENT_TYPE	string	The type of the assignment.
ASSIGNMENT_USERNAME	string	The name of the user that initiates the assignment.

The ASSIGNMENT_STATUS filter field supports the following string filter values.

Value	Description
"Pending"	Pending status.
"Downloading"	Downloading status.
"Committing"	Committing status.
"Done"	Done status.
"Blocked"	Blocked status.
"Throttled"	Throttled status.
"Rejected"	Rejected status.
"Cancelled"	Canceled status.
"Total"	Total status.
"PendingBiCopy"	PendingBiCopy status.

The ASSIGNMENT_TASK_NAME filter field supports the following string filter values.

Value	Description
"AssignBaseImage"	AssignBaseImage task.
"Migration"	Migration task.
"DeviceProvisioning"	DeviceProvisioning task.
"AssignAppLayer"	AssignAppLayer task.
"Enforce"	Enforce task.
"RemoveApp"	RemoveApp task.
"MigrateDownloadOnly"	MigrateDownloadOnly task.
"BaseImageDownloadOnly"	BaseImageDownloadOnly task.
"AppLayerDownloadOnly"	AppLayerDownloadOnly task.
"ApplyLayers"	ApplyLayers task.

The ASSIGNMENT_TYPE filter field supports the following string filter values.

Value	Description
"AssignBaseLayer"	AssignBaseLayer type.
"RemoveBaseLayer"	RemoveBaseLayer type.
"MigrationBaseLayer"	MigrationBaseLayer type.
"DeviceProvisioningBaseLayer"	DeviceProvisioningBaseLayer type.
"DownloadOnlyBaseLayer"	DownloadOnlyBaseLayer type.
"DownloadOnlyRemoveBaseLayer"	DownloadOnlyRemoveBaseLayer type.
"MigrationBaseLayerDownloadOnly"	MigrationBaseLayerDownloadOnly type
"AssignAppLayer"	AssignAppLayer type.
"RemoveAppLayer"	RemoveAppLayer type.
"MigrationAppLayer"	MigrationAppLayer type.
"DeviceProvisioningAppLayer"	DeviceProvisioningAppLayer type.
"DownloadOnlyAppLayer"	DownloadOnlyAppLayer type.
"DownloadOnlyRemoveAppLayer"	DownloadOnlyRemoveAppLayer type
"MigrationAppLayerDownloadOnly"	MigrationAppLayerDownloadOnly type

- Page

  Starts at 1.

Return:

- QueryResult

  The type of element is AssignmentDetails.

Example:

```
QueryFilter filter = new QueryFilterEquals
{
    Field = FilterField.ASSIGNMENT_TYPE,
    Value = "DownloadOnlyBaseLayer"
};
QueryResult queryResult = Client.Assignment_Query(new QueryDefinition { Filter = filter, Page =
1 });
if (!queryResult.Elements.Any())
{
    return;
}
AssignmentDetails[] assignments = queryResult.Elements.Cast<AssignmentDetails>().ToArray();
```

Cvd_AssignBaseLayer

Cvd_AssignBaseLayer assigns a base layer to a CVD. This method first validates the CVD and base layer, then it assigns the base layer to the CVD.

Input:

- cvdIds
  - cvdIds is a list of the CVD Ids. The list should not contain more than 500 entries. In the event that there are invalid CVD Ids in the list, this method only processes the valid CVD Ids.

- baseLayerImageId
  - The base layer ImageId that is assigned to the CVD.
- ignoreWarnings
  - Ignore validation warnings or not. If ignoreWarnings is true, base layer assignment procedures proceed even if there are validation warnings. Otherwise, base layer assignment procedures do not proceed.

Return:

- BatchResult
  - For each CVD, BatchResult has an OperationResult, which displays the result of assigning a base layer to a CVD. If OperationResult's Success is true, Mirage assigns the base layer, and the Result is the IdValue of Id of the CVD. If the Success is false, refer to the MethodFault of OperationResult.

Example:

```
Id[] cvdIds = GetCvdIds();
ImageId baseLayerId = GetBaseLayerImageId();
Client.Cvd_AssignBaseLayer(cvdIds, baseLayerId, true);
```

Cvd_AssignBaseLayer creates the assignment of type "AssignBaseLayer". You can query assignment for detailed operation status.

Cvd_AssignBaseLayerDownloadOnly

Cvd_AssignBaseLayerDownloadOnly downloads a base layer for a CVD. This method first validates the CVD and base layer, then it downloads the base layer.

Input:

- cvdIds
  - cvdIds is a list of the CVD Ids. The list should not contain more than 500 entries. In the event that there are invalid CVD Ids in the list, this method only processes the valid CVD Ids.
- baseLayerImageId
  - The base layer ImageId that is assigned to the CVD.
- ignoreWarnings
  - Ignore validation warnings or not. If ignoreWarnings is true, base layer download procedures proceed even if there are validation warnings. If ignoreWarnings is false, base layer download procedures do not proceed.

Return:

- BatchResult
  - For each CVD, BatchResult has an OperationResult, which displays the result of updating app layers on a CVD. If the OperationResult's Success is true, Mirage downloads the base layer, and the Result is the IdValue of Id of the CVD. If the Success is false, refer to the MethodFault of OperationResult.

Example:

```
Id[] cvdIds = GetCvdIds();
ImageId baseLayerId = GetBaseLayerImageId();
Client.Cvd_AssignBaseLayerDownloadOnly(cvdIds, baseLayerId, true);
```

Cvd_AssignBaseLayerDownloadOnly creates the assignments of type "DownloadOnlyBaseLayer". You can query assignment for detailed operation status.

Cvd_UpdateAppLayer

Cvd_UpdateAppLayer updates app layers on a CVD. This method first validates the CVD and app layers, then it updates the app layers.

Input:

- cvdIds
  - cvdIds is a list of the CVD Ids. The list should not contain more than 500 entries. In the event that there are invalid CVD Ids in the list, this method only processes the valid CVD Ids.
- addAppLayerImageIds
  - The ImageId of the app layers that are assigned to the CVD.
- removeAppLayerImageIds
  - The ImageId of the app layers that are removed from the CVD.
- ignoreWarnings
  - Ignore validation warnings or not. If ignoreWarnings is true, app layers update procedures proceed even if there are validation warnings. If ignoreWarnings is false, app layers update procedures do not proceed.

Return:

- BatchResult
  - For each CVD, BatchResult has an OperationResult, which displays the result of updating app layers on a CVD. If OperationResult's Success is true, Mirage updates the app layers, and the Result is the IdValue of Id of the CVD. If Success is false, refer to the MethodFault of OperationResult.

Example:

```
Id[] cvdIds = GetCvdId();
ImageId[] addLayerIds = GetAddAppLayerImageIds();
ImageId[] removeLayerIds = GetRemoveAppLayerImageIds();

Client.Cvd_UpdateAppLayer(cvdIds, addLayerIds, removeLayerIds, true);
```

Cvd_UpdateAppLayer creates the assignments of type "AssignAppLayer" and "RemoveAppLayer". You can query assignment for detailed operation status.

Cvd_UpdateAppLayerDownloadOnly

Cvd_UpdateAppLayerDownloadOnly downloads app layers for a CVD. This method first validates the CVD and app layers, then it downloads the app layers.

Input:

- cvdIds
  - cvdIds is a list of the CVD Ids. The list should not contain more than 500 entries. In the event that there are invalid CVD Ids in the list, this method only processes the valid CVD Ids.
- addAppLayerImageIds
  - The ImageId of the app layers that are assigned to the CVD.
- removeAppLayerImageIds
  - The ImageId of the app layers that removed from the CVD.

- ignoreWarnings
  - Ignore validation warnings or not. If ignoreWarnings is true, app layers download procedures proceed even if there are validation warnings. If ignoreWarnings is false, app layers download procedures do not proceed.

Return:

- BatchResult
  - For each CVD, BatchResult has an OperationResult, which displays the result of downloading app layers on a CVD. If the OperationResult's Success is true, Mirage downloads the app layers, and the Result is the IdValue of Id of the CVD. If the Success is false, refer to the MethodFault of OperationResult.

Example:

```
Id[] cvdIds = GetCvdId();
ImageId[] addLayerIds = GetAddAppLayerImageIds();
ImageId[] removeLayerIds = GetRemoveAppLayerImageIds ();

Client.Cvd_UpdateAppLayerDownloadOnly(cvdIds, addLayerIds, removeLayerIds, true);
```

Cvd_UpdateAppLayerDownloadOnly creates the assignments of type "DownloadOnlyAppLayer" and "DownloadOnlyRemoveAppLayer". You can query assignment for detailed operation status.

PendingDevice_ProvisionWithProvisionTarget

PendingDevice_ProvisionWithProvisionTarget provisions the pending devices. It provides the ability to change each device's machine name and join domain.

Input:

- pendingDevices
  - pendingDevices is a list of the pending devices. Each element of the list contains the pending device ID and the desired machine name and domain information after provision.
- policyImageId
  - The policy ImageId that is applied.
- baseLayerImageId
  - The base layer ImageId that is used to provision the devices.
- appLayerImageIds
  - The app layer ImageId that is used to provision the devices.
- volumeId
  - The volume Id in which the CVD is stored. If this parameter is null, the system automatically selects a volume to store the CVD.
- ignoreWarnings
  - Ignore validation warnings or not. When ignoreWarnings is true, provisioning will start even if there are warnings of validation, otherwise provisioning will not start when there is a warning.

Return:

- BatchResult
  - For each CVD, BatchResult has an OperationResult, which displays the result of downloading app layers on a CVD. If the OperationResult's Success is true, Mirage downloads the app layers, and the Result is the IdValue of Id of the CVD. If the Success is false, refer to the MethodFault of OperationResult.

Example:

```
QueryResult pendingDeviceQuery = client.PendingDevice_Query(new QueryDefinition{ Page = 1 });

int length = pendingDeviceQuery.Elements.Length;
if (length < 0)
    return;

QueryResult baseLayerQuery = client.BaseLayer_Query(new QueryDefinition
{
    Filter = new QueryFilterEquals
    {
        Field = FilterField.BASE_IMAGE_LAYER_ID,
        Value = new ImageId
        {
            Id = new Id
            {
                IdValue = 1
            },
            Version = new ImageVersion
            {
                Major = 1,
                Minor = 0
            }
        }
    },
    Page = 1
});

if (baseLayerQuery.Elements.Length <= 0)
    return;

LayerDetails baseLayer = (LayerDetails)baseLayerQuery.Elements[0];

QueryResult appLayerQuery = client.AppLayer_Query(new QueryDefinition { Page = 1 });
ImageId[] appLayers = null;
if (appLayerQuery.Elements.Length > 0)
    appLayers = appLayerQuery.Elements.Select(layer => ((LayerDetails)layer).ImageId).ToArray();
QueryResult policyQuery = client.Policy_Query(new QueryDefinition { Page = 1 });
if (policyQuery.Elements.Length <= 0)
    return;

PolicyDetails policy = (PolicyDetails)policyQuery.Elements[0];

QueryResult volumeQuery = client.Volume_Query(new QueryDefinition { Page = 1 });
if (volumeQuery.Elements.Length <= 0)
    return;
VolumeDetails volume = (VolumeDetails)volumeQuery.Elements[0];

ProvisionTarget[] targets = new ProvisionTarget[length];
for (int i = 0; i < length; ++i)
{
    targets[i].PendingDeviceId = ((DeviceDetails)pendingDeviceQuery.Elements[i]).Id;
    targets[i].IdentityInfo = new MachineIdentityInfo
    {
        DomainMember = true,
```

```
        DomainOrWorkgroupName = "domainName",
        User = "user",
        Password = "password",
        MachineName = string.Format("Machine-{0}", i)
    };
}

BatchResult batchResult = client.PendingDevice_ProvisionWithProvisionTarget(targets,
policy.ImageId, baseLayer.ImageId, appLayers, volume.Id, false);
for (int i = 0; i < length; ++i)
{
    OperationResult opResult = batchResult.results[i];
    if (!opResult.Success)
    {
        Console.WriteLine("failed to provision device {0}, error: {1}",
targets[i].PendingDeviceId.IdValue,
        opResult.Fault.Message);
    }
}
```

Cvd_EnforceAllLayers

Cvd_EnforceAllLayers enforce all the layers on the CVDs.

Input:

- cvdIds
  - cvdIds is a list of the CVD ids. The list should not contain more than 500 entries. In the event that there are invalid CVD Ids in the list, this method only processes the valid CVD Ids.
- cleanUserData
  - cleanUserData indicates to remove the user's data.
- ignoreWarnings
  - ignoreWarnings used to ignore the validation warnings.

Return:

- BatchResult
  - For each CVD, BatchResult has an OperationResult, which displays the result of enforcing layers to a CVD. If OperationResult's Success is true, Mirage will create assignment for each layer of the CVD, and the Result is the IdValue of Id of the CVD. If the Success is false, refer to the MethodFault of OperationResult.

Example:

```
Id[] cvdIds = GetCvdId();
Client.Cvd_EnforceAllLayers(cvdIds, false, true);
```

Cvd_AllowNetworkOperations

Cvd_AllowNetworkOperations can be used to suspend or resume the device's network operations.

Input:

- cvdIds
  - cvdIds is a list of the CVD Ids. The list should not contain more than 500 entries. In the event that there are invalid CVD Ids in the list, this method only processes the valid CVD Ids.

- allow
  - If this parameter is false, the network operations will be halt, or the network operations will be resumed.

Return:

- BatchResult
  - For each CVD, BatchResult has an OperationResult, which displays the result of the CVD. If the Success is false, refer to the MethodFault of OperationResult.

Example:

```
Id[] cvdIds = GetCvdId();
Client.Cvd_AllowNetworkOperations(cvdIds, false);
```

Cvd_Restart

Cvd_Restart is used to reboot the device.

Input:

- cvdIds
  - cvdIds is a list of the CVD Ids. The list should not contain more than 500 entries. In the event that there are invalid CVD Ids in the list, this method only processes the valid CVD Ids..

Return:

- BatchResult
  - For each CVD, BatchResult has an OperationResult, which displays the result of the CVD. If the Success is false, refer to the MethodFault of OperationResult.

Example:

```
Id[] cvdIds = GetCvdId();
Client.Cvd_Restart(cvdIds);
```

Performance 3

The Mirage API operation method supports up to 500 operation targets.

- Cvd_ApplyPolicy
- Cvd_Archive
- Cvd_Delete
- Cvd_Sync
- OsMigration_Begin
- OsMigration_BeginDownloadOnly
- OsMigration_ApplyDownloadOnlyMigration
- PendingDevice_CreateNewCvd
- PendingDevice_Provision
- Assignment_ApplyDownloadOnly
- Cvd_AssignBaseLayer
- Cvd_AssignBaseLayerDownloadOnly
- Cvd_UpdateAppLayer
- Cvd_UpdateAppLayerDownloadOnly

Permissions, Configuration and Logging

4

You can configure Mirage API via a configuration file. Mirage API also writes log messages to log files to facilitate debugging. To successfully call some methods, a user must have specific permissions.

This chapter includes the following topics:

- "Permissions," on page 49
- "Configuration," on page 50
- "Logging," on page 50

Permissions

Mirage uses role-based access control to restrict system access to authorized users.

The following table lists the roles that are required by the API methods.

Table 4-1. Roles Required by Methods

Method	Roles Needed
Login	None
Logout	None
AppLayer_Query	ListBaseImages
BaseLayer_Query	ListBaseImages
Cvd_Query	ListBaseImages, ListEvents
Cvd_Get	ListBaseImages, ListEvents
CollectionCvd_Query	ListBaseImages, ListCvds
Policy_Query	ListPolicies
Volume_Query	ListLicense
PendingDevice_Query	ListDevices
PendingDevice_CreateNewCvd	AdministerCvds, ListDevices
OsMigration_Begin	AdministerCvds, ListCvds
OsMigration_BeginDownloadOnly	AdministerCvds, ListCvds
OsMigration_ApplyDownloadOnlyMigration	AdministerCvds, ListCvds
OsMigration_QueryDownloadOnlyInProgress	ListCvds
OsMigration_QueryDownloadOnlyCompleted	ListCvds
OsMigrationCvd_QueryDownloadOnly	ListCvds

Table 4-1. Roles Required by Methods (Continued)

Method	Roles Needed
Cvd_Archive	AdministerCvds, AdministerReferenceCvds, ListReferenceCvds, ListTasks
Cvd_Delete	AdministerCvds, ListCvds
Cvd_Sync	ListDevices, SupportDevices
Cvd_ApplyPolicy	AdministerCvds, ListCvds, ListPolicies
PendingDevice_Provision	AdministerCvds, ListBaseImages, ListDevices, ListLicense, ListPolicies
Collection_Query	ListCvds
Assignment_ApplyDownloadOnly	AdministerCvds, ListCvds
Assignment_Query	ListCvds
Cvd_AssignBaseLayer	AdministerCvds, ListCvds, ListBaseImages
Cvd_AssignBaseLayerDownloadOnly	AdministerCvds, ListCvds, ListBaseImages
Cvd_UpdateAppLayer	AdministerCvds, ListCvds, ListBaseImages
Cvd_UpdateAppLayerDownloadOnly	AdministerCvds, ListCvds, ListBaseImages

Configuration

Mirage API runs on the Mirage Web Manager. You can configure Mirage API by editing a configuration file.

The name of the configuration file is `C:\Program Files\Wanova\Mirage API\web.config`. The following table lists some important settings.

Table 4-2. Configuration File Settings

Setting	Default Value
Max receiving message size	31457280 (30MB)
Receive timeout	5 minutes
Send timeout	5 minutes

Logging

Mirage writes log messages to a log file for auditing and troubleshooting.

You can configure logging by editing `C:\Program Files\Wanova\Mirage API\log4net.config`. It contains all the settings for logging. The following table lists some of the settings and their default values.

Setting	Default Value
Log file name	mirageapi.log
Max log file size	10MB
Max log file backups	5 log files

The log files are located in `C:\ProgramData\Wanova Mirage\Web Management\logs`.

Sample Applications 5

Two sample applications are provided for your reference. One is written in C# and the other is written in Java. You can use them as a guide when you develop your applications.

This chapter includes the following topics:

- "Sample C# Application," on page 51
- "Sample Java Application," on page 64

Sample C# Application

This is a sample application that is written in C#. It queries entities in Mirage, centralizes endpoints, and migrates a CVD operating system.

```
//This is the C# sample code of Mirage API, which includes three parts.
//1. Query entities in Mirage, such as policy, volume, pending device, CVD, base layer and app
layer
//2. Centralize endpoints.
//3. Migrate CVD Operating System.
//4. CVD assign base layer.
//5. CVD update app layer.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.ServiceModel;
using System.Threading;
using vmware.mirage.mit.faults;
using vmware.mirage.mit.query;
using vmware.mirage.mit.types;

namespace SampleCode
{
    class MirageApiSample
    {
        private static MitServiceClient Client;

        private static int Main(string[] args)
        {
            if (args.Length < 4)
            {
```

```
                Console.WriteLine(
                                "Usage: SampleCode <Web_Management_Server_IP> <Username>
<Password> <migration_domain>");
                return -1;
            }
            Console.WriteLine("----------------- Login -----------------");
            Login(args[0], args[1], args[2]);

            Console.WriteLine("---------------VolumeQuery---------------");
            VolumeQuery("Default");

            Console.WriteLine("---------------PolicyQuery---------------");
            PolicyQuery("VMware Mirage default CVD policy");

            Console.WriteLine("-----------------CvdQuery----------------");
            CvdQuery(10001, "MIRAGEDOMAIN");

            Console.WriteLine("------------PendingDeviceQuery-----------");
            PendingDeviceQuery(23);

            Console.WriteLine("-----------------CeFlow------------------");
            CeFlow();

            Console.WriteLine("-------------OsMigrationFlow-------------");
            OsMigrationFlow(args[1], args[2], args[3]);

            Console.WriteLine("-----------ProvisioningFlow--------------");
            ProvisioningFlow(args[1], args[2], args[3]);

            Console.WriteLine("----------CvdAssignBaseLayerFlow----------");
            CvdAssignBaseLayerFlow();

            Console.WriteLine("---------CvdUpdateAppLayerFlow------------");
            CvdUpdateAppLayerFlow();

            return 0;
        }

        //Login Mirage API server with address, username and password.
        public static void Login(string address, string username, string password)
        {
            // Trust all certificates. This is just for sample code.
            ServicePointManager.ServerCertificateValidationCallback =
                ((sender, certificate, chain, sslPolicyErrors) => true);

            BasicHttpBinding binding = new BasicHttpBinding
            {
                AllowCookies = true,
                Security =
                {
                    Mode = BasicHttpSecurityMode.Transport
                }
            };

            //Connect to Mirage Web Management Server, default port is 7443
```

```
            EndpointAddress endpoint =
                new EndpointAddress(string.Format("https://{0}:7443/mirageapi/MitService.svc",
address));
            Client = new MitServiceClient(binding, endpoint);
            Client.Login(username, password);
            Console.WriteLine("Login success!");
        }

        // Query pending devices whose device id is deviceId, only the first page returns.
        public static void PendingDeviceQuery(long deviceId)
        {
            QueryDefinition query = new QueryDefinition
            {
                Filter = new QueryFilterEquals
                {
                    Field = FilterField.DEVICE_ID,
                    Value = new Id
                    {
                        IdValue = deviceId
                    }
                },
                Page = 1 //Page index starts from 1, not 0
            };
            QueryResult result = Client.PendingDevice_Query(query);
            if (result.Elements.Length == 0)
            {
                return;
            }
            foreach (DeviceDetails deviceDetails in result.Elements)
            {
                Console.WriteLine(
                                "ID: {0}, name: {1}, user name: {2}, OS version: {3}, vendor
name {4}, model name {5}, connected: {6}, last connected time: {7}, operation time: {8}",
                                deviceDetails.Id.IdValue, deviceDetails.Name,
deviceDetails.UserName,
                                deviceDetails.Os, deviceDetails.VendorName,
                                deviceDetails.ModelName, deviceDetails.Connected,
deviceDetails.LastConnectedTime,
                                deviceDetails.OperationTime);
            }
        }

        // Query volumes whose name is volumeName, only the first page returns.
        public static void VolumeQuery(string volumeName)
        {
            QueryDefinition query = new QueryDefinition
            {
                Filter = new QueryFilterBeginsWith
                {
                    Field = FilterField.VOLUME_NAME,
                    Value = volumeName
                },
                Page = 1
            };
            QueryResult result = Client.Volume_Query(query);
```

```
            if (result.Elements.Length == 0)
            {
                return;
            }
            foreach (VolumeDetails volumeDetails in result.Elements)
            {
                Console.WriteLine("ID: {0}, name: {1}, path: {2}", volumeDetails.Id.IdValue,
volumeDetails.Name,
                                  volumeDetails.Path);
            }
        }

        // Query polices whose name is policyName, only the first page returns.
        public static void PolicyQuery(string policyName)
        {
            QueryDefinition query = new QueryDefinition
            {
                Filter = new QueryFilterContains
                {
                    Field = FilterField.POLICY_NAME,
                    Value = policyName
                },
                Page = 1
            };
            QueryResult result = Client.Policy_Query(query);
            if (result.Elements.Length == 0)
            {
                return;
            }
            foreach (PolicyDetails policyDetails in result.Elements)
            {
                Console.WriteLine("ID: {0}, version: {1}.{2}, name: {3}",
                                  policyDetails.ImageId.Id.IdValue,
policyDetails.ImageId.Version.Major,
                                  policyDetails.ImageId.Version.Minor, policyDetails.Name);
            }
        }

        // Query Cvd with cvdId. It will throw FaultException<InvalidArgument> when the cvdId is
not the Id type.
        public static void CvdQuery(long cvdId)
        {
            QueryFilterEquals filter = new QueryFilterEquals
            {
                Field = FilterField.CVD_ID,
                Value = cvdId //Invalid value type for CVD_ID, expected type: Id
            };
            QueryDefinition query = new QueryDefinition
            {
                Filter = filter,
                Page = 1
            };
            try
            {
                Client.Cvd_Query(query);
```

```
            }
            catch (FaultException<InvalidArgument> e)
            {
                InvalidArgument detail = e.Detail;
                if (detail.GetType() == typeof(InvalidArgument))
                {
                    Console.WriteLine("InvalidArgument fault, detail message: " +
detail.Message);
                    //Expected output:
                    //InvalidArgument fault, detail message: Bad value for queryDefinition:
"Filter value for CVD_ID must be of type Id
                    //Parameter name: queryDefinition
                }
            }
        }

        // Query Cvd, whose Id isn't cvdId, and cvd name begins with cvdUsername.
        public static void CvdQuery(long cvdId, string cvdUsername)
        {
            QueryFilterNotEquals filterNotEquals = new QueryFilterNotEquals
            {
                Field = FilterField.CVD_ID,
                Value = new Id
                {
                    IdValue = cvdId
                }
            };
            QueryFilterBeginsWith filterBeginsWith = new QueryFilterBeginsWith
            {
                Field = FilterField.CVD_USER_NAME,
                Value = cvdUsername
            };
            QueryDefinition query = new QueryDefinition
            {
                Filter = new QueryFilterAnd
                {
                    Filters = new QueryFilter[] { filterNotEquals, filterBeginsWith }
                },
                Page = 1
            };
            QueryResult result = Client.Cvd_Query(query);
            if (result.Elements.Length == 0)
            {
                return;
            }
            foreach (CvdDetails cvdDetails in result.Elements)
            {
                Console.WriteLine("ID: {0}, device ID: {1}, policy ID: {2}, machine version: {3}.
{4}",
                                  cvdDetails.Id.IdValue, cvdDetails.DeviceId.IdValue,
cvdDetails.PolicyId.IdValue,
                                  cvdDetails.MachineVersion.Major,
cvdDetails.MachineVersion.Minor);
            }
        }
```

```
        // Centralization endpoint flow.
        public static void CeFlow()
        {
            QueryResult result = Client.PendingDevice_Query(new QueryDefinition { Filter = null,
Page = 1 });
            if (result.Elements.Length == 0)
            {
                return;
            }
            DeviceDetails pendingDevice = (DeviceDetails)result.Elements[0];

            result = Client.Volume_Query(new QueryDefinition { Filter = null, Page = 1 });
            if (result.Elements.Length == 0)
            {
                return;
            }
            VolumeDetails volume = (VolumeDetails)result.Elements[0];

            result = Client.Policy_Query(new QueryDefinition { Filter = null, Page = 1 });
            if (result.Elements.Length == 0)
            {
                return;
            }
            PolicyDetails policy = (PolicyDetails)result.Elements[0];

            Console.WriteLine("Creating new Cvd...");
            Id[] ids = { pendingDevice.Id };
            BatchResult batchResult = Client.PendingDevice_CreateNewCvd(ids, policy.ImageId,
volume.Id);
            OperationResult opResult = batchResult.results[0];
            if (opResult.Success == false)
            {
                Console.WriteLine("failed to create cvd, error: {0}", opResult.Fault);
                return;
            }

            // Check the centralization endpoint flow complete.
            while (true)
            {
                result = Client.Cvd_Query(new QueryDefinition
                {
                    Filter = new QueryFilterEquals
                    {
                        Field = FilterField.CVD_ID,
                        Value = new Id
                        {
                            IdValue = (long)opResult.Result
                        }
                    },
                    Page = 1
                });
                if (result.Elements.Length == 0)
                {
                    return;
```

```
                }
                CvdDetails cvdDetails = (CvdDetails)result.Elements[0];
                if (cvdDetails.MachineVersion.Major > 1 || cvdDetails.MachineVersion.Minor > 0)
                {
                    Console.WriteLine("ID: {0}, device ID: {1}, policy ID: {2}, machine version:
{3}.{4}",
                                      cvdDetails.Id.IdValue, cvdDetails.DeviceId.IdValue,
cvdDetails.PolicyId.IdValue,
                                      cvdDetails.MachineVersion.Major,
cvdDetails.MachineVersion.Minor);
                    break;
                }
                Thread.Sleep(TimeSpan.FromMinutes(2));
            }
        }

        //Pending device provisioning flow
        public static void ProvisioningFlow(string username, string password, string domain)
        {
            QueryResult result = Client.PendingDevice_Query(new QueryDefinition { Filter = null,
Page = 1 });
            if (result.Elements.Length == 0)
            {
                return;
            }
            DeviceDetails pendingDevice = (DeviceDetails)result.Elements[0];

            result = Client.Volume_Query(new QueryDefinition { Filter = null, Page = 1 });
            if (result.Elements.Length == 0)
            {
                return;
            }
            VolumeDetails volume = (VolumeDetails)result.Elements[0];

            result = Client.Policy_Query(new QueryDefinition { Filter = null, Page = 1 });
            if (result.Elements.Length == 0)
            {
                return;
            }
            PolicyDetails policy = (PolicyDetails)result.Elements[0];

            result = Client.BaseLayer_Query(new QueryDefinition { Filter = null, Page = 1 });
            if (result.Elements.Length == 0)
            {
                return;
            }
            LayerDetails baseLayer = (LayerDetails)result.Elements[0];

            result = Client.AppLayer_Query(new QueryDefinition { Filter = null, Page = 1 });
            if (result.Elements.Length == 0)
            {
                return;
            }
            LayerDetails appLayer = (LayerDetails)result.Elements[0];
```

```
            Console.WriteLine("Provisioning new Cvd...");

            Id[] ids = { pendingDevice.Id };
            ImageId[] appLayers = { appLayer.ImageId };
            MachineIdentityInfo identityInfo = new MachineIdentityInfo
            {
                DomainMember = true,
                DomainOrWorkgroupName = domain,
                User = username,
                Password = password
            };
            BatchResult batchResult = Client.PendingDevice_Provision(ids, policy.ImageId,
baseLayer.ImageId, appLayers,
                                                                     identityInfo, volume.Id,
false);
            OperationResult opResult = batchResult.results[0];
            if (opResult.Success == false)
            {
                Console.WriteLine("failed to provision cvd, error: {0}", opResult.Fault);
                return;
            }

            //Wait for provision complete
            //Assignments include both base layer and app layer assignment.
            WaitForAssignmentsComplete("DeviceProvisioning", new Id{ IdValue =
(long)opResult.Result });
        }

        // OS migration flow.
        public static void OsMigrationFlow(string username, string password, string domain)
        {
            // Suppose we have one win7 32bit base layer and one xp cvd
            QueryDefinition query = new QueryDefinition
            {
                Filter = null,
                Page = 1
            };
            QueryResult result = Client.Cvd_Query(query);
            if (result.Elements.Length == 0)
            {
                return;
            }
            CvdDetails cvd = (CvdDetails)result.Elements[0];
            result = Client.BaseLayer_Query(query);
            if (result.Elements.Length == 0)
            {
                return;
            }
            LayerDetails baseLayer = (LayerDetails)result.Elements[0];

            ImageId[] appLayerImageIds = GetAppLayerImageIds();

            MigrationTarget target = new MigrationTarget
            {
                CvdId = cvd.Id,
```

```
                IdentityInfo = new MachineIdentityInfo
                {
                    DomainMember = true,
                    DomainOrWorkgroupName = domain,
                    User = username,
                    Password = password
                }
            };

            OperationResult operationResult =
                Client.OsMigration_BeginDownloadOnly(new[] { target }, baseLayer.ImageId,
appLayerImageIds, true).results[0];
            if (!operationResult.Success)
            {
                return;
            }

            WaitForOsMigrationDownloadInProgress();
            WaitForOsMigrationDownloadCompleted();

            operationResult = Client.OsMigration_ApplyDownloadOnlyMigration(new[]
{ cvd.Id }).results[0];
            if (!operationResult.Success)
            {
                return;
            }
            Console.WriteLine("Applying migration...");
            //Wait for migration complete
            WaitForAssignmentsComplete("Migration", cvd.Id);
        }

        //Cvd assign base layer flow
        public static void CvdAssignBaseLayerFlow()
        {
            const string baselayerDownloadOnlyTaskName = "BaseImageDownloadOnly";
            const string applyLayerTaskName = "ApplyLayers";
            QueryResult result = Client.Cvd_Query(new QueryDefinition
            {
                Filter = null,
                Page = 1
            });

            if (result.Elements.Length == 0)
            {
                return;
            }
            CvdDetails cvd = (CvdDetails)result.Elements[0];
            result = Client.BaseLayer_Query(new QueryDefinition
            {
                Filter = null,
                Page = 1
            });

            if (result.Elements.Length == 0)
            {
```

```
                return;
            }
            LayerDetails baseLayer = (LayerDetails)result.Elements[0];


            //Download only assign base layer
            Console.WriteLine("Base layer download...");
            BatchResult batchResult = Client.Cvd_AssignBaseLayerDownloadOnly(new[]{cvd.Id},
baseLayer.ImageId, true);
            OperationResult opResult = batchResult.results[0];
            if (opResult.Success == false)
            {
                Console.WriteLine("failed to assign base layer for cvd, error: {0}",
opResult.Fault);
                return;
            }

            WaitForAssignmentsComplete(baselayerDownloadOnlyTaskName, cvd.Id);

            //Query Assignment and apply
            result = AssignmentQueryWithTaskAndCvdId(baselayerDownloadOnlyTaskName, cvd.Id);
            if (result.Elements.Length == 0)
            {
                return;
            }
            AssignmentDetails assignmentDetails = (AssignmentDetails) result.Elements[0];

            Console.WriteLine("Apply base layer assignment...");
            batchResult = Client.Assignment_ApplyDownloadOnly(new[]{assignmentDetails.Id}, true);
            opResult = batchResult.results[0];
            if (opResult.Success == false)
            {
                Console.WriteLine("failed to apply layers for cvd, error: {0}", opResult.Fault);
                return;
            }
            //Wait for assignment complete
            //Apply assignment will create a new assignment with task name "ApplyLayers"
            WaitForAssignmentsComplete(applyLayerTaskName, cvd.Id);
        }

        //Cvd update app layer flow
        public static void CvdUpdateAppLayerFlow()
        {
            const string applayerDownloadOnlyTaskName = "AppLayerDownloadOnly";
            const string applyLayerTaskName = "ApplyLayers";
            QueryResult result = Client.Cvd_Query(new QueryDefinition
            {
                Filter = null,
                Page = 1
            });

            if (result.Elements.Length == 0)
            {
                return;
            }
```

```
            CvdDetails cvd = (CvdDetails)result.Elements[0];
            result = Client.AppLayer_Query(new QueryDefinition
            {
                Filter = null,
                Page = 1
            });

            if (result.Elements.Length == 0)
            {
                return;
            }
            LayerDetails appLayer = (LayerDetails)result.Elements[0];

            //Download only update app layer
            Console.WriteLine("App layer download...");
            BatchResult batchResult = Client.Cvd_UpdateAppLayerDownloadOnly(new[] { cvd.Id },
new []{ appLayer.ImageId }, null, true);
            OperationResult opResult = batchResult.results[0];
            if (opResult.Success == false)
            {
                Console.WriteLine("failed to update app layer for cvd, error: {0}",
opResult.Fault);
                return;
            }

            WaitForAssignmentsComplete(applayerDownloadOnlyTaskName, cvd.Id);

            //Query assignment and apply
            result = AssignmentQueryWithTaskAndCvdId(applayerDownloadOnlyTaskName, cvd.Id);
            if (result.Elements.Length == 0)
            {
                return;
            }
            AssignmentDetails assignmentDetails = (AssignmentDetails)result.Elements[0];

            Console.WriteLine("Apply app layer assignment...");
            batchResult = Client.Assignment_ApplyDownloadOnly(new[] { assignmentDetails.Id },
true);
            opResult = batchResult.results[0];
            if (opResult.Success == false)
            {
                Console.WriteLine("failed to apply layers for cvd, error: {0}", opResult.Fault);
                return;
            }
            //Wait for assignment complete
            //Apply assignment will create a new assignment with task name "ApplyLayers"
            WaitForAssignmentsComplete(applyLayerTaskName, cvd.Id);
        }

        // Check the Cvd completes to download base layer and app layers for migration.
        private static void WaitForOsMigrationDownloadCompleted()
        {
            while (true)
            {
                QueryDefinition query = new QueryDefinition
```

```
                {
                    Filter = new QueryFilterEquals
                    {
                        Field = FilterField.DOWNLOAD_ONLY_MIGRATION_STATUS,
                        Value = "Complete"
                    },
                    Page = 1
                };
                QueryResult result = Client.OsMigrationCvd_QueryDownloadOnly(query);
                if (result.Elements.Length != 0)
                {
                    Console.WriteLine("Migration download success!");
                    break;
                }
                Thread.Sleep(TimeSpan.FromMinutes(2));
            }
        }

        // Check the Cvd is downloading base layer or app layers for migration.
        private static void WaitForOsMigrationDownloadInProgress()
        {
            while (true)
            {
                QueryDefinition query = new QueryDefinition
                {
                    Filter = new QueryFilterEquals
                    {
                        Field = FilterField.DOWNLOAD_ONLY_MIGRATION_STATUS,
                        Value = "Downloading"
                    },
                    Page = 1
                };
                QueryResult result = Client.OsMigrationCvd_QueryDownloadOnly(query);
                if (result.Elements.Length != 0)
                {
                    Console.WriteLine("Migration downloading...");
                    break;
                }
                Thread.Sleep(TimeSpan.FromMinutes(2));
            }
        }

        //Wait for assignment complete
        private static void WaitForAssignmentsComplete(string taskName, Id cvdId)
        {
            while (true)
            {
                QueryResult result = AssignmentQueryWithTaskAndCvdId(taskName, cvdId);
                if (result.Elements != null && result.Elements.Length > 0)
                {
                    AssignmentDetails[] assignments = result.Elements.Select(e =>
(AssignmentDetails)e).ToArray();
                    if (IsAssignmentComplete(assignments))
                    {
                        break;
```

```
                    }
                }

                Thread.Sleep(TimeSpan.FromMinutes(2));
            }
        }

        //Query assignment
        private static QueryResult AssignmentQueryWithTaskAndCvdId(string taskName, Id cvdId)
        {
            QueryFilter taskNameFilter = new QueryFilterEquals
            {
                Field = FilterField.ASSIGNMENT_TASK_NAME,
                Value = taskName
            };
            QueryFilter cvdIdFilter = new QueryFilterEquals
            {
                Field = FilterField.ASSIGNMENT_CVD_ID,
                Value = cvdId
            };
            QueryFilterAnd filter = new QueryFilterAnd { Filters = new[] { taskNameFilter,
cvdIdFilter } };

            return Client.Assignment_Query(new QueryDefinition { Filter = filter, Page = 1 });
        }

        //Check if all assignments complete
        private static bool IsAssignmentComplete(AssignmentDetails[] assignments)
        {
            if (assignments == null || assignments.Length == 0)
            {
                return false;
            }
            HashSet<string> statusFailed = new HashSet<string>(new[] { "Cancelled", "Rejected",
"Blocked" });
            const string statusDone = "Done";

            bool assignmentsDone = true;
            foreach (AssignmentDetails assignment in assignments)
            {
                if (statusFailed.Contains(assignment.Status))
                {
                    Console.WriteLine("Assignment failed, operation failed!");
                    return true;
                }
                assignmentsDone &= assignment.Status.Equals(statusDone);
            }
            if (assignmentsDone)
            {
                Console.WriteLine("Assignment done, operation succeed!");
            }
            return assignmentsDone;
        }

        // Get App layers.
```

```
        private static ImageId[] GetAppLayerImageIds()
        {
            QueryResult result = Client.AppLayer_Query(new QueryDefinition { Filter = null, Page
= 1 });
            ImageId[] appLayerImageIds;
            if (result.Elements.Length != 0)
            {
                LayerDetails[] appLayers = Array.ConvertAll(result.Elements, a =>
(LayerDetails)a);
                appLayerImageIds = appLayers.Select(a => a.ImageId).ToArray();
            }
            else
            {
                appLayerImageIds = null;
            }
            return appLayerImageIds;
        }
    }
}
```

Sample Java Application

This is a sample application that is written in Java. It queries entities in Mirage, centralizes endpoints, and migrates a CVD operating system.

```
//This is the Java sample code of Mirage API, which includes three parts.
//1. Query entities in Mirage, such as policy, volume, pending device, CVD, base layer and app
layer
//2. Centralize endpoints.
//3. Migrate CVD Operating System.
//4. CVD assign base layer.
//5. CVD update app layer.
package com.vmware.mirage.mit.sample;

import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.util.HashSet;
import java.util.concurrent.TimeUnit;

import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;

import org.apache.axis2.AxisFault;
import org.apache.axis2.java.security.SSLProtocolSocketFactory;
import org.apache.axis2.java.security.TrustAllTrustManager;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.commons.httpclient.HttpState;
import org.apache.commons.httpclient.protocol.Protocol;
import org.apache.commons.httpclient.protocol.ProtocolSocketFactory;
import org.apache.log4j.Logger;

import com.vmware.mirage.mit.MitServiceStub;
import com.vmware.mirage.mit.MitServiceStub.*;

public class MirageAPISample {
```

```
    private static Logger logger = Logger.getLogger(MirageAPISample.class);

    private MitServiceStub client = null;

    private static String ENDPOINT = "https://<Web Management Server IP>:
7443/mirageapi/mitservice.svc";

    private static String USERNAME = "<username>";

    private static String PASSWORD = "<password>";

    private static String DOMAIN = "<domain name>";

    public static void main(final String... args) throws Exception {
        final MirageAPISample sample = new MirageAPISample();
        sample.login();
        try {
            sample.policyQuery();
            sample.volumeQuery();
            sample.cvdQuery();
            sample.pendingDeviceQuery();
            sample.baselayerQuery();
            sample.applayerQuery();
            sample.ceFlow();
            sample.migrationFlow();
            sample.provisionFlow();
            sample.cvdAssignBaseLayerFlow();
            sample.cvdUpdateAppLayerFlow();
        } finally {
            try {
                sample.logout();
            } catch (final Exception e) {
                logger.error("Failed to logout.", e);
            }
        }
    }

    // Create and configure API client with ENDPOINT
    public MirageAPISample() throws AxisFault, KeyManagementException, NoSuchAlgorithmException {
        client = new MitServiceStub(ENDPOINT);
        configureClient();
    }

    // Login Mirage API server with username and password
    public void login() throws Exception {

        final Login login = new Login();
        login.setUsername(USERNAME);
        login.setPassword(PASSWORD);
        client.login(login);
    }

    // Logout Mirage API server
    public void logout() throws Exception {
        final Logout logout = new Logout();
```

```
            client.logout(logout);
        }

        // Query policy without filter, only the first page returns.
        public void policyQuery() throws Exception {
            final QueryDefinition queryDefinition = new QueryDefinition();
            queryDefinition.setPage(1);

            final Policy_Query policy_Query = new Policy_Query();
            policy_Query.setQueryDefinition(queryDefinition);
            final Policy_QueryResponse policies = client.policy_Query(policy_Query);
            final QueryResult result = policies.getPolicy_QueryResult();

            if (result.getElements().getAnyType() != null) {
                for (final Object element : result.getElements().getAnyType()) {
                    final PolicyDetails policyDetails = (PolicyDetails) element;
                    System.out.println(String.format("policyDetails %s %s", policyDetails.getName(),
                            policyDetails.getImageId()));
                }
            }
        }

        // Query volumes without filter, only the first page returns.
        public void volumeQuery() throws Exception {
            final QueryDefinition queryDefinition = new QueryDefinition();
            queryDefinition.setPage(1);
            queryDefinition.setFilter(null);

            final Volume_Query volume_Query = new Volume_Query();
            volume_Query.setQueryDefinition(queryDefinition);
            final Volume_QueryResponse volumes = client.volume_Query(volume_Query);
            final QueryResult result = volumes.getVolume_QueryResult();

            if (result.getElements().getAnyType() != null) {
                for (final Object element : result.getElements().getAnyType()) {
                    final VolumeDetails volumeDetails = (VolumeDetails) element;
                    System.out.println(String.format("volumeDetails %s %s", volumeDetails.getName(),
                            volumeDetails.getPath()));
                }
            }
        }

        // Query Cvds whose name starts with "VMware" and progress equals to 100, only the first
    page returns.
        public void cvdQuery() throws Exception {

            /* name starts with "VMware" */
            final QueryFilterBeginsWith beginFilter = new QueryFilterBeginsWith();
            beginFilter.setField(FilterField.CVD_NAME);
            beginFilter.setValue("VMware");

            /* progress is 100 */
            final QueryFilterEquals equalFilter = new QueryFilterEquals();
            equalFilter.setField(FilterField.CVD_PROGRESS);
            equalFilter.setValue((long) 100); // must use long for progress
```

```
        /* Create and filter */
        final QueryFilterAnd andFilter = new QueryFilterAnd();
        final ArrayOfQueryFilter filterArr = new ArrayOfQueryFilter();
        filterArr.addQueryFilter(beginFilter);
        filterArr.addQueryFilter(equalFilter);
        andFilter.setFilters(filterArr);

        final QueryDefinition queryDefinition = new QueryDefinition();
        queryDefinition.setPage(1);
        queryDefinition.setFilter(andFilter);
        final Cvd_Query cvdQuery = new Cvd_Query();
        cvdQuery.setQueryDefinition(queryDefinition);
        final Cvd_QueryResponse cvds = client.cvd_Query(cvdQuery);
        final QueryResult result = cvds.getCvd_QueryResult();
        if (result.getElements().getAnyType() != null) {
            for (final Object element : result.getElements().getAnyType()) {
                final CvdDetails cvdDetails = (CvdDetails) element;
                System.out.println(String.format("cvdDetails %s %s %s", cvdDetails.getId(),
cvdDetails.getName(),
                        cvdDetails.getOperationProgress()));
            }
        }
    }

    // Query pending device whose state is disconnect, only the first page returns.
    public void pendingDeviceQuery() throws Exception {

        /* Disconnected */
        final QueryFilterEquals equalFilter = new QueryFilterEquals();
        equalFilter.setField(FilterField.DEVICE_CONNECTION_STATE);
        equalFilter.setValue(false);

        final QueryDefinition queryDefinition = new QueryDefinition();
        queryDefinition.setFilter(equalFilter);
        queryDefinition.setPage(1);

        final PendingDevice_Query pendingDeviceQuery = new PendingDevice_Query();
        pendingDeviceQuery.setQueryDefinition(queryDefinition);

        final PendingDevice_QueryResponse pendingDevices =
client.pendingDevice_Query(pendingDeviceQuery);
        final QueryResult result = pendingDevices.getPendingDevice_QueryResult();
        if (result.getElements().getAnyType() != null) {
            for (final Object element : result.getElements().getAnyType()) {
                final DeviceDetails deviceDetails = (DeviceDetails) element;
                System.out.println(String.format("deviceDetails %s %s", deviceDetails.getName(),
                        deviceDetails.getConnected()));
            }
        }
    }

    // Query base layers without filter, only the first page returns.
    public void baselayerQuery() throws Exception {
        final QueryDefinition queryDefinition = new QueryDefinition();
```

```
            queryDefinition.setPage(1);

            final BaseLayer_Query baseLayerQuery = new BaseLayer_Query();
            baseLayerQuery.setQueryDefinition(queryDefinition);

            final BaseLayer_QueryResponse baseLayers = client.baseLayer_Query(baseLayerQuery);
            final QueryResult result = baseLayers.getBaseLayer_QueryResult();
            if (result.getElements().getAnyType() != null) {
                for (final Object element : result.getElements().getAnyType()) {
                    final LayerDetails baseLayerDetails = (LayerDetails) element;
                    System.out.println(String.format("baseLayer %s %s", baseLayerDetails.getName(),
                            baseLayerDetails.getImageId()));
                }
            }
        }

        // Query app layers without filter, only the first page returns.
        public void applayerQuery() throws Exception {
            final QueryDefinition queryDefinition = new QueryDefinition();
            queryDefinition.setPage(1);

            final AppLayer_Query baseLayerQuery = new AppLayer_Query();
            baseLayerQuery.setQueryDefinition(queryDefinition);

            final AppLayer_QueryResponse baseLayers = client.appLayer_Query(baseLayerQuery);
            final QueryResult result = baseLayers.getAppLayer_QueryResult();
            if (result.getElements().getAnyType() != null) {
                for (final Object element : result.getElements().getAnyType()) {
                    final LayerDetails baseLayerDetails = (LayerDetails) element;
                    System.out.println(String.format("appLayer %s %s", baseLayerDetails.getName(),
                            baseLayerDetails.getImageId()));
                }
            }
        }

        // Centralization endpoint flow.
        public void ceFlow() throws Exception {
            /* Get policy */
            final QueryFilterContains policyFilter = new QueryFilterContains();
            policyFilter.setField(FilterField.POLICY_NAME);
            policyFilter.setValue("default");
            final QueryDefinition queryDefinition = new QueryDefinition();
            queryDefinition.setPage(1);
            queryDefinition.setFilter(policyFilter);
            final Policy_Query policyQuery = new Policy_Query();
            policyQuery.setQueryDefinition(queryDefinition);
            final Policy_QueryResponse policyResponse = client.policy_Query(policyQuery);
            final Object[] policyArr =
    policyResponse.getPolicy_QueryResult().getElements().getAnyType();
            if (policyArr.length == 0) {
                return;
            }
            final PolicyDetails policy = (PolicyDetails) policyArr[0];

            /* Get volume */
```

```
        final QueryFilterEquals volumeFilter = new QueryFilterEquals();
        policyFilter.setField(FilterField.VOLUME_NAME);
        policyFilter.setValue("DefaultVolume");
        queryDefinition.setFilter(volumeFilter);
        final Volume_Query volumeQuery = new Volume_Query();
        volumeQuery.setQueryDefinition(queryDefinition);
        final Volume_QueryResponse volumeResponse = client.volume_Query(volumeQuery);
        final Object[] volumeArr =
volumeResponse.getVolume_QueryResult().getElements().getAnyType();
        if (volumeArr.length == 0) {
            return;
        }
        final VolumeDetails volume = (VolumeDetails) volumeArr[0];

        /* Get device */
        final PendingDevice_Query pendingDeviceQuery = new PendingDevice_Query();
        pendingDeviceQuery.setQueryDefinition(queryDefinition);
        final PendingDevice_QueryResponse pendingDeviceResponse =
client.pendingDevice_Query(pendingDeviceQuery);
        final Object[] deviceArr =
pendingDeviceResponse.getPendingDevice_QueryResult().getElements().getAnyType();
        if (deviceArr.length == 0) {
            return;
        }
        final DeviceDetails device = (DeviceDetails) deviceArr[0];

        /* Create new Cvd */
        final PendingDevice_CreateNewCvd pendingDeviceCreateNewCvd = new
PendingDevice_CreateNewCvd();
        pendingDeviceCreateNewCvd.setPolicyId(policy.getImageId());
        pendingDeviceCreateNewCvd.setVolumeId(volume.getId());
        final ArrayOfId ids = new ArrayOfId();
        ids.addId(device.getId());
        pendingDeviceCreateNewCvd.setDeviceIds(ids);
        final PendingDevice_CreateNewCvdResponse batchResultResponse = client
                .pendingDevice_CreateNewCvd(pendingDeviceCreateNewCvd);
        final BatchResult batchResult =
batchResultResponse.getPendingDevice_CreateNewCvdResult();

        /* Get the Id of new Cvd from BatchResult */
        final OperationResult[] operationResultArr =
batchResult.getResults().getOperationResult();
        if (operationResultArr.length == 0) {
            throw new Exception("There is no result for PendingDevice_CreateNewCvd.");
        }
        final OperationResult result = operationResultArr[0];
        final long idValue = (long) result.getResult();
        final Id cvdId = new Id();
        cvdId.setIdValue(idValue);

        /* Validate the completion of creating Cvd */
        final QueryFilterEquals equalFilter = new QueryFilterEquals();
        equalFilter.setField(FilterField.CVD_ID);
        equalFilter.setValue(cvdId);
        queryDefinition.setFilter(equalFilter);
```

```
        queryDefinition.setPage(1);
        final Cvd_Query cvdQuery = new Cvd_Query();
        cvdQuery.setQueryDefinition(queryDefinition);
        while (true) {
            final Cvd_QueryResponse cvds = client.cvd_Query(cvdQuery);
            final QueryResult cvdResult = cvds.getCvd_QueryResult();
            final Object[] cvdResultArr = cvdResult.getElements().getAnyType();
            if (cvdResultArr.length > 0) {
                final CvdDetails cvdDetails = (CvdDetails) cvdResultArr[0];
                final ImageVersion machineVersion = cvdDetails.getMachineVersion();
                /* The complete condition of CE flow is that the machine version of CVD is not
1.0 */
                if (machineVersion.getMajor() != 1 || machineVersion.getMinor() != 0) {
                    break;
                }
                TimeUnit.MINUTES.sleep(2);
            }
        }
        System.out.println("OK to create new cvd.");
    }

    public void provisionFlow() throws Exception {
        /* Get policy */
        final QueryDefinition queryDefinition = new QueryDefinition();
        queryDefinition.setPage(1);
        final Policy_Query policyQuery = new Policy_Query();
        policyQuery.setQueryDefinition(queryDefinition);
        final Policy_QueryResponse policyResponse = client.policy_Query(policyQuery);
        final Object[] policyArr =
policyResponse.getPolicy_QueryResult().getElements().getAnyType();
        if (policyArr.length == 0) {
            return;
        }
        final PolicyDetails policy = (PolicyDetails) policyArr[0];

        /* Get volume */
        final Volume_Query volumeQuery = new Volume_Query();
        volumeQuery.setQueryDefinition(queryDefinition);
        final Volume_QueryResponse volumeResponse = client.volume_Query(volumeQuery);
        final Object[] volumeArr =
volumeResponse.getVolume_QueryResult().getElements().getAnyType();
        if (volumeArr.length == 0) {
            return;
        }
        final VolumeDetails volume = (VolumeDetails) volumeArr[0];

        /* Get device */
        final PendingDevice_Query pendingDeviceQuery = new PendingDevice_Query();
        pendingDeviceQuery.setQueryDefinition(queryDefinition);
        final PendingDevice_QueryResponse pendingDeviceResponse =
client.pendingDevice_Query(pendingDeviceQuery);
        final Object[] deviceArr =
pendingDeviceResponse.getPendingDevice_QueryResult().getElements().getAnyType();
        if (deviceArr.length == 0) {
            return;
```

```
        }
        final DeviceDetails device = (DeviceDetails) deviceArr[0];

        /* Get base layer */
        final BaseLayer_Query baseLayerQuery = new BaseLayer_Query();
        baseLayerQuery.setQueryDefinition(queryDefinition);
        final BaseLayer_QueryResponse baseLayerResponse = client.baseLayer_Query(baseLayerQuery);
        final Object[] baseLayerArr =
baseLayerResponse.getBaseLayer_QueryResult().getElements().getAnyType();
        if (baseLayerArr.length == 0) {
            return;
        }
        final LayerDetails baseLayerDetails = (LayerDetails) baseLayerArr[0];

        /* Start device provision */
        final MachineIdentityInfo machineIdentityInfo = new MachineIdentityInfo();
        machineIdentityInfo.setDomainMember(true);
        machineIdentityInfo.setDomainOrWorkgroupName(DOMAIN);
        machineIdentityInfo.setUser(USERNAME);
        machineIdentityInfo.setPassword(PASSWORD);

        final PendingDevice_Provision pendingDeviceProvision = new PendingDevice_Provision();
        pendingDeviceProvision.setPolicyImageId(policy.getImageId());
        pendingDeviceProvision.setVolumeId(volume.getId());
        pendingDeviceProvision.setBaseLayerImageId(baseLayerDetails.getImageId());
        pendingDeviceProvision.setIdentityInfo(machineIdentityInfo);
        pendingDeviceProvision.setIgnoreWarnings(true);

        final ArrayOfId ids = new ArrayOfId();
        ids.addId(device.getId());
        pendingDeviceProvision.setPendingDevices(ids);
        final PendingDevice_ProvisionResponse provisionResponse = client
                .pendingDevice_Provision(pendingDeviceProvision);
        final BatchResult batchResult = provisionResponse.getPendingDevice_ProvisionResult();
        final OperationResult[] provisionResultArr =
batchResult.getResults().getOperationResult();
        if (provisionResultArr.length == 0) {
            throw new Exception("There is no result for pendingDevice_Provision.");
        }
        final OperationResult result = provisionResultArr[0];
        if (result.getSuccess() == false) {
            final String message = result.getFault() == null ? "" :
result.getFault().getMessage();
            throw new Exception("Failed to provision device. " + message);
        }

        /* Wait for provision complete */
        final long idValue = (long) result.getResult();
        final Id cvdId = new Id();
        cvdId.setIdValue(idValue);

        waitForAssignmentComplete("DeviceProvisioning", cvdId);
    }

    // OS migration flow.
```

```
    public void migrationFlow() throws Exception {
        final QueryDefinition queryDefinition = new QueryDefinition();
        queryDefinition.setPage(1);

        final Cvd_Query cvdQuery = new Cvd_Query();
        cvdQuery.setQueryDefinition(queryDefinition);

        final Cvd_QueryResponse cvdResponse = client.cvd_Query(cvdQuery);
        final QueryResult cvdResult = cvdResponse.getCvd_QueryResult();

        final Object[] cvdArr = cvdResult.getElements().getAnyType();
        if (cvdArr.length == 0) {
            throw new Exception("There is no result for Cvd_Query");
        }

        final CvdDetails cvd = (CvdDetails) cvdArr[0];

        final BaseLayer_Query baseLayerQuery = new BaseLayer_Query();
        baseLayerQuery.setQueryDefinition(queryDefinition);
        final BaseLayer_QueryResponse baseLayerResponse = client.baseLayer_Query(baseLayerQuery);
        final QueryResult baseLayerResult = baseLayerResponse.getBaseLayer_QueryResult();

        final Object[] baseLayerArr = baseLayerResult.getElements().getAnyType();
        if (baseLayerArr.length == 0) {
            throw new Exception("There is no result for baseLayer_Query");
        }
        final LayerDetails baseLayer = (LayerDetails) baseLayerArr[0];

        final MachineIdentityInfo machineIdentityInfo = new MachineIdentityInfo();
        machineIdentityInfo.setDomainMember(true);
        machineIdentityInfo.setDomainOrWorkgroupName(DOMAIN);
        machineIdentityInfo.setUser(USERNAME);
        machineIdentityInfo.setPassword(PASSWORD);

        final MigrationTarget migrationTarget = new MigrationTarget();
        migrationTarget.setIdentityInfo(machineIdentityInfo);
        migrationTarget.setCvdId(cvd.getId());

        final ArrayOfMigrationTarget migrationTargets = new ArrayOfMigrationTarget();
        migrationTargets.addMigrationTarget(migrationTarget);

        final OsMigration_Begin osMigration_Begin = new OsMigration_Begin();
        osMigration_Begin.setMigrationTargets(migrationTargets);
        osMigration_Begin.setBaseLayerId(baseLayer.getImageId());
        osMigration_Begin.setAppLayerIds(null);
        osMigration_Begin.setIgnoreWarnings(true);

        final OsMigration_BeginResponse migrationResponse =
client.osMigration_Begin(osMigration_Begin);
        final BatchResult migrationResult = migrationResponse.getOsMigration_BeginResult();

        final OperationResult[] results = migrationResult.getResults().getOperationResult();
        if (results.length == 0) {
            throw new Exception("There is no result for osMigration_Begin");
        }
```

```
        final OperationResult result = results[0];

        if (result.getSuccess() == true) {
            System.out.println(String.format("Migration begin successfully for Cvd(Id=%d)",
cvd.getId().getIdValue()));
        } else {
            System.out.println(String.format("Migration begin failed for Cvd(Id=%d), fault is:
%s", cvd.getId()
                    .getIdValue(), result.getFault().getMessage()));
        }

        /* Wait for migration complete */
        waitForAssignmentComplete("Migration", cvd.getId());
    }

    //Cvd assign base layer flow
    public void cvdAssignBaseLayerFlow() throws Exception {
        final QueryDefinition queryDefinition = new QueryDefinition();
        queryDefinition.setPage(1);

        final Cvd_Query cvdQuery = new Cvd_Query();
        cvdQuery.setQueryDefinition(queryDefinition);

        final Cvd_QueryResponse cvdResponse = client.cvd_Query(cvdQuery);
        final QueryResult cvdResult = cvdResponse.getCvd_QueryResult();

        final Object[] cvdArr = cvdResult.getElements().getAnyType();
        if (cvdArr.length == 0) {
            throw new Exception("There is no result for Cvd_Query");
        }

        final CvdDetails cvd = (CvdDetails) cvdArr[0];

        final BaseLayer_Query baseLayerQuery = new BaseLayer_Query();
        baseLayerQuery.setQueryDefinition(queryDefinition);
        final BaseLayer_QueryResponse baseLayerResponse = client.baseLayer_Query(baseLayerQuery);
        final QueryResult baseLayerResult = baseLayerResponse.getBaseLayer_QueryResult();

        final Object[] baseLayerArr = baseLayerResult.getElements().getAnyType();
        if (baseLayerArr.length == 0) {
            throw new Exception("There is no result for BaseLayer_Query");
        }
        final LayerDetails baseLayer = (LayerDetails) baseLayerArr[0];

        /* Assign base layer to the CVD */
        final Cvd_AssignBaseLayer cvdAssignBaseLayer = new Cvd_AssignBaseLayer();
        final ArrayOfId ids = new ArrayOfId();
        ids.addId(cvd.getId());
        cvdAssignBaseLayer.setCvdIds(ids);
        cvdAssignBaseLayer.setBaseLayerImageId(baseLayer.getImageId());
        cvdAssignBaseLayer.setIgnoreWarnings(true);

        final Cvd_AssignBaseLayerResponse assignResponse =
client.cvd_AssignBaseLayer(cvdAssignBaseLayer);
```

```
            final BatchResult assignResult = assignResponse.getCvd_AssignBaseLayerResult();

            final OperationResult[] results = assignResult.getResults().getOperationResult();
            if (results.length == 0) {
                throw new Exception("There is no result for cvd_AssignBaseLayer");
            }

            final OperationResult result = results[0];

            if (result.getSuccess() == true) {
                System.out.println(String.format("Cvd assign baselayer begin successfully for Cvd(Id=
        %d)", cvd.getId()
                        .getIdValue()));
            } else {
                System.out.println(String.format("Cvd assign baselayer begin failed for Cvd(Id=%d),
        fault is: %s", cvd
                        .getId().getIdValue(), result.getFault().getMessage()));
            }

            /* Wait for base layer assign complete */
            waitForAssignmentComplete("AssignBaseImage", cvd.getId());
        }

        //Cvd update app layer flow
        public void cvdUpdateAppLayerFlow() throws Exception {
            final QueryDefinition queryDefinition = new QueryDefinition();
            queryDefinition.setPage(1);

            final Cvd_Query cvdQuery = new Cvd_Query();
            cvdQuery.setQueryDefinition(queryDefinition);

            final Cvd_QueryResponse cvdResponse = client.cvd_Query(cvdQuery);
            final QueryResult cvdResult = cvdResponse.getCvd_QueryResult();

            final Object[] cvdArr = cvdResult.getElements().getAnyType();
            if (cvdArr.length == 0) {
                throw new Exception("There is no result for Cvd_Query");
            }

            final CvdDetails cvd = (CvdDetails) cvdArr[0];

            final AppLayer_Query appLayerQuery = new AppLayer_Query();
            appLayerQuery.setQueryDefinition(queryDefinition);
            final AppLayer_QueryResponse appLayerResponse = client.appLayer_Query(appLayerQuery);
            final QueryResult appLayerResult = appLayerResponse.getAppLayer_QueryResult();

            final Object[] appLayerArr = appLayerResult.getElements().getAnyType();
            if (appLayerArr.length == 0) {
                throw new Exception("There is no result for AppLayer_Query");
            }
            final LayerDetails appLayer = (LayerDetails) appLayerArr[0];

            /*Update Cvd app layer*/
            final Cvd_UpdateAppLayer cvdUpdateAppLayer = new Cvd_UpdateAppLayer();
            final ArrayOfId ids = new ArrayOfId();
```

```
        ids.addId(cvd.getId());
        cvdUpdateAppLayer.setCvdIds(ids);
        final ArrayOfImageId appLayerIds = new ArrayOfImageId();
        appLayerIds.addImageId(appLayer.getImageId());
        cvdUpdateAppLayer.setAddAppLayerImageIds(appLayerIds);
        cvdUpdateAppLayer.setRemoveAppLayerImageIds(null);
        cvdUpdateAppLayer.setIgnoreWarnings(true);

        final Cvd_UpdateAppLayerResponse assignResponse =
client.cvd_UpdateAppLayer(cvdUpdateAppLayer);
        final BatchResult assignResult = assignResponse.getCvd_UpdateAppLayerResult();

        final OperationResult[] results = assignResult.getResults().getOperationResult();
        if (results.length == 0) {
            throw new Exception("There is no result for cvd_UpdateAppLayer");
        }

        final OperationResult result = results[0];

        if (result.getSuccess() == true) {
            System.out.println(String.format("Cvd update applayer begin successfully for Cvd(Id=
%d)", cvd.getId()
                    .getIdValue()));
        } else {
            System.out.println(String.format("Cvd update applayer begin failed for Cvd(Id=%d),
fault is: %s", cvd
                    .getId().getIdValue(), result.getFault().getMessage()));
        }
        /*Wait for app layer update complete*/
        waitForAssignmentComplete("AssignAppLayer", cvd.getId());
    }

    private void waitForAssignmentComplete(final String taskName, final Id cvdId) throws
Exception {
        final QueryFilterEquals taskNameFilter = new QueryFilterEquals();
        taskNameFilter.setField(FilterField.ASSIGNMENT_TASK_NAME);
        taskNameFilter.setValue(taskName);

        final QueryFilterEquals cvdIdFilter = new QueryFilterEquals();
        cvdIdFilter.setField(FilterField.ASSIGNMENT_CVD_ID);
        cvdIdFilter.setValue(cvdId);

        final QueryFilterAnd andFilter = new QueryFilterAnd();
        final ArrayOfQueryFilter filterArr = new ArrayOfQueryFilter();
        filterArr.addQueryFilter(taskNameFilter);
        filterArr.addQueryFilter(cvdIdFilter);
        andFilter.setFilters(filterArr);

        final QueryDefinition queryDefinition = new QueryDefinition();
        queryDefinition.setFilter(andFilter);
        queryDefinition.setPage(1);

        final Assignment_Query assignmentQuery = new Assignment_Query();
        assignmentQuery.setQueryDefinition(queryDefinition);
```

```
        while (true) {
            final Assignment_QueryResponse assignmentResponse =
client.assignment_Query(assignmentQuery);
            final QueryResult assignmentResult = assignmentResponse.getAssignment_QueryResult();
            final Object[] assignmentArr = assignmentResult.getElements().getAnyType();
            if (IsAssignmentDone(assignmentArr)) {
                break;
            }
            TimeUnit.MINUTES.sleep(2);
        }
    }

    private boolean IsAssignmentDone(final Object[] assignments) {
        if (assignments.length == 0) {
            return false;
        }
        final HashSet<String> statusFailed = new HashSet<String>();
        statusFailed.add("Cancelled");
        statusFailed.add("Rejected");
        statusFailed.add("Blocked");
        final String statusDone = "Done";
        boolean assignmentDone = true;
        for (final Object assignment : assignments) {
            final AssignmentDetails assignmentDetails = (AssignmentDetails) assignment;
            if (statusFailed.contains(assignmentDetails.getStatus())) {
                System.out.println(String.format("Assignment failed for Cvd(Id=%d), operation
failed.",
                        assignmentDetails.getCvdId().getIdValue()));
                return true;
            }
            assignmentDone &= assignmentDetails.getStatus().equals(statusDone);
        }
        if (assignmentDone) {
            System.out.println("Assignment done, operation succeed.");
        }
        return assignmentDone;
    }

    // Configure client to skip certificate validation and support Http session
    private void configureClient() throws NoSuchAlgorithmException, KeyManagementException {
        final SSLContext sslCtx = SSLContext.getInstance("TLS");
        sslCtx.init(null, new TrustManager[] { new TrustAllTrustManager() }, null);
        client._getServiceClient()
                .getOptions()
                .setProperty(HTTPConstants.CUSTOM_PROTOCOL_HANDLER,
                        new Protocol("https", (ProtocolSocketFactory) new
SSLProtocolSocketFactory(sslCtx), 7443));
        final HttpState httpState = new HttpState();
        client._getServiceClient().getOptions()
                .setProperty(org.apache.axis2.transport.http.HTTPConstants.CACHED_HTTP_STATE,
httpState);
    }
}
```

Index

A

API, performance **47**

C

configuration **49, 50**

D

development environment
- C# and .NET 4.5 **8**
- C# and .NET 4.0 **9**
- Java and .NET 4.0 **10**
- Java and .NET 4.5 **9**
- setting up **7**

G

glossary **5**

I

intended audience **5**

L

logging **49, 50**

M

methods **13**

P

performance **47**

permissions **49**

R

roles **49**

S

sample applications
- C# **51**
- Java **64**

W

WCF HTTP activation
- Windows Server 2008 R2 **7**
- Windows Server 2012 **8**