You can find the most up-to-date technical documentation on the VMware Web site at:
http://www.vmware.com/support/
The VMware Web site also provides the latest product updates.
If you have comments about this documentation, submit your feedback to:
docfeedback@vmware.com
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

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 -i ru 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.
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

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:

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

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

```csharp
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**

2. Unzip the file `axis2-1.x.y` to a folder and set the environment variable `%AXIS2_HOME%` to the path of this folder.
4 Generate Java classes with the following command:

```java
%AXIS2_HOME%/bin/wsdl2java -uri MitService.wsdl -p com.vmware.mirage.mit
```

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

```java
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**

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:

```java
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);
```
login.setPassword("password");
client.login(login);

//Perform tasks
Mirage API has 23 methods.

**Table 2-1. Mirage API Methods**

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POLICY_ID</td>
<td>Id</td>
<td>Id of the policy.</td>
</tr>
<tr>
<td>POLICY_NAME</td>
<td>string</td>
<td>The name of the policy.</td>
</tr>
<tr>
<td>POLICY_IMAGEID</td>
<td>ImageId</td>
<td>ImageId of the policy.</td>
</tr>
</tbody>
</table>

- Page
  Starts at 1.

Return:
- QueryResult
  The type of element is PolicyDetails.
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:

```csharp
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);
```

PendingDevice_Query

PendingDevice_Query queries pending devices.
Input:
- queryDefinition
- Filter

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVICE_CONNECTION_STATE</td>
<td>bool</td>
<td>The connection state of the device.</td>
</tr>
<tr>
<td>DEVICE_ID</td>
<td>Id</td>
<td>The Id of the device.</td>
</tr>
<tr>
<td>DEVICE_MODEL_NAME</td>
<td>string</td>
<td>The model name of the device.</td>
</tr>
<tr>
<td>DEVICE_NAME</td>
<td>string</td>
<td>The name of the device.</td>
</tr>
<tr>
<td>DEVICE_OS</td>
<td>string</td>
<td>The OS version of the device.</td>
</tr>
<tr>
<td>DEVICE_OS_VERSION</td>
<td>OsVersion</td>
<td>The OS version of the device.</td>
</tr>
<tr>
<td>DEVICE_VENDOR_NAME</td>
<td>string</td>
<td>The vendor name of the device.</td>
</tr>
</tbody>
</table>

The DEVICE_OS filter field supports the following string filter values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;XP&quot;</td>
<td>Windows XP</td>
</tr>
<tr>
<td>&quot;XP_EMBEDDED&quot;</td>
<td>Windows XP Embedded</td>
</tr>
<tr>
<td>&quot;VISTA&quot;</td>
<td>Windows Vista x86</td>
</tr>
<tr>
<td>&quot;VISTAX64&quot;</td>
<td>Windows Vista x64</td>
</tr>
<tr>
<td>&quot;WIN7&quot;</td>
<td>Windows 7 x86</td>
</tr>
<tr>
<td>&quot;WIN7_EMBEDDED&quot;</td>
<td>Windows 7 x86 Embedded</td>
</tr>
<tr>
<td>&quot;WIN7X64&quot;</td>
<td>Windows 7 x64</td>
</tr>
<tr>
<td>&quot;WIN7_EMBEDDEDX64&quot;</td>
<td>Windows 7 x64 Embedded</td>
</tr>
<tr>
<td>&quot;WIN8_0&quot;</td>
<td>Windows 8 x86</td>
</tr>
<tr>
<td>&quot;WIN8_0X64&quot;</td>
<td>Windows 8 x64</td>
</tr>
<tr>
<td>&quot;WIN8_1&quot;</td>
<td>Windows 8.1 x86</td>
</tr>
<tr>
<td>&quot;WIN8_1X64&quot;</td>
<td>Windows 8.1 x64</td>
</tr>
<tr>
<td>&quot;WINPE&quot;</td>
<td>Windows PE 5</td>
</tr>
</tbody>
</table>

- Page
  Starts at 1.

Return:
- QueryResult
  The type of element is DeviceDetails.

Example:

```java
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVD_CLIENT_STATE</td>
<td>string</td>
<td>The client state of the CVD.</td>
</tr>
<tr>
<td>CVD_DEVICE_CLIENT_STATE (Deprecated)</td>
<td>ClientState</td>
<td>To filter by client state, see CVD_CLIENT_STATE.</td>
</tr>
<tr>
<td>CVD_DEVICE_CONNECTION_STATE</td>
<td>bool</td>
<td>The connection state of the CVD.</td>
</tr>
<tr>
<td>CVD_DEVICE_ID</td>
<td>Id</td>
<td>The device Id of the CVD.</td>
</tr>
<tr>
<td>CVD_DEVICE_OS</td>
<td>string</td>
<td>The device OS version of the CVD.</td>
</tr>
<tr>
<td>CVD_ID</td>
<td>Id</td>
<td>The Id of the CVD.</td>
</tr>
<tr>
<td>CVD_MACHINE_VERSION</td>
<td>ImageVersion</td>
<td>The machine version of the CVD.</td>
</tr>
<tr>
<td>CVD_NAME</td>
<td>string</td>
<td>The name of the CVD.</td>
</tr>
<tr>
<td>CVD_POLICY_ID</td>
<td>Id</td>
<td>The policy Id of the CVD.</td>
</tr>
<tr>
<td>CVD_POLICY_IMAGEID</td>
<td>ImageId</td>
<td>The policy ImageId of the CVD.</td>
</tr>
<tr>
<td>CVD_POLICY_NAME</td>
<td>string</td>
<td>The policy name of the CVD.</td>
</tr>
<tr>
<td>CVD_PROGRESS</td>
<td>long</td>
<td>The operation progress of the CVD.</td>
</tr>
<tr>
<td>CVD_USER_NAME</td>
<td>string</td>
<td>The user name of the CVD.</td>
</tr>
</tbody>
</table>

The CVD_CLIENT_STATE filter field supports the following string filter values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Idle&quot;</td>
<td>Idle state.</td>
</tr>
<tr>
<td>&quot;PendingReboot&quot;</td>
<td>Pending reboot state.</td>
</tr>
<tr>
<td>&quot;ForceReboot&quot;</td>
<td>Force reboot state.</td>
</tr>
<tr>
<td>&quot;UploadInitializing&quot;</td>
<td>Upload initializing state.</td>
</tr>
<tr>
<td>&quot;Uploading&quot;</td>
<td>Uploading state.</td>
</tr>
<tr>
<td>&quot;RestorePrefetch&quot;</td>
<td>Restore pre-fetching states.</td>
</tr>
<tr>
<td>&quot;RestoreStreaming&quot;</td>
<td>Restore streaming states.</td>
</tr>
<tr>
<td>&quot;UpdateLayers&quot;</td>
<td>Downloading image states.</td>
</tr>
<tr>
<td>&quot;RebasePrefetching&quot;</td>
<td>Restore and base layer update pre-fetching states.</td>
</tr>
<tr>
<td>&quot;Migration&quot;</td>
<td>Migration states.</td>
</tr>
<tr>
<td>&quot;DriverLibraryUpdate&quot;</td>
<td>Driver library updating states.</td>
</tr>
<tr>
<td>&quot;RestoreUserData&quot;</td>
<td>Restore and profile pre-fetching states.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>&quot;DeviceProvisioning&quot;</td>
<td>Device provisioning states.</td>
</tr>
<tr>
<td>&quot;AppLayerRecording&quot;</td>
<td>App layer recording states.</td>
</tr>
<tr>
<td>&quot;AppLayerCapture&quot;</td>
<td>App layer capture states.</td>
</tr>
<tr>
<td>&quot;Synchronizing&quot;</td>
<td>Synchronizing states.</td>
</tr>
<tr>
<td>&quot;PendingUpgrade&quot;</td>
<td>Pending upgrade states.</td>
</tr>
<tr>
<td>&quot;PendingRestore&quot;</td>
<td>Pending restore states.</td>
</tr>
<tr>
<td>&quot;PendingAssignment&quot;</td>
<td>Pending assignment states.</td>
</tr>
<tr>
<td>&quot;Suspended&quot;</td>
<td>Suspended states.</td>
</tr>
<tr>
<td>&quot;WaitingForService&quot;</td>
<td>Waiting for service states.</td>
</tr>
<tr>
<td>&quot;Throttled&quot;</td>
<td>Throttled states.</td>
</tr>
<tr>
<td>&quot;PendingUserAuthentication&quot;</td>
<td>Pending user authentication states.</td>
</tr>
<tr>
<td>&quot;Snoozed&quot;</td>
<td>Snoozed states.</td>
</tr>
<tr>
<td>&quot;BranchReflectorIsBusy&quot;</td>
<td>Branch reflector is busy states.</td>
</tr>
<tr>
<td>&quot;WaitingDeviceProvisioning&quot;</td>
<td>Waiting for device provisioning states.</td>
</tr>
<tr>
<td>&quot;RecordingAppLayer&quot;</td>
<td>Recording app layer states.</td>
</tr>
<tr>
<td>&quot;NoPotentialBranchReflectors&quot;</td>
<td>No Potential Branch Reflectors states.</td>
</tr>
<tr>
<td>&quot;BranchReflectorIsCaching&quot;</td>
<td>Branch Reflector Is Caching states.</td>
</tr>
<tr>
<td>&quot;PendingGatewayLogin&quot;</td>
<td>Pending authentication from Gateway states.</td>
</tr>
<tr>
<td>&quot;PendingRebootStreaming&quot;</td>
<td>Pending reboot for restore streaming states.</td>
</tr>
<tr>
<td>&quot;RestoreInitializing&quot;</td>
<td>Restore initializing states.</td>
</tr>
<tr>
<td>&quot;UpdateLayersInitializing&quot;</td>
<td>Download image initializing states.</td>
</tr>
<tr>
<td>&quot;RebasePrefetchInitializing&quot;</td>
<td>Restore and base layer update initializing states.</td>
</tr>
<tr>
<td>&quot;MigrationInitializing&quot;</td>
<td>Migration initializing states.</td>
</tr>
<tr>
<td>&quot;DriverLibraryUpdateInitializing&quot;</td>
<td>Driver library update initializing states.</td>
</tr>
<tr>
<td>&quot;RestoreUserDataInitializing&quot;</td>
<td>Restore user data initializing states.</td>
</tr>
<tr>
<td>&quot;DeviceProvisioningInitializing&quot;</td>
<td>Device provisioning initializing states.</td>
</tr>
<tr>
<td>&quot;AppLayerRecordingInitializing&quot;</td>
<td>App layer recording initializing states.</td>
</tr>
<tr>
<td>&quot;AppLayerCaptureInitializing&quot;</td>
<td>App layer capture initializing states.</td>
</tr>
<tr>
<td>&quot;SynchronizingInitializing&quot;</td>
<td>Synchronize initializing states.</td>
</tr>
<tr>
<td>&quot;RecordingAppLayerInitializing&quot;</td>
<td>Record app layer initializing states.</td>
</tr>
<tr>
<td>&quot;RestoreStreamingInitializing&quot;</td>
<td>Restore streaming initializing states.</td>
</tr>
<tr>
<td>&quot;UploadChangesFinalizing&quot;</td>
<td>Upload changes finalizing states.</td>
</tr>
<tr>
<td>&quot;RestorePrefetchFinalizing&quot;</td>
<td>Restore prefetching finalizing states.</td>
</tr>
<tr>
<td>&quot;UpdateLayersFinalizing&quot;</td>
<td>Download image finalizing states.</td>
</tr>
<tr>
<td>&quot;RebasePrefetchFinalizing&quot;</td>
<td>Restore and base layer update prefetching finalizing states.</td>
</tr>
<tr>
<td>&quot;MigrationFinalizing&quot;</td>
<td>Migration finalizing states.</td>
</tr>
<tr>
<td>&quot;DriverLibraryUpdateFinalizing&quot;</td>
<td>Driver library update finalizing states.</td>
</tr>
<tr>
<td>&quot;RestoreUserDataFinalizing&quot;</td>
<td>Restore user data finalizing states.</td>
</tr>
<tr>
<td>&quot;DeviceProvisioningFinalizing&quot;</td>
<td>Device provisioning finalizing states.</td>
</tr>
</tbody>
</table>
The CVD_DEVICE_OS filter field supports the following string filter values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;XP&quot;</td>
<td>Windows XP</td>
</tr>
<tr>
<td>&quot;XP_EMBEDDED&quot;</td>
<td>Windows XP Embedded</td>
</tr>
<tr>
<td>&quot;VISTA&quot;</td>
<td>Windows Vista x86</td>
</tr>
<tr>
<td>&quot;VISTAX64&quot;</td>
<td>Windows Vista x64</td>
</tr>
<tr>
<td>&quot;WIN7&quot;</td>
<td>Windows 7 x86</td>
</tr>
<tr>
<td>&quot;WIN7_EMBEDDED&quot;</td>
<td>Windows 7 x86 Embedded</td>
</tr>
<tr>
<td>&quot;WIN7X64&quot;</td>
<td>Windows 7 x64</td>
</tr>
<tr>
<td>&quot;WIN7X64_EMBEDDED&quot;</td>
<td>Windows 7 x64 Embedded</td>
</tr>
<tr>
<td>&quot;WIN8_0&quot;</td>
<td>Windows 8 x86</td>
</tr>
<tr>
<td>&quot;WIN8_0X64&quot;</td>
<td>Windows 8 x64</td>
</tr>
<tr>
<td>&quot;WIN8_1&quot;</td>
<td>Windows 8.1 x86</td>
</tr>
<tr>
<td>&quot;WIN8_1X64&quot;</td>
<td>Windows 8.1 x64</td>
</tr>
<tr>
<td>&quot;WINPE&quot;</td>
<td>Windows PE 5</td>
</tr>
</tbody>
</table>

Page

Starts at 1.

Return:

- QueryResult
  - The type of element is CvdDetails.

Example:

```csharp
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE_IMAGE_LAYER_NAME</td>
<td>string</td>
<td>The name of the base layer.</td>
</tr>
<tr>
<td>BASE_IMAGE_LAYER_ID</td>
<td>Id</td>
<td>The Id of the ImageId of the base layer.</td>
</tr>
<tr>
<td>BASE_IMAGE_LAYER_IMAGEID</td>
<td>ImageId</td>
<td>The ImageId of the base layer.</td>
</tr>
</tbody>
</table>

- 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**
      - **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:**

```java
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 imageld 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.
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",
    }
}
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:

```csharp
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:

```csharp
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:

```csharp
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:

```csharp
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)
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
  pendingDevices 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:

```csharp
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
    - 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

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVD_CLIENT_STATE</td>
<td>string</td>
<td>The client state of the CVD.</td>
</tr>
<tr>
<td>CVD_DEVICE_CLIENT_STATUS</td>
<td>ClientState</td>
<td>To filter with client state, see CVD_CLIENT_STATE.</td>
</tr>
<tr>
<td>CVD_DEVICE_CONNECTION_STATUS</td>
<td>bool</td>
<td>The connection state of the CVD.</td>
</tr>
<tr>
<td>CVD_DEVICE_ID</td>
<td>Id</td>
<td>The device Id of the CVD.</td>
</tr>
<tr>
<td>CVD_DEVICE_OS</td>
<td>string</td>
<td>The device OS version of the CVD.</td>
</tr>
<tr>
<td>CVD_ID</td>
<td>Id</td>
<td>The Id of the CVD.</td>
</tr>
<tr>
<td>CVD_MACHINE_VERSION</td>
<td>ImageVersion</td>
<td>The machine version of the CVD.</td>
</tr>
<tr>
<td>CVD_NAME</td>
<td>string</td>
<td>The name of the CVD.</td>
</tr>
<tr>
<td>CVD_POLICY_ID</td>
<td>Id</td>
<td>The policy Id of the CVD.</td>
</tr>
<tr>
<td>CVD_POLICY_IMAGEID</td>
<td>ImageId</td>
<td>The policy Imageld of the CVD.</td>
</tr>
<tr>
<td>CVD_POLICY_NAME</td>
<td>string</td>
<td>The policy name of the CVD.</td>
</tr>
<tr>
<td>CVD_PROGRESS</td>
<td>long</td>
<td>The operation progress of the CVD.</td>
</tr>
<tr>
<td>CVD_USER_NAME</td>
<td>string</td>
<td>The user name of the CVD.</td>
</tr>
</tbody>
</table>

The CVD_CLIENT_STATE filter field supports the following string filter values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Idle&quot;</td>
<td>Idle state.</td>
</tr>
<tr>
<td>&quot;PendingReboot&quot;</td>
<td>Pending reboot state.</td>
</tr>
<tr>
<td>&quot;ForceReboot&quot;</td>
<td>Force reboot state.</td>
</tr>
<tr>
<td>&quot;UploadInitializing&quot;</td>
<td>Upload initializing state.</td>
</tr>
<tr>
<td>&quot;RestorePrefetch&quot;</td>
<td>Restore prefetching states.</td>
</tr>
<tr>
<td>&quot;RestoreStreaming&quot;</td>
<td>Restore streaming states.</td>
</tr>
<tr>
<td>&quot;UpdateLayers&quot;</td>
<td>Downloading image states.</td>
</tr>
<tr>
<td>&quot;RebasePrefetching&quot;</td>
<td>Restore and base layer update prefetching states.</td>
</tr>
<tr>
<td>&quot;Migration&quot;</td>
<td>Migration states.</td>
</tr>
<tr>
<td>&quot;DriverLibraryUpdate&quot;</td>
<td>Driver library updating states.</td>
</tr>
<tr>
<td>&quot;RestoreUserData&quot;</td>
<td>Restore and profile prefetching states.</td>
</tr>
<tr>
<td>&quot;DeviceProvisioning&quot;</td>
<td>Device provisioning states.</td>
</tr>
<tr>
<td>&quot;AppLayerCapture&quot;</td>
<td>App layer capture states.</td>
</tr>
<tr>
<td>&quot;Synchronizing&quot;</td>
<td>Synchronizing states.</td>
</tr>
<tr>
<td>&quot;PendingUpgrade&quot;</td>
<td>Pending upgrade states.</td>
</tr>
<tr>
<td>&quot;PendingRestore&quot;</td>
<td>Pending restore states.</td>
</tr>
<tr>
<td>&quot;PendingAssignment&quot;</td>
<td>Pending assignment states.</td>
</tr>
</tbody>
</table>

VMware Mirage API Programming Guide
32 VMware, Inc.
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Suspended&quot;</td>
<td>Suspended states.</td>
</tr>
<tr>
<td>&quot;WaitingForService&quot;</td>
<td>Waiting for service states.</td>
</tr>
<tr>
<td>&quot;Throttled&quot;</td>
<td>Throttled states.</td>
</tr>
<tr>
<td>&quot;PendingUserAuthentication&quot;</td>
<td>Pending user authentication states.</td>
</tr>
<tr>
<td>&quot;Snoozed&quot;</td>
<td>Snoozed states.</td>
</tr>
<tr>
<td>&quot;BranchReflectorIsBusy&quot;</td>
<td>Branch refector is busy states.</td>
</tr>
<tr>
<td>&quot;WaitingDeviceProvisioning&quot;</td>
<td>Waiting for device provision states.</td>
</tr>
<tr>
<td>&quot;RecordingAppLayer&quot;</td>
<td>Recording app layer states.</td>
</tr>
<tr>
<td>&quot;NoPotentialBranchReflectors&quot;</td>
<td>No potential branch reflectors states.</td>
</tr>
<tr>
<td>&quot;BranchReflectorIsCaching&quot;</td>
<td>Branch refector is caching states.</td>
</tr>
<tr>
<td>&quot;PendingGatewayLogin&quot;</td>
<td>Pending authentication from Gateway states.</td>
</tr>
<tr>
<td>&quot;PendingRebootStreaming&quot;</td>
<td>Pending reboot for restore streaming states.</td>
</tr>
<tr>
<td>&quot;RestoreInitializing&quot;</td>
<td>Restore initializing states.</td>
</tr>
<tr>
<td>&quot;UpdateLayersInitializing&quot;</td>
<td>Download image initializing states.</td>
</tr>
<tr>
<td>&quot;RebasePrefetchInitializing&quot;</td>
<td>Restore and base layer update initializing states.</td>
</tr>
<tr>
<td>&quot;MigrationInitializing&quot;</td>
<td>Migration initializing states.</td>
</tr>
<tr>
<td>&quot;DriverLibraryUpdateInitializing&quot;</td>
<td>Driver library update initializing states.</td>
</tr>
<tr>
<td>&quot;RestoreUserDataInitializing&quot;</td>
<td>Restore user data initializing states.</td>
</tr>
<tr>
<td>&quot;DeviceProvisioningInitializing&quot;</td>
<td>Device provisioning initializing states.</td>
</tr>
<tr>
<td>&quot;AppLayerRecordingInitializing&quot;</td>
<td>App layer recording initializing states.</td>
</tr>
<tr>
<td>&quot;AppLayerCaptureInitializing&quot;</td>
<td>App layer capture initializing states.</td>
</tr>
<tr>
<td>&quot;SynchronizingInitializing&quot;</td>
<td>Synchronizing initializing states.</td>
</tr>
<tr>
<td>&quot;RecordingAppLayerInitializing&quot;</td>
<td>Record app layer initializing states.</td>
</tr>
<tr>
<td>&quot;RestoreStreamingInitializing&quot;</td>
<td>Restore streaming initializing states.</td>
</tr>
<tr>
<td>&quot;UploadChangesFinalizing&quot;</td>
<td>Upload changes finalizing states.</td>
</tr>
<tr>
<td>&quot;RestorePrefetchFinalizing&quot;</td>
<td>Restore prefetch finalizing states.</td>
</tr>
<tr>
<td>&quot;UpdateLayersFinalizing&quot;</td>
<td>Download image finalizing states.</td>
</tr>
<tr>
<td>&quot;RebasePrefetchFinalizing&quot;</td>
<td>Restore and base layer update prefetch finalizing states.</td>
</tr>
<tr>
<td>&quot;MigrationFinalizing&quot;</td>
<td>Migration finalizing states.</td>
</tr>
<tr>
<td>&quot;DriverLibraryUpdateFinalizing&quot;</td>
<td>Driver library update finalizing states.</td>
</tr>
<tr>
<td>&quot;RestoreUserDataFinalizing&quot;</td>
<td>Restore user data finalizing states.</td>
</tr>
<tr>
<td>&quot;DeviceProvisioningFinalizing&quot;</td>
<td>Device provisioning finalizing states.</td>
</tr>
<tr>
<td>&quot;AppLayerRecordingFinalizing&quot;</td>
<td>App layer recording finalizing states.</td>
</tr>
<tr>
<td>&quot;AppLayerCaptureFinalizing&quot;</td>
<td>App layer capture finalizing states.</td>
</tr>
<tr>
<td>&quot;RecordingAppLayerFinalizing&quot;</td>
<td>Recording app layer finalizing states.</td>
</tr>
<tr>
<td>&quot;RestoreStreamingFinalizing&quot;</td>
<td>Restore streaming finalizing states.</td>
</tr>
<tr>
<td>&quot;SynchronizingFinalizing&quot;</td>
<td>Synchronizing finalizing states.</td>
</tr>
</tbody>
</table>
The CVD_DEVICE_OS filter field supports the following string filter values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;XP&quot;</td>
<td>Windows XP</td>
</tr>
<tr>
<td>&quot;XP_EMBEDDED&quot;</td>
<td>Windows XP Embedded</td>
</tr>
<tr>
<td>&quot;VISTA&quot;</td>
<td>Windows Vista x86</td>
</tr>
<tr>
<td>&quot;VISTAX64&quot;</td>
<td>Windows Vista x64</td>
</tr>
<tr>
<td>&quot;WIN7&quot;</td>
<td>Windows 7 x86</td>
</tr>
<tr>
<td>&quot;WIN7_EMBEDDED&quot;</td>
<td>Windows 7 x86 Embedded</td>
</tr>
<tr>
<td>&quot;WIN7X64&quot;</td>
<td>Windows 7 x64</td>
</tr>
<tr>
<td>&quot;WIN7_EMBEDDEDX64&quot;</td>
<td>Windows 7 x64 Embedded</td>
</tr>
<tr>
<td>&quot;WIN8_0&quot;</td>
<td>Windows 8 x86</td>
</tr>
<tr>
<td>&quot;WIN8_0X64&quot;</td>
<td>Windows 8 x64</td>
</tr>
<tr>
<td>&quot;WIN8_1&quot;</td>
<td>Windows 8.1 x86</td>
</tr>
<tr>
<td>&quot;WIN8_1X64&quot;</td>
<td>Windows 8.1 x64</td>
</tr>
<tr>
<td>&quot;WINPE&quot;</td>
<td>Windows PE 5</td>
</tr>
</tbody>
</table>

- Page
  Starts at 1.

Return:
- QueryResult
  The type of element is CvdDetails.

Example:
```csharp
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,
}

OsMigrationCvd_QueryDownloadOnly

OsMigrationCvd_QueryDownloadOnly queries the CVD Ids for download only migration.

Input:

- QueryDefinition

- Filter

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVD_ID</td>
<td>Id</td>
<td>The Id of the CVD.</td>
</tr>
<tr>
<td>DOWNLOAD_ONLY_MIGRATION_STATUS</td>
<td>string</td>
<td>The status of the download only migration</td>
</tr>
</tbody>
</table>

- Page

  Starts at 1.

Return:

- QueryResult

  The type of element is CvdDownloadOnlyMigrationDetails.

Example:

```csharp
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}".m, migration.CvdId.IdValue, migration.DownloadOnlyMigrationStatus);
    total += result.Elements.Length;
    page ++;
    continue;
} break;
}
Console.WriteLine("Total number: {0}", total);
Mirage API has pre-defined types that you can use to develop applications.

The types are in the following categories. For more information, see the Mirage API Reference:
- Query
- Fault
- Service
- Other

This chapter includes the following topics:
- “Query Types,” on page 37
- “Fault Types,” on page 43
- “Service Types,” on page 43
- “Other Types,” on page 44

Query Types
Mirage API has various query types for the query methods.

QueryDefinition
QueryDefinition sets a filter and page number. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter</td>
<td>QueryFilter</td>
<td>See the QueryFilter type.</td>
</tr>
<tr>
<td>Page</td>
<td>int</td>
<td>The query result comes back in pages. This field indicates which page should be returned. The page index starts from 1, not 0.</td>
</tr>
</tbody>
</table>

QueryFilter
QueryFilter is the abstract base class for all other filters. There are 2 types of query filters: basic and composite. The basic type has one filter. The composite type has multiple filters. QueryFilterEquals, QueryFilterNotEquals, QueryFilterBeginsWith, QueryFilterEndsWith, and QueryFilterContains are basic filters. QueryFilterAnd and QueryFilterOr are composite filters.

QueryFilterEquals
QueryFilterEquals specifies an equality filter. It has the following properties:
<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>FilterField</td>
<td>See the FilterField type.</td>
</tr>
<tr>
<td>Value</td>
<td>object</td>
<td>The value to be compared against.</td>
</tr>
</tbody>
</table>

Sample code to query a pending device's OS version:

```csharp
QueryDefinition query = new QueryDefinition
{
    Filter = new QueryFilterEquals
    {
        Field = FilterField.DEVICE_OS,
        Value = "WIN7x64"
    },
    Page = 1
};
QueryResult result = client.PendingDevice_Query(query);
```

**QueryFilterNotEquals**

QueryFilterNotEquals specifies an inequality filter. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>FilterField</td>
<td>See the FilterField type.</td>
</tr>
<tr>
<td>Value</td>
<td>object</td>
<td>The value to be compared against.</td>
</tr>
</tbody>
</table>

Sample code to query a pending device's OS version:

```csharp
QueryDefinition query = new QueryDefinition
{
    Filter = new QueryFilterNotEquals
    {
        Field = FilterField.DEVICE_OS,
        Value = "WIN7x64"
    },
    Page = 1
};
QueryResult result = client.PendingDevice_Query(query);
```

**QueryFilterBeginsWith**

QueryFilterBeginsWith specifies a starting value for the field in the filter. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>FilterField</td>
<td>See the FilterField type.</td>
</tr>
<tr>
<td>Value</td>
<td>object</td>
<td>The value that the specified field must begin with.</td>
</tr>
</tbody>
</table>

Sample code to query a CVD with a name that begins with "John_":

```csharp
QueryDefinition query = new QueryDefinition
{
    Filter = new QueryFilterBeginsWith
    {
        Field = FilterField.CVD_NAME,
        Value = "John_"
    },
    Page = 1
};
QueryResult result = client.PendingDevice_Query(query);
```
QueryFilterEndsWith

QueryFilterEndsWith specifies an ending value for the field in the filter. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>FilterField</td>
<td>See the FilterField type.</td>
</tr>
<tr>
<td>Value</td>
<td>object</td>
<td>The value that the specified field must end with.</td>
</tr>
</tbody>
</table>

Sample code to query a CVD with a name that ends with "Green":

```csharp
QueryDefinition query = new QueryDefinition
{
    Filter = new QueryFilterEndsWith
    {
        Field = FilterField.CVD_NAME,
        Value = "Green"
    },
    Page = 1
};
QueryResult result = client.Cvd_Query(query);
```

QueryFilterContains

QueryFilterContains specifies a value that the field in the filter must contain. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>FilterField</td>
<td>See the FilterField type.</td>
</tr>
<tr>
<td>Value</td>
<td>object</td>
<td>The value that the specified field must contain</td>
</tr>
</tbody>
</table>

Sample code to query a CVD with a name that contains "VM":

```csharp
QueryDefinition query = new QueryDefinition
{
    Filter = new QueryFilterContains
    {
        Field = FilterField.CVD_NAME,
        Value = "VM"
    },
    Page = 1
};
QueryResult result = client.Cvd_Query(query);
```

QueryFilterAnd

QueryFilterAnd specifies multiple filters that have an "and" relationship. It has the following property:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filters</td>
<td>QueryFilters[]</td>
<td>All the filters joined by the &quot;and&quot; operation.</td>
</tr>
</tbody>
</table>
Sample code to query a pending device whose OS version is Windows 7 x86 and that belongs to user "John":

```csharp
QueryFilter osFilter = new QueryFilterEquals
{
    Field = FilterField.DEVICE_OS,
    Value = "WIN7"
};
QueryFilter nameFilter = new QueryFilterContains
{
    Field = FilterField.DEVICE_USER_NAME,
    Value = "John"
};
QueryDefinition query = new QueryDefinition
{
    Filter = new QueryFilterAnd
    {
        Filters = new[] { osFilter, nameFilter }
    },
    Page = 1
};
QueryResult result = client.PendingDevice_Query(query);
```

**QueryFilterOr**

QueryFilterOr specifies multiple filters that have an "or" relationship. It has the following property:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filters</td>
<td>QueryFilters[]</td>
<td>All the filters joined by the &quot;or&quot; operation.</td>
</tr>
</tbody>
</table>

Sample code to find devices that run Windows 7 or Windows XP:

```csharp
QueryFilter win7Filter = new QueryFilterEquals
{
    Field = FilterField.DEVICE_OS,
    Value = "WIN7"
};
QueryFilter xpFilter = new QueryFilterEquals
{
    Field = FilterField.DEVICE_OS,
    Value = "XP"
};
QueryDefinition query = new QueryDefinition
{
    Filter = new QueryFilterOr
    {
        Filters = new[] { win7Filter, xpFilter }
    },
    Page = 1
};
QueryResult result = client.PendingDevice_Query(query);
```

**FilterField**

FilterField lists all the fields that can be filtered on. Each filter has its own allowed basic filter types and filter value type. For example, VOLUME_ID can be used by QueryFilterEquals or QueryFilterNotEquals but not other query filters. Its filter value should be of type Id.
<table>
<thead>
<tr>
<th>Filter Field</th>
<th>Allowed Filters</th>
<th>Filter Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOLUME_ID</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>Id</td>
</tr>
<tr>
<td>VOLUME_NAME</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>VOLUME_PATH</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>DEVICE_ID</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>Id</td>
</tr>
<tr>
<td>DEVICE_NAME</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>DEVICE_USER_NAME</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>DEVICE_MODEL_NAME</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>DEVICE_VENDOR_NAME</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>DEVICE_OS_VERSION (Deprecated)</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>OsVersion</td>
</tr>
<tr>
<td>DEVICE_CONNECTION_STATE</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>bool (connected -- true)</td>
</tr>
<tr>
<td>CVD_ID</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>Id</td>
</tr>
<tr>
<td>CVD_DEVICE_ID</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>Id</td>
</tr>
<tr>
<td>CVD_POLICY_ID</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>Id</td>
</tr>
<tr>
<td>CVD_NAME</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>CVD_USER_NAME</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>Filter Field</td>
<td>Allowed Filters</td>
<td>Filter Value Type</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>CVD_POLICY_NAME</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>CVD_DEVICE_CONNECTION_STATE</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>bool (connected -- true)</td>
</tr>
<tr>
<td>CVDDEVICE_CLIENT_STATUS (Deprecated)</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>ClientState</td>
</tr>
<tr>
<td>CVD_PROGRESS</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>Long</td>
</tr>
<tr>
<td>CVD_MACHINE_VERSION</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>ImageVersion</td>
</tr>
<tr>
<td>POLICY_ID</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>Id</td>
</tr>
<tr>
<td>POLICY_NAME</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>BASE_IMAGE_LAYER_ID</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>Id</td>
</tr>
<tr>
<td>BASE_IMAGE_LAYER_TYPE</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>Int</td>
</tr>
<tr>
<td>BASE_IMAGE_LAYER_NAME</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>BASE_IMAGE_LAYER_IMAGEID</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>Id</td>
</tr>
<tr>
<td>COLLECTION_DESCRIPTION</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>COLLECTION_ID</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>Id</td>
</tr>
<tr>
<td>COLLECTION_NAME</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith, QueryFilterEndsWith</td>
<td>string</td>
</tr>
<tr>
<td>CVD_CLIENT_STATE</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>string</td>
</tr>
<tr>
<td>CVD_POLICY_IMAGEID</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>ImageId</td>
</tr>
<tr>
<td>DEVICE_OS</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>string</td>
</tr>
<tr>
<td>POLICY_IMAGEID</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>ImageId</td>
</tr>
<tr>
<td>Filter Field</td>
<td>Allowed Filters</td>
<td>Filter Value Type</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>DOWNLOAD_ONLY_MIGRATION_STATUS</td>
<td>QueryFilterEquals</td>
<td>string</td>
</tr>
<tr>
<td>CVD_DEVICE_OS</td>
<td>QueryFilterEquals, QueryFilterNotEquals</td>
<td>string</td>
</tr>
</tbody>
</table>

**Fault Types**

Mirage API has various fault types to return faults.

**MethodFault**

This is a common fault. It contains a description of the error. It is also the base type of other faults.

**InvalidArgument**

The input parameters are invalid. For example, when you call Cvd_Query with the parameter queryDefinition set to null.

**InvalidLogin**

Authentication fails. For example, when you call Login with an invalid user name or password.

**NotAuthenticated**

The client has not called Login successfully before calling other methods.

**NotSupportedFault**

The input parameters are not supported. For example, calling OsMigration_QueryDownloadOnlyInProgress with queryDefinition containing a query filter.

**InvalidRequest**

Internal error.

**ManagementFault**

Internal error.

**RuntimeFault**

Internal error.

**AccessDenied**

The client calls failed due to a permissions issue.

**Service Types**

Mirage API has one service type.

**IMitService**

The interface of Mirage API.
Other Types

Mirage API has other types in addition to the query, fault, and service types.

Id

Id represents the identity of Mirage inventory objects such as volume, device, CVD, and so on. This type has one property, IdValue, which uniquely identifies an object. If two objects have the same IdValue, then they are the same object. For example:

```csharp
VolumeDetails volumeDetails1 = GetVolumeDetailsFromOneQuery();
VolumeDetails volumeDetails2 = GetVolumeDetailsFromAnotherQuery();
if (volumeDetails1.Id.IdValue == volumeDetails2.Id.IdValue)
{
    // volumeDetails1 and volumeDetails2 represent the same Volume
}
```

Id is used in the following types:
- VolumeDetails
- CvdDetails
- DeviceDetails
- ImageId
- ClientEvent
- CollectionDetails
- MigrationTarget
- CvdDownloadOnlyMigrationDetails

Note that Id of ImageId cannot represent the identity of ImageId.

PolicyDetails and LayerDetails do not have the Id field. For these two types, ImageId is often used as the identify of PolicyDetails or LayerDetails. Please refer to PolicyDetails and LayerDetails for a detailed description.

An Id object is needed when we use it as a filter value. IdValue cannot be used as the filter value directly. For example:

```csharp
QueryDefinition queryDefinition = new QueryDefinition
{
    Filter = new QueryFilterEquals
    {
        Field = FilterField.VOLUME_ID,
        /* Value = (long)2047300368 Incorrect */
        Value = new Id { IdValue = 2047300368 }
    },
    Page = 1 // Page starts from 1, not 0
};
QueryResult result = Client.Volume_Query(queryDefinition);
```

PolicyDetails

PolicyDetails represents information about a policy. It has the following properties:
Name represents the name of a policy. ImageId contains Id and Version of a policy. ImageId is often used when a method needs a policy. For example:

```java
Id[] deviceIds = GetDeviceIds();
PolicyDetails policyDetails = GetPolicyDetails();
VolumeDetails volumeDetails = GetVolumeDetails();
BatchResult result = PendingDevice_CreateNewCvd(deviceIds, policyDetails.ImageId, volumeDetails.Id);
```

**VolumeDetails**

VolumeDetails represents information about a volume. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Id</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

**DeviceDetails**

DeviceDetails represents information about a device. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Id</td>
<td>The identity of the device.</td>
</tr>
<tr>
<td>Name</td>
<td>string</td>
<td>The name of the device.</td>
</tr>
<tr>
<td>UserName</td>
<td>string</td>
<td>The device's user name including the domain name.</td>
</tr>
<tr>
<td>OsVersion ( Deprecated )</td>
<td>OsVersion</td>
<td>To get device Os, see Os.</td>
</tr>
<tr>
<td>Os</td>
<td>string</td>
<td>The OS version type of the device.</td>
</tr>
<tr>
<td>VendorName</td>
<td>string</td>
<td>The device's vendor name.</td>
</tr>
<tr>
<td>ModelName</td>
<td>string</td>
<td>The device's model name as reported by Windows. For example, Latitude E6430.</td>
</tr>
<tr>
<td>Connected</td>
<td>bool</td>
<td>Whether the device is connected to the server.</td>
</tr>
<tr>
<td>LastConnectedTime</td>
<td>DateTime</td>
<td>The last time the device connected to the server.</td>
</tr>
<tr>
<td>OperationTime</td>
<td>DateTime</td>
<td>The last operation time on the device.</td>
</tr>
</tbody>
</table>

**CvdDetails**

CvdDetails represents information about a CVD. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Id</td>
<td>Id of the CVD.</td>
</tr>
<tr>
<td>DeviceId</td>
<td>Id</td>
<td>Id of the device from which the CVD is created.</td>
</tr>
<tr>
<td>PolicyId ( Deprecated )</td>
<td>Id</td>
<td>To get policy ImageId, see PolicyImageId.</td>
</tr>
<tr>
<td>PolicyImageId</td>
<td>ImageId</td>
<td>ImageId of the policy when creating the CVD.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Connected</td>
<td>bool</td>
<td>Whether the device is connected to the server.</td>
</tr>
<tr>
<td>State (Deprecated)</td>
<td>ClientState</td>
<td>To get client state, see ClientState.</td>
</tr>
<tr>
<td>ClientState</td>
<td>string</td>
<td>The current state of the device.</td>
</tr>
<tr>
<td>Name</td>
<td>string</td>
<td>The name of the CVD.</td>
</tr>
<tr>
<td>UserName</td>
<td>string</td>
<td>The device’s user name including the domain name.</td>
</tr>
<tr>
<td>OperationProgress</td>
<td>long</td>
<td>The progress of the operation currently running on the CVD.</td>
</tr>
<tr>
<td>BaseImageId</td>
<td>ImageId</td>
<td>Base layer image used when creating the CVD.</td>
</tr>
<tr>
<td>LastConnectedTime</td>
<td>DateTime</td>
<td>The last time the device connected to the server.</td>
</tr>
<tr>
<td>LastUploadTime</td>
<td>DateTime</td>
<td>Last upload time of the device.</td>
</tr>
<tr>
<td>MachineVersion</td>
<td>ImageVersion</td>
<td>Version of the CVD which the device is using.</td>
</tr>
<tr>
<td>ClientEvents</td>
<td>ClientEvent</td>
<td>Array of events related to the CVD operation. Contains the events CLIENT_PROGRESS AND JOIN_DOMAIN only. There are at most 10 events, sorted by time in descending order.</td>
</tr>
<tr>
<td>Os</td>
<td>string</td>
<td>OS version of the device.</td>
</tr>
</tbody>
</table>

**QueryResult**

QueryResult represents the result of a query, such as AppLayer_Query, Cvd_Query, OsMigration_QueryDownloadOnlyCompleted, and so on. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>object []</td>
<td>Contains the detailed element returned by the query.</td>
</tr>
<tr>
<td>PageIndex</td>
<td>int</td>
<td>The page the elements reside in.</td>
</tr>
<tr>
<td>PageCount</td>
<td>int</td>
<td>Page count.</td>
</tr>
<tr>
<td>TotalElements</td>
<td>int</td>
<td>Element count.</td>
</tr>
<tr>
<td>nextPageAvailable</td>
<td>bool</td>
<td>Whether next page is available.</td>
</tr>
</tbody>
</table>

**Method**

<table>
<thead>
<tr>
<th>Method</th>
<th>Type of QueryResult Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppLayer_Query</td>
<td>LayerDetails</td>
</tr>
<tr>
<td>BaseLayer_Query</td>
<td>LayerDetails</td>
</tr>
<tr>
<td>Cvd_Query</td>
<td>CvdDetails</td>
</tr>
<tr>
<td>OsMigration_QueryDownloadOnlyCompleted</td>
<td>CvdDetails</td>
</tr>
<tr>
<td>OsMigration_QueryDownloadOnlyInProgress</td>
<td>CvdDetails</td>
</tr>
<tr>
<td>PendingDevice_Query</td>
<td>DeviceDetails</td>
</tr>
<tr>
<td>Policy_Query</td>
<td>PolicyDetails</td>
</tr>
<tr>
<td>Volume_Query</td>
<td>VolumeDetails</td>
</tr>
<tr>
<td>Collection_Query</td>
<td>CollectionDetails</td>
</tr>
</tbody>
</table>
Sample code to query CVDs, get QueryResult and convert the element to CvdDetails.

QueryResult queryResult = Client.Cvd_Query(new QueryDefinition { Filter = null, Page = 1 });
if (queryResult.Elements.Length != 0)
{
    CvdDetails cvdDetails = (CvdDetails)queryResult.Elements[0];
    // do your work with cvdDetails
}

**BatchResult**

BatchResult represents the result of methods for batch operations, such as PendingDevice_CreateNewCvd or OsMigration_Begin. These methods operate on multiple inventories. It has the following property:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Results</td>
<td>OperationResult</td>
<td>Operation result for an inventory. See OperationResult for details.</td>
</tr>
</tbody>
</table>

**OperationResult**

OperationResult represents the operation result for an inventory. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>bool</td>
<td>Whether the operation is successful.</td>
</tr>
<tr>
<td>Fault</td>
<td>MethodFault</td>
<td>Error message.</td>
</tr>
<tr>
<td>Result</td>
<td>Object</td>
<td>Result of the operation.</td>
</tr>
</tbody>
</table>

**MigrationTarget**

MigrationTarget represents information for a migration. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Id</td>
<td>Id of the CVD that is migrated.</td>
</tr>
<tr>
<td>IdentityInfo</td>
<td>MachineIdentityInfo</td>
<td>Information such as domain name, user name, password, and so on. See MachineIdentityInfo.</td>
</tr>
</tbody>
</table>

Sample code:

MigrationTarget migrationTarget = new MigrationTarget
{
    CvdId = cvdDetails.Id,
    IdentityInfo = new MachineIdentityInfo
    {
        DomainMember = true,
        DomainOrWorkgroupName = "domain.com",
        User = "bob",
        Password = "password"
    }
};

**MachineIdentityInfo**

MachineIdentityInfo represents information that is used after migration. It has the following properties:
LayerDetails

MigrationTarget represents information for a migration. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>string</td>
<td>Name of the layer.</td>
</tr>
<tr>
<td>ImageId</td>
<td>ImageId</td>
<td>Contains Id and Version of the layer. Id cannot represent a unique layer. Different layers can have the same Id. ImageId is often used when a method needs a layer.</td>
</tr>
</tbody>
</table>

Sample code:

```java
MigrationTarget[] migrationTargets = GetMigrationTargets();
LayerDetails baseLayer = GetBaseLayer();
LayerDetails[] appLayers = GetAppLayers();
BatchResult result = OsMigration(migrationTargets, baseLayer.ImageId, appLayers.Select(appLayer => appLayer.ImageId).ToArray(), false /* no warning */);
```

Imgeld

Imgeld represents the identity of an image, such as a layer or a policy. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Id</td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>ImageVersion</td>
<td></td>
</tr>
</tbody>
</table>

Note that Id cannot represent a unique Imgeld. Different Imgeld objects can have the same Id. Two Imgeld objects are the same if and only if their Id and Version are the same. Imgeld is used in the following types. It is Imgeld rather than Id that identifies the object.

- LayerDetails
- PolicyDetails

Sample code to query a base layer and get its Imgeld:

```java
QueryResult queryResult = Client.BaseLayer_Query(new QueryDefinition { Page = 1 });
if (queryResult.Elements.Length > 0)
{
    Imgeld imageId = ((LayerDetails)queryResult.Elements[0]).ImageId;
    // do your work with imageId
}
```
**ImageVersion**

ImageVersion represents the version of an image. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major</td>
<td>int</td>
<td>Major version.</td>
</tr>
<tr>
<td>Minor</td>
<td>int</td>
<td>Minor version.</td>
</tr>
</tbody>
</table>

ImageVersion is used in the following types:

- CvdDetails
- ImageId

Sample code to query a CVD and get its machine version:

```csharp
QueryResult queryResult = Client.Cvd_Query(new QueryDefinition { Page = 1 });
if (queryResult.Elements.Length > 0)
{
    ImageVersion machineVersion = ((CvdDetails)queryResult.Elements[0]).MachineVersion;
    // do your work with machineVersion
}
```

**ClientEvent**

ClientEvent represents client event details in CvdDetails. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventId</td>
<td>Id</td>
</tr>
<tr>
<td>CvdId</td>
<td>Id</td>
</tr>
<tr>
<td>DeviceId</td>
<td>Id</td>
</tr>
<tr>
<td>Source</td>
<td>string</td>
</tr>
<tr>
<td>Counter</td>
<td>int</td>
</tr>
<tr>
<td>StartTime</td>
<td>DateTime</td>
</tr>
<tr>
<td>LastTime</td>
<td>DateTime</td>
</tr>
<tr>
<td>Type (Deprecated)</td>
<td>ClientEventType</td>
</tr>
<tr>
<td>EventType</td>
<td>string</td>
</tr>
</tbody>
</table>

In CvdDetails, the ClientEvent array is sorted by LastTime in descending order. ClientEventType is of enum type. Sample code to query a CVD and get its ClientEvent:

```csharp
QueryResult queryResult = Client.Cvd_Query(new QueryDefinition { Filter = null, Page = 1 });
if (queryResult.Elements.Length != 0)
{
    ClientEvent[] clientEvents = ((CvdDetails)queryResult.Elements[0]).ClientEvents;
    // do your work with clientEvents
}
```

**ClientEventType (Deprecated)**

ClientEventType is of enum type. It has the following values:
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT_PROGRESS</td>
<td>Client progress client event.</td>
</tr>
<tr>
<td>JOIN_DOMAIN</td>
<td>Joining domain client event.</td>
</tr>
</tbody>
</table>

**ClientState (Deprecated)**

ClientState is an enum type. It has the following states:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Idle.</td>
</tr>
<tr>
<td>PendingReboot</td>
<td>Pending a reboot.</td>
</tr>
<tr>
<td>ForceReboot</td>
<td>Forcing a reboot.</td>
</tr>
<tr>
<td>UploadInitializing</td>
<td>Initializing an upload.</td>
</tr>
<tr>
<td>Uploading</td>
<td>Uploading.</td>
</tr>
<tr>
<td>Others</td>
<td>Other states.</td>
</tr>
<tr>
<td>RestorePrefetch</td>
<td>Restoring prefetching.</td>
</tr>
<tr>
<td>RestoreStreaming</td>
<td>Restoring streaming.</td>
</tr>
<tr>
<td>UpdateLayers</td>
<td>Downloading image.</td>
</tr>
<tr>
<td>RebasePrefetching</td>
<td>Prefetching for restore and base layer update.</td>
</tr>
<tr>
<td>Migration</td>
<td>Migration.</td>
</tr>
<tr>
<td>DriverLibraryUpdate</td>
<td>Driver library updating.</td>
</tr>
<tr>
<td>RestoreUserData</td>
<td>Restore and profile prefetching.</td>
</tr>
<tr>
<td>DeviceProvisioning</td>
<td>Device provisioning.</td>
</tr>
<tr>
<td>AppLayerRecording</td>
<td>App layer recording.</td>
</tr>
<tr>
<td>AppLayerCapture</td>
<td>App layer capture.</td>
</tr>
<tr>
<td>Synchronizing</td>
<td>Synchronizing.</td>
</tr>
<tr>
<td>PendingUpgrade</td>
<td>Pending an upgrade.</td>
</tr>
<tr>
<td>PendingRestore</td>
<td>Pending a restore.</td>
</tr>
<tr>
<td>PendingAssignment</td>
<td>Pending an assignment.</td>
</tr>
<tr>
<td>Suspended</td>
<td>Suspended.</td>
</tr>
<tr>
<td>WaitingForService</td>
<td>Waiting for service.</td>
</tr>
<tr>
<td>Throttled</td>
<td>Throttled.</td>
</tr>
<tr>
<td>PendingUserAuthentication</td>
<td>Pending user authentication.</td>
</tr>
<tr>
<td>Snoozed</td>
<td>Snoozed.</td>
</tr>
<tr>
<td>BranchReflectorIsBusy</td>
<td>Branch reflector is busy.</td>
</tr>
<tr>
<td>WaitingDeviceProvisioning</td>
<td>Waiting for device provisioning.</td>
</tr>
<tr>
<td>RecordingAppLayer</td>
<td>Recording app layer.</td>
</tr>
<tr>
<td>NoPotentialBranchReflectors</td>
<td>No potential branch reflectors.</td>
</tr>
<tr>
<td>BranchReflectorIsCaching</td>
<td>Branch reflector is caching.</td>
</tr>
<tr>
<td>PendingGatewayLogin</td>
<td>Pending authentication from gateway.</td>
</tr>
<tr>
<td>PendingRebootStreaming</td>
<td>Pending reboot for restore streaming.</td>
</tr>
<tr>
<td>RestoreInitializing</td>
<td>Restore initializing.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>UpdateLayersInitializing</td>
<td>Download image initializing.</td>
</tr>
<tr>
<td>RebasePrefetchInitializing</td>
<td>Restore and base layer update initializing.</td>
</tr>
<tr>
<td>MigrationInitializing</td>
<td>Migration initializing.</td>
</tr>
<tr>
<td>DriverLibraryUpdateInitializing</td>
<td>Driver library update initializing.</td>
</tr>
<tr>
<td>RestoreUserDataInitializing</td>
<td>Restore user data initializing.</td>
</tr>
<tr>
<td>DeviceProvisioningInitializing</td>
<td>Device provisioning initializing.</td>
</tr>
<tr>
<td>AppLayerRecordingInitializing</td>
<td>App layer recording initializing.</td>
</tr>
<tr>
<td>AppLayerCaptureInitializing</td>
<td>App layer capture initializing.</td>
</tr>
<tr>
<td>SynchronizingInitializing</td>
<td>Synchronize initializing.</td>
</tr>
<tr>
<td>RecordingAppLayerInitializing</td>
<td>Record app layer initializing.</td>
</tr>
<tr>
<td>RestoreStreamingInitializing</td>
<td>Restore streaming initializing.</td>
</tr>
<tr>
<td>UploadChangesFinalizing</td>
<td>Upload changes finalizing.</td>
</tr>
<tr>
<td>RestorePrefetchFinalizing</td>
<td>Restore prefetch finalizing.</td>
</tr>
<tr>
<td>UpdateLayersFinalizing</td>
<td>Download image finalizing.</td>
</tr>
<tr>
<td>RebasePrefetchFinalizing</td>
<td>Restore and base layer update prefetch finalizing.</td>
</tr>
<tr>
<td>MigrationFinalizing</td>
<td>Migration finalizing.</td>
</tr>
<tr>
<td>DriverLibraryUpdateFinalizing</td>
<td>Driver library update finalizing.</td>
</tr>
<tr>
<td>RestoreUserDataFinalizing</td>
<td>Restore user data finalizing.</td>
</tr>
<tr>
<td>DeviceProvisioningFinalizing</td>
<td>Device provisioning finalizing.</td>
</tr>
<tr>
<td>AppLayerRecordingFinalizing</td>
<td>App layer recording finalizing.</td>
</tr>
<tr>
<td>AppLayerCaptureFinalizing</td>
<td>App layer capture finalizing.</td>
</tr>
<tr>
<td>RecordingAppLayerFinalizing</td>
<td>Recording app layer finalizing.</td>
</tr>
<tr>
<td>RestoreStreamingFinalizing</td>
<td>Restore streaming finalizing.</td>
</tr>
<tr>
<td>SynchronizingFinalizing</td>
<td>Synchronizing finalizing.</td>
</tr>
</tbody>
</table>

**OsVersion (Deprecated)**

OsVersion is an enum type. It has the following OS versions:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIN7</td>
<td>Windows 7 x86</td>
</tr>
<tr>
<td>WIN7X64</td>
<td>Windows 7 x64</td>
</tr>
<tr>
<td>XP</td>
<td>Windows XP</td>
</tr>
<tr>
<td>VISTA</td>
<td>Windows Vista x86</td>
</tr>
<tr>
<td>VISTAX64</td>
<td>Windows Vista x64</td>
</tr>
<tr>
<td>WIN8</td>
<td>Windows 8 x86</td>
</tr>
<tr>
<td>WIN8X64</td>
<td>Windows 8 x64</td>
</tr>
<tr>
<td>XP_EMBEDDED</td>
<td>Windows XP Embedded</td>
</tr>
<tr>
<td>WIN8_1</td>
<td>Windows 8.1 x86</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>WIN8_1X64</td>
<td>Windows 8.1 x64</td>
</tr>
<tr>
<td>OTHER</td>
<td>Others</td>
</tr>
</tbody>
</table>

**CollectionDetails**

CollectionDetails contains the details of a CVD collection. CollectionDetails has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Id</td>
<td>Id of the collection.</td>
</tr>
<tr>
<td>Name</td>
<td>string</td>
<td>Name of the collection.</td>
</tr>
<tr>
<td>Description</td>
<td>string</td>
<td>Description of the collection.</td>
</tr>
</tbody>
</table>

**ServerInformation**

ServerInformation contains about a Mirage API server. It has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>ServerVersion</td>
<td>The version of a Mirage API server.</td>
</tr>
<tr>
<td>BatchOperationMaximum</td>
<td>int</td>
<td>The maximum number of batch operations. The default value is 500.</td>
</tr>
</tbody>
</table>

**ServerVersion**

ServerVersion contains the version information of a Mirage API server. It has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major</td>
<td>int</td>
<td>Major version.</td>
</tr>
<tr>
<td>Minor</td>
<td>int</td>
<td>Minor version.</td>
</tr>
<tr>
<td>Build</td>
<td>int</td>
<td>Build number.</td>
</tr>
<tr>
<td>Revision</td>
<td>int</td>
<td>Revision</td>
</tr>
</tbody>
</table>
Performance

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
Permissions, Configuration and Logging

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 55
- “Configuration,” on page 56
- “Logging,” on page 56

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 5-1. Roles Required by Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Roles Needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>None</td>
</tr>
<tr>
<td>Logout</td>
<td>None</td>
</tr>
<tr>
<td>AppLayer_Query</td>
<td>ListBaseImages</td>
</tr>
<tr>
<td>BaseLayer_Query</td>
<td>ListBaseImages</td>
</tr>
<tr>
<td>Cvd_Query</td>
<td>ListCvds, ListEvents</td>
</tr>
<tr>
<td>Cvd_Get ListCvds</td>
<td>ListCvds, ListEvents</td>
</tr>
<tr>
<td>Policy_Query</td>
<td>ListPolicies</td>
</tr>
<tr>
<td>Volume_Query</td>
<td>ListLicense</td>
</tr>
<tr>
<td>PendingDevice_Query</td>
<td>ListDevices</td>
</tr>
<tr>
<td>PendingDevice_CreateNewCvd</td>
<td>AdministerCvds, ListDevices</td>
</tr>
<tr>
<td>OsMigration_Begin</td>
<td>AdministerCvds, ListCvds</td>
</tr>
<tr>
<td>OsMigration_BeginDownloadOnly</td>
<td>AdministerCvds, ListCvds</td>
</tr>
<tr>
<td>OsMigration_ApplyDownloadOnlyMigration</td>
<td>AdministerCvds, ListCvds</td>
</tr>
<tr>
<td>OsMigration_QueryDownloadOnlyInProgress</td>
<td>ListCvds</td>
</tr>
<tr>
<td>OsMigration_QueryDownloadOnlyCompleted</td>
<td>ListCvds</td>
</tr>
<tr>
<td>Cvd_Archive</td>
<td>AdministerCvds, AdministerReferenceCvds, ListReferenceCvds, ListTasks</td>
</tr>
<tr>
<td>Cvd_Delete</td>
<td>AdministerCvds, ListCvds</td>
</tr>
</tbody>
</table>
Table 5-1. Roles Required by Methods (Continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Roles Needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cvd_Sync</td>
<td>ListDevices, SupportDevices</td>
</tr>
<tr>
<td>Cvd_ApplyPolicy</td>
<td>AdministerCvds, ListCvds, ListPolicies</td>
</tr>
<tr>
<td>PendingDevice_Provision</td>
<td>AdministerCvds, ListBaseImages, ListDevices, ListLicense, ListPolicies</td>
</tr>
<tr>
<td>Collection_Query</td>
<td>ListCvds</td>
</tr>
</tbody>
</table>

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 5-2. Configuration File Settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max receiving message size</td>
<td>31457280 (30MB)</td>
</tr>
<tr>
<td>Receive timeout</td>
<td>5 minutes</td>
</tr>
<tr>
<td>Send timeout</td>
<td>5 minutes</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log file name</td>
<td>mirageapi.log</td>
</tr>
<tr>
<td>Max log file size</td>
<td>10MB</td>
</tr>
<tr>
<td>Max log file backups</td>
<td>5 log files</td>
</tr>
</tbody>
</table>

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

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 57
- “Sample Java Application,” on page 68

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.

using System;
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;
            }
            return 0;
        }
    }
}
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("-OsMigrationDownloadOnlyInProgressQuery-");
OsMigrationDownloadOnlyInProgressQuery();

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

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

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

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)
    {
        // Handle exception
    }
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..." WARRANTIES PROTECTED BY COPYRIGHT. NO UNAUTHORIZED USE.

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];
VMware, Inc. 62
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;
}
// Check the provisioning 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 (IsImageIdEquals(cvdDetails.BaseImageId, baseLayer.ImageId))
    {
        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));
}
// 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;
}

private static bool IsImageIdEquals(ImageId imageIdA, ImageId imageIdB)
{
  if (imageIdA == null || imageIdB == null)
  {
    return false;
  }
  return imageIdA.Id.IdValue == imageIdB.Id.IdValue && imageIdA.Version.Major ==
  imageIdB.Version.Minor &&
  imageIdA.Version.Minor == imageIdB.Version.Minor;
}

// Query Cvd which completes downloading base layer and app layers.
private static void OsMigrationDownloadOnlyCompletedQuery()
{
  QueryDefinition query = new QueryDefinition
  {
    Filter = null,
    Page = 1
  };
  QueryResult result = Client.OsMigration_QueryDownloadOnlyCompleted(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);}
}

// Query Cvd which is downloading base layer or app layers.
private static void OsMigrationDownloadOnlyInProgressQuery()
QueryDefinition query = new QueryDefinition
{
    Filter = null,
    Page = 1
};
QueryResult result = Client.OsMigration_QueryDownloadOnlyInProgress(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},
    progress: {4}",
    cvdDetails.Id.IdValue, cvdDetails.DeviceId.IdValue,
    cvdDetails.PolicyId.IdValue,
    cvdDetails.MachineVersion.Major,
    cvdDetails.MachineVersion.Minor,
    cvdDetails.OperationProgress);
}

// OS migration flow.
private 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...");
WaitForOsMigrationApplyCompleted(baseLayer);

// Check the migration completes.
private static void WaitForOsMigrationApplyCompleted(LayerDetails baseLayer)
{
    QueryDefinition query = new QueryDefinition
    {
        Filter = null,
        Page = 1
    };
    while (true)
    {
        QueryResult result = Client.Cvd_Query(query);
        if (result.Elements.Length == 0)
        {
            return;
        }
        CvdDetails cvdNew = (CvdDetails) result.Elements[0];
        if (IsImageIdEquals(cvdNew.BaseImageId, baseLayer.ImageId))
        {
            break;
        }
        Thread.Sleep(TimeSpan.FromMinutes(2));
    }
    Console.WriteLine("Migration Apply success!");
}

// Check the Cvd completes to download base layer and app layers.
private static void WaitForOsMigrationDownloadCompleted()
while (true)
{
    QueryResult result =
        Client.OsMigration_QueryDownloadOnlyCompleted(new QueryDefinition {Filter = null, Page = 1});
    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.
private static void WaitForOsMigrationDownloadInProgress()
{
    while (true)
    {
        QueryResult result =
            Client.OsMigration_QueryDownloadOnlyInProgress(new QueryDefinition {Filter = null, Page = 1});
        if (result.Elements.Length != 0)
        {
            Console.WriteLine("Migration downloading...");
            break;
        }
        Thread.Sleep(TimeSpan.FromMinutes(2));
    }
}

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.
package com.vmware.mirage.mit.sample;

import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
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 javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
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();
        } 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 {
        // Code...
    }
}
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 (int i = 0; i < result.getElements().getAnyType().length; i++) {
            final PolicyDetails policyDetails = (PolicyDetails)
            result.getElements().getAnyType()[i];
            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 (int i = 0; i < result.getElements().getAnyType().length; i++) {
            final VolumeDetails volumeDetails = (VolumeDetails)
            result.getElements().getAnyType()[i];
            System.out.println(String.format("volumeDetails %s %s", volumeDetails.getName(),
            volumeDetails.getPath()));
        }
    }
}

// Query Cvs whose name starts with "VMware" and progress equals to 100, only the first
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 (int i = 0; i < result.getElements().getAnyType().length; i++) {
            final CvdDetails cvdDetails = (CvdDetails) result.getElements().getAnyType()[i];
            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 (int i = 0; i < result.getElements().getAnyType().length; i++) {
        final DeviceDetails deviceDetails = (DeviceDetails)
            result.getElements().getAnyType()[i];
        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 (int i = 0; i < result.getElements().getAnyType().length; i++) {
            final LayerDetails baseLayerDetails = (LayerDetails)
                result.getElements().getAnyType()[i];
            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 (int i = 0; i < result.getElements().getAnyType().length; i++) {
            final LayerDetails baseLayerDetails = (LayerDetails)
                result.getElements().getAnyType()[i];
            System.out.println(String.format("appLayer %s %s", baseLayerDetails.getName(),
                baseLayerDetails.getImageId()));
        }
    }
}

// Centralization endpoint flow.
public void ceFlow() 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];

/* 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 */
        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);
}

/* Get the Id of new Cvd from BatchResult */
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 ImageId baseImageId = cvdDetails.getBaseImageId();
        /* The complete condition of provision flow is that the base image Id has been
        updated */
        if (baseImageId.getId().getIdValue() ==
            baseLayerDetails.getImageId().getId().getIdValue()
            && baseImageId.getVersion().getMajor() ==
            baseLayerDetails.getImageId().getVersion().getMajor()
            && baseImageId.getVersion().getMinor() ==
            baseLayerDetails.getImageId().getVersion().getMinor()) {
            break;
        }
        TimeUnit.MINUTES.sleep(2);
    }
}
System.out.println("OK to provision pending device.");

// OS migration flow.
public void migrationFlow() throws Exception {
    QueryDefinition queryDefinition = new QueryDefinition();
    queryDefinition.setPage(1);
    Cvd_Query cvdQuery = new Cvd_Query();
cvdQuery.setQueryDefinition(queryDefinition);
    final Cvd_QueryResponse cvdResponse = client.cvd_Query(cvdQuery);
    QueryResult cvdResult = cvdResponse.getCvd_QueryResult();
    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()));
}

// check migration finish
final QueryFilterEquals equalFilter = new QueryFilterEquals();
equalFilter.setField(FilterField.CVD_ID);
equalFilter.setValue(cvd.getId());

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

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

while (true) {
    final Cvd_QueryResponse cvds = client.cvd_Query(cvdQuery);
    cvdResult = cvds.getCvd_QueryResult();
    cvdArr = cvdResult.getElements().getAnyType();
    if (cvdArr.length > 0) {
        final CvdDetails cvdDetails = (CvdDetails) cvdArr[0];
        if (cvdDetails.getBaseImageId() != null
            && cvdDetails.getBaseImageId().getId().getIdValue() ==
            baseLayer.getImageId().getId().getIdValue()) {
            break;
        }
    }
    TimeUnit.MINUTES.sleep(2);
    System.out.println("Waiting for migration... ");
}

System.out.println("Migration finished.");

// 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 53

C
configuration 55, 56

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

F
fault types 43

G
glossary 5

I
IMitService 43
intended audience 5

L
logging 55, 56

M
methods 13

P
performance 53
permissions 55

Q
query types 37

R
roles 55

S
sample applications
C# 57
Java 68
service type 43

T
types
fault 43
other 44
query 37
service 43

W
WCF HTTP activation
Windows Server 2008 R2 7
Windows Server 2012 8