This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see http://www.vmware.com/support/pubs.
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,” on page 8
- “Set Up a Java Development Environment,” on page 9

Enable the WCF HTTP Activation Feature in Windows Server 2008 R2

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

Prerequisites

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

Procedure

1. Log in as an administrator.
2. Click Start > Administrative Tools > Server Manager.
3. When the Server Manager window displays, click Features > Add Features.
4. Expand .NET Framework 3.5.1 Features.
5. Expand WCF Activation.
6. Select HTTP Activation and click Install.
7. Follow the prompts and finish the installation.
8. Open a Command Prompt window.
9. Run the %WINDIR%\Microsoft.NET\Framework\v4.0.30319\aspnet_regiis.exe -iru command.
Enable the WCF HTTP Activation Feature in Windows Server 2012

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

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

**Procedure**
1. Log in as an administrator.
2. Click **Start > Control Panel > Turn Windows features on or off**.
3. Click **Next** until the Select Features window appears.
4. Expand **.NET Framework 4.5 Features**.
5. Expand **WCF Services**.
6. Select **HTTP Activation** and click **Install**.
7. Follow the prompts and finish the installation.

Set Up a Windows C# Development Environment

If you plan to use the Microsoft C# programming language to develop your applications, 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.wsdl` 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 Java Development Environment**

If you plan to use the Java programming language to develop your applications, 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:

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

//Perform tasks
```

```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
```
Mirage API has 15 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</td>
<td>Queries the CVDs which are downloading the base layer or app layers.</td>
</tr>
<tr>
<td>OsMigration_QueryDownloadOnlyCompleted</td>
<td>Queries the CVDs which are finished downloading the base layer or app layers.</td>
</tr>
</tbody>
</table>

For more information about each method, including input parameters, return value, and faults, see the Mirage API Reference in [http://pubs.vmware.com/mirage-51/index.jsp](http://pubs.vmware.com/mirage-51/index.jsp).

**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 role Administrator, Web Help Desk, or Web Protection Manager 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:
None

Logout
Logout logs off the current user and invalidates the session.
Input:
None
Return:
None

Policy_Query
Policy_Query queries policies.
Input:
- queryDefinition
- Filter

<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>
</tbody>
</table>

- Page
  Starts at 1.

Return:
- QueryResult
  The type of element is PolicyDetails.

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

Volume_Query
Volume_Query queries a volume.
PendingDevice_Query

PendingDevice_Query queries pending devices.

Input:
- queryDefinition

Filter

<table>
<thead>
<tr>
<th>Property</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_VERSION</td>
<td>OsVersion</td>
<td>The OS version of the device.</td>
</tr>
<tr>
<td>DEVICE_USER_NAME</td>
<td>string</td>
<td>The user name 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>

Page
Starts at 1.
Return:

- QueryResult
  The type of element is DeviceDetails.

Example:

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

**Cvd_Query**

Cvd_Query queries CVDs.

Input:

- queryDefinition
  - Filter

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVDDEVICE_CLIENT_STATUS</td>
<td>ClientState</td>
<td>The client status of the CVD.</td>
</tr>
<tr>
<td>CVDDEVICE_CONNECTION_STATE</td>
<td>bool</td>
<td>The connection state of the CVD.</td>
</tr>
<tr>
<td>CVDDEVICE_ID</td>
<td>Id</td>
<td>The device Id 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_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>

- 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>
</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.
    - Page
      - Starts at 1.

**Return:**
- QueryResult
  - The type of element is LayerDetails.

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

**PendingDevice_CreateNewCvd**

PendingDevice_CreateNewCvd creates new CVDs from existing pending devices.

**Input:**
- deviceIds
  - A list of pending device Ids. These Ids can be queried from PendingDevice_Query. If there are invalid Ids in the list, they will be skipped and faults will be set in BatchResult. New CVDs will be created for devices whose Id is valid.
- policyId
  - The imageId of a policy.
- **volumeId**
  
  The Id of the volume that is used to store the CVDs.

Return:

- **BatchResult**

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

Example:

```csharp
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:

```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_Begin(new MigrationTarget[] { migrationTarget }, baseLayer.ImageId,  
appLayers.Select(appLayer => appLayer.ImageId).ToArray(), true /* ignore warnings */);

OsMigration_BeginDownloadOnly

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

Input:

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

- **baseLayerId**
  
  The ImageId of the base layer.

- **appLayerIds**
  
  The ImageIds of app layers.

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

Return:

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

Example:

LayerDetails baseLayer = GetBaseLayer();
LayerDetails[] appLayers = GetAppLayers();
MigrationTarget migrationTarget = new MigrationTarget  
{  
  CvdId = cvd.Id,  
  IdentityInfo = new MachineIdentityInfo  
  {  
    DomainMember = true,  
    DomainOrWorkgroupName = "mydomain.com",  
    User = "bob",  
    Password = "password"  
  }  
}
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

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**

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:

```java
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.
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* in http://pubs.vmware.com/mirage-51/index.jsp

- Query
- Fault
- Service
- Other

This chapter includes the following topics:

- “Query Types,” on page 21
- “Fault Types,” on page 26
- “Service Types,” on page 27
- “Other Types,” on page 27

### 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 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_VERSION,
        Value = OsVersion.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_VERSION,
        Value = OsVersion.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.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_VERSION,
    Value = OsVersion.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:

QueryFilter win7Filter = new QueryFilterEquals
{
    Field = FilterField.DEVICE_OS_VERSION,
    Value = OsVersion.WIN7
};
QueryFilter xpFilter = new QueryFilterEquals
{
    Field = FilterField.DEVICE_OS_VERSION,
    Value = OsVersion.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</td>
<td>string</td>
</tr>
<tr>
<td>VOLUME_PATH</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith</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>DEVICE_ID</td>
<td>QueryFilterEquals,</td>
<td>Id</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals</td>
<td></td>
</tr>
<tr>
<td>DEVICE_NAME</td>
<td>QueryFilterEquals,</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterContains,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterBeginsWith</td>
<td></td>
</tr>
<tr>
<td>DEVICE_USER_NAME</td>
<td>QueryFilterEquals,</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterContains,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterBeginsWith</td>
<td></td>
</tr>
<tr>
<td>DEVICE_MODEL_NAME</td>
<td>QueryFilterEquals,</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterContains,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterBeginsWith</td>
<td></td>
</tr>
<tr>
<td>DEVICE_VENDOR_NAME</td>
<td>QueryFilterEquals,</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterContains,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterBeginsWith</td>
<td></td>
</tr>
<tr>
<td>DEVICE_OS_VERSION</td>
<td>QueryFilterEquals,</td>
<td>OsVersion</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals</td>
<td></td>
</tr>
<tr>
<td>DEVICE_CONNECTION_STATE</td>
<td>QueryFilterEquals,</td>
<td>bool (connected -- true)</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals</td>
<td></td>
</tr>
<tr>
<td>CVD_ID</td>
<td>QueryFilterEquals,</td>
<td>Id</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals</td>
<td></td>
</tr>
<tr>
<td>CVD_DEVICE_ID</td>
<td>QueryFilterEquals,</td>
<td>Id</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals</td>
<td></td>
</tr>
<tr>
<td>CVD_POLICY_ID</td>
<td>QueryFilterEquals,</td>
<td>Id</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals</td>
<td></td>
</tr>
<tr>
<td>CVD_NAME</td>
<td>QueryFilterEquals,</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterContains,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterBeginsWith</td>
<td></td>
</tr>
<tr>
<td>CVD_USER_NAME</td>
<td>QueryFilterEquals,</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterContains,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterBeginsWith</td>
<td></td>
</tr>
<tr>
<td>CVD_POLICY_NAME</td>
<td>QueryFilterEquals,</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterContains,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QueryFilterBeginsWith</td>
<td></td>
</tr>
<tr>
<td>CVD_DEVICE_CONNECTION_STATE</td>
<td>QueryFilterEquals,</td>
<td>bool (connected -- true)</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals</td>
<td></td>
</tr>
<tr>
<td>CVD_DEVICE_CLIENT_STATUS</td>
<td>QueryFilterEquals,</td>
<td>ClientState</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals</td>
<td></td>
</tr>
<tr>
<td>CVD_PROGRESS</td>
<td>QueryFilterEquals,</td>
<td>Long</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals</td>
<td></td>
</tr>
<tr>
<td>CVD_MACHINE_VERSION</td>
<td>QueryFilterEquals,</td>
<td>ImageVersion</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals</td>
<td></td>
</tr>
<tr>
<td>POLICY_ID</td>
<td>QueryFilterEquals,</td>
<td>Id</td>
</tr>
<tr>
<td></td>
<td>QueryFilterNotEquals</td>
<td></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>POLICY_NAME</td>
<td>QueryFilterEquals, QueryFilterNotEquals, QueryFilterContains, QueryFilterBeginsWith</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</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.
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:

```java
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

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:

```java
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</td>
<td>OsVersion</td>
<td>See the OsVersion type.</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</td>
<td>Id</td>
<td>Id of the policy when creating the CVD.</td>
</tr>
<tr>
<td>Connected</td>
<td>bool</td>
<td>Whether the device is connected to the server.</td>
</tr>
<tr>
<td>State</td>
<td>ClientState</td>
<td>The current state of the device.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</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>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>
</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**

**Type of QueryResult Elements**

- **AppLayer_Query**  LayerDetails
- **BaseLayer_Query**  LayerDetails
- **Cvd_Query**  CvdDetails
- **OsMigration_QueryDownloadOnlyCompleted**  CvdDetails
- **OsMigration_QueryDownloadOnlyInProgress**  CvdDetails
- **PendingDevice_Query**  DeviceDetails
- **Policy_Query**  PolicyDetails
- **Volume_Query**  VolumeDetails

Sample code to query CVDs, get QueryResult and convert the element to CvdDetails.

```csharp
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:

```csharp
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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MachineName</td>
<td>string</td>
<td>Yes</td>
<td>Machine name after migration.</td>
</tr>
<tr>
<td>DomainMember</td>
<td>bool</td>
<td>No</td>
<td>Whether the machine will join a domain after migration. If false, the machine will be in a workgroup.</td>
</tr>
<tr>
<td>DomainOrWorkgroupName</td>
<td>string</td>
<td>No</td>
<td>Domain or workgroup name. Cannot be null or an empty string.</td>
</tr>
</tbody>
</table>
### User
- **Type**: string
- **Optional**: No
- **Description**: Login user name after migration.

### Password
- **Type**: string
- **Optional**: No
- **Description**: Login password after migration.

### OU
- **Type**: string
- **Optional**: Yes
- **Description**: Organizational unit information of the domain server. Can be null.

---

**LayerDetails**

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>string</td>
<td></td>
<td>Name of the layer.</td>
</tr>
<tr>
<td>ImageId</td>
<td>ImageId</td>
<td></td>
<td>Contains Id and Version of the 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 */);
```

---

**ImageId**

ImageId 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>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Id</td>
</tr>
<tr>
<td>Version</td>
<td>ImageVersion</td>
</tr>
</tbody>
</table>

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

- LayerDetails
- PolicyDetails

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

```java
QueryResult queryResult = Client.BaseLayer_Query(new QueryDefinition { Page = 1 });
if (queryResult.Elements.Length > 0)
{
  ImageId 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</td>
<td>ClientEventType</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 ClientState:
```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**

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>
Sample code to verify that a CVD client event type is JOIN_DOMAIN:

```csharp
CvdDetails cvdMigrated = CvdQuery(); // Get cvd with Cvd_Query
if (cvdMigrated == null)
{
    return false;
}
IEnumerable<ClientEvent> clientEvents = cvdMigrated.ClientEvents.Where(ce => (ce.Type == ClientEventType.JOIN_DOMAIN));
return clientEvents.Count() != 0;
```

### ClientState

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>Client is idle.</td>
</tr>
<tr>
<td>PendingReboot</td>
<td>Client is pending a reboot.</td>
</tr>
<tr>
<td>ForceReboot</td>
<td>Client is forcing a reboot.</td>
</tr>
<tr>
<td>UploadInitializing</td>
<td>Client is initializing an upload.</td>
</tr>
<tr>
<td>Uploading</td>
<td>Client is uploading.</td>
</tr>
<tr>
<td>Others</td>
<td>Other states.</td>
</tr>
</tbody>
</table>

### OsVersion

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>WIN8_1X64</td>
<td>Windows 8.1 x64</td>
</tr>
<tr>
<td>OTHER</td>
<td>Others</td>
</tr>
</tbody>
</table>
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 35
- “Configuration,” on page 36
- “Logging,” on page 36

Permissions

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

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

**Table 4-1. Roles Required by Methods**

<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>AdministratorCvds, ListDevices</td>
</tr>
<tr>
<td>OsMigration_Begin</td>
<td>AdministratorCvds, ListCvds</td>
</tr>
<tr>
<td>OsMigration_BeginDownloadOnly</td>
<td>AdministratorCvds, ListCvds</td>
</tr>
<tr>
<td>OsMigration_ApplyDownloadOnlyMigration</td>
<td>AdministratorCvds, ListCvds</td>
</tr>
<tr>
<td>OsMigration_QueryDownloadOnlyInProgress</td>
<td>ListCvds</td>
</tr>
<tr>
<td>OsMigration_QueryDownloadOnlyCompleted</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>
<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.
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 37
- “Sample Java Application,” on page 46

**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.

```csharp
// 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.query;
using vmware.mirage.mit.types;
using vmware.mirage.mit.faults;

namespace SampleCode
{
    class MirageApi Sample
    {
        static private MitServiceClient Client;

        static int Main(string[] args)
        {
            if (args.Length < 4)
            {
                Console.WriteLine(
                    "Usage: SampleCode <Web_Management_Server_IP> <Username> <Password> <migration_domain>\n";
                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]);
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.OsVersion, deviceDetails.VendorName, deviceDetails.ModelName, deviceDetails.Connected, deviceDetails.LastConnectedTime, deviceDetails.OperationTime);
    }
}

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

// Query Cvd with cvdId. It will throw FaultException<InvalidArgument> when the cvdId
// is not the Id type.
public static void CvdQuery(long cvdId)
{
    QueryFilterEquals filter = new QueryFilterEquals
    {
        Field = FilterField.CVD_ID,
        Value = cvdId //Invalid value type for CVD_ID, expected type: Id
    };
    QueryDefinition query = new QueryDefinition
    {
        Filter = filter,
        Page = 1
    };
    try
    {
        Client.Cvd_Query(query);
    }
    catch (FaultException<InvalidArgument> e)
    {
        InvalidArgument detail = e.Detail;
        if (detail.GetType() == typeof(InvalidArgument))
        {
            Console.WriteLine("InvalidArgument fault, detail message: " +
                detail.Message);
            //Expected output:
            //InvalidArgument fault, detail message: Bad value for queryDefinition:
            "Filter value for CVD_ID must be of type Id"
            //Parameter name: queryDefinition
        }
// Query Cvd, whose Id isn't cvdId, and cvd name begins with cvdUsername.
public static void CvdQuery(long cvdId, string cvdUsername) {
    QueryFilterNotEquals filterNotEquals = new QueryFilterNotEquals
    {
        Field = FilterField.CVD_ID,
        Value = new Id
        {
            IdValue = cvdId
        },
    };

    QueryFilterBeginsWith filterBeginsWith = new QueryFilterBeginsWith
    {
        Field = FilterField.CVD_USER_NAME,
        Value = cvdUsername
    };

    QueryDefinition query = new QueryDefinition
    {
        Filter = new QueryFilterAnd
        {
            Filters = new QueryFilter[] { filterNotEquals, filterBeginsWith }
        },
        Page = 1
    };

    QueryResult result = Client.Cvd_Query(query);
    if (result.Elements.Length == 0)
    {
        return;
    }

    foreach (CvdDetails cvdDetails in result.Elements)
    {
        Console.WriteLine("ID: {0}, device ID: {1}, policy ID: {2}, machine version: {3}.
{4} ",
            cvdDetails.Id.IdValue, cvdDetails.DeviceId.IdValue,
    }
}

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

// Check the centralization endpoint flow complete.
while (true)
{
    result = Client.Cvd_Query(new QueryDefinition
    {
        Filter = new QueryFilterEquals
        {
            Field = FilterField.CVD_ID,
            Value = new Id
            {
                IdValue = (long)opResult.Result
            }
        },
        Page = 1
    });
    if (result.Elements.Length == 0)
    {
        return;
    }
    CvdDetails cvdDetails = (CvdDetails)result.Elements[0];
    if (cvdDetails.OperationProgress == 100 && cvdDetails.State == ClientState.Idle)
    {
        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));
}

// 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}. {4}, progress: {5}",
            cvdDetails.Id.IdValue, cvdDetails.DeviceId.IdValue,
            cvdDetails.OperationProgress);
    }
}

// Query Cvd which is complete 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,
    }
}

// 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)
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);

// 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);
        return appLayerImageIds;
    }
    return new ImageId[0];
}
appLayerImageIds = appLayers.Select(a => a.ImageId).ToArray();

else
{
    appLayerImageIds = null;
}
return appLayerImageIds;

// 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));
    }
}

// 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 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 (cvdNew.BaseImageId != null && cvdNew.BaseImageId.Id.IdValue ==
baseLayer.ImageId.Id.IdValue)
{
    break;
}
Thread.Sleep(TimeSpan.FromMinutes(2));
}
Console.WriteLine("Migration Apply success!");
}
}

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 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/miteservice.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();
    } finally {
        try {
            sample.logout();
        } catch (final Exception e) {
            logger.error("Failed to logout.", e);
        }
    }
}

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

// Login Mirage API server with username and password
public void login() throws Exception {
    final Login login = new Login();
    login.setUsername(USERNAME);
    login.setPassword(PASSWORD);
    client.login(login);
}

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

// Query policy without filter, only the first page returns.
public void policyQuery() throws Exception {
    final QueryDefinition queryDefinition = new QueryDefinition();
    queryDefinition.setPage(1);
    final Policy_Query policy_Query = new Policy_Query();
    policy_Query.setQueryDefinition(queryDefinition);
    final Policy_QueryResponse policies = client.policy_Query(policy_Query);
    final QueryResult result = policies.getPolicy_QueryResult();
}
if (result.getElements().getAnyType() != null) {
    for (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 Cvds whose name starts with "DCNBEIJL" and progress equals to 100, only the first
// page returns.
public void cvdQuery() throws Exception {

    /* name starts with "DCNBEIJL" */
    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_PROGESS);
    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++) {
            // Add code for processing base layer elements
        }
    }
}
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();
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];
        if (cvdDetails.getOperationProgress() >= 100) {
            break;
        }
    }
    TimeUnit.MINUTES.sleep(2);
}
System.out.println("OK to create new cvd.");
}

// 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.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().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

**C**
configuration 35, 36

**D**
development environment
C# 8
Java 9
setting up 7

**F**
fault types 26

**G**
glossary 5

**I**
IMitService 27
intended audience 5

**L**
logging 35, 36

**M**
methods 11

**P**
permissions 35

**Q**
query types 21

**R**
roles 35

**S**
sample applications
C# 37
Java 46
service type 27

**T**
types
fault 26
other 27
query 21
service 27

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