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
# Contents

**About This Book** 5

1 vCenter Chargeback Manager APIs 7  
   - What Is vCenter Chargeback Manager? 7  
   - REST Architecture 7  
   - Requests 8  
   - Responses 8  
   - Common Elements in the Request and Response XMLs 9  
   - vCenter Chargeback Manager API Syntax 9  
   - API Versioning 10

2 Understanding the Workflow 13  
   - Requirements for Code Examples 13  
     - JAR Files 13  
       - CommonUtil Class 14  
       - FakeSSLCertificateSocketFactory Class 15  
   - Log In to vCenter Chargeback Manager 16  
   - Add vCenter Server Information 17  
   - Add a Custom Chargeback Hierarchy 18  
   - Add a vCenter Server Entity to the Chargeback Hierarchy 19  
   - Add a Fixed Cost 21  
   - Modify a Fixed Cost Value 22  
   - Generate a Report 23

3 Using vCenter Chargeback Manager with a Billing System 25  
   - Requirements for Code Examples 25  
     - JAR Files 25  
       - CommonUtil Class 26  
       - FakeSSLCertificateSocketFactory Class 27  
   - Add Pricing Models 28  
   - Add a Billing Policy 29  
   - Retrieve List of Hierarchies 30  
   - Get Details of a Hierarchy 31  
   - Add Report Schedule for Hierarchy 32  
   - Get Report Schedule by Hierarchy Name 33  
   - Reschedule a Report 35  
   - Delete Report Schedule 36  
   - Get List of Archived Reports for a Hierarchy 37  
   - Get a Report as XML 38

Index 39
About This Book

The *API Programming Guide* provides information on how to use vCenter Chargeback Manager APIs.

**Intended Audience**

This book is intended for anyone who develop applications to work with vCenter Chargeback Manager.

**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](http://www.vmware.com/support/pubs).

**Document Feedback**

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

**vCenter Chargeback Manager Documentation**

The vCenter Chargeback Manager documentation includes the following guides:

- Installation and Upgrade Guide
- User’s Guide
- API Programming Guide
- API Reference
- SDK for Java Developer’s Guide

To access the complete vCenter Chargeback Manager documentation, go to [http://www.vmware.com/support/pubs/vcbm_pubs.html](http://www.vmware.com/support/pubs/vcbm_pubs.html).

**Technical Support and Education Resources**

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

**Online and Telephone Support**

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

Customers with appropriate support contracts should use telephone support for the fastest response on priority 1 issues. Go to [http://www.vmware.com/support/phone_support](http://www.vmware.com/support/phone_support).
Support Offerings

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

VMware Professional Services

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

- “What Is vCenter Chargeback Manager?” on page 7
- “REST Architecture” on page 7
- “vCenter Chargeback Manager API Syntax” on page 9
- “API Versioning” on page 10

What Is vCenter Chargeback Manager?

vCenter Chargeback Manager is a cost reporting solution for environments virtualized using vSphere. This Web-based application interacts with the vCenter Server, vCloud Director, and vShield Manager to retrieve resource usage information and other configuration details, calculates the cost by using the chargeback formulas defined in the application, and generates reports.

vCenter Chargeback Manager runs on an Apache Tomcat server instance. The user interacts with the vCenter Chargeback Manager through a load balancer (Apache httpd Server). vCenter Chargeback Manager connects to a database that stores application-related information, such as the defined hierarchies, pricing model, users, roles, and so on. The application also interacts with the vCenter Server and vCenter Server Database through a data collector. The data collector uses VIM APIs to communicate with the vCenter Server and JDBC to communicate with the vCenter Server Database.

vCenter Chargeback Manager can be integrated with a vCloud Director setup using the vCloud Director data collector. This data collector interacts with the vCloud Director Database and fetches the Org data required for importing the hierarchies and the relevant data into vCenter Chargeback Manager. You can also install the vShield Manager data collector which helps you fetch the external network related information for each Org in the vCloud Director setup and replicate the same in the vCenter Chargeback Manager database.

vCenter Chargeback Manager APIs provide an interface for application developers to programmatically use the functionality of vCenter Chargeback Manager.

For more information about the vCenter Chargeback Manager and its capabilities, see the vCenter Chargeback Manager User’s Guide.

REST Architecture

vCenter Chargeback Manager APIs implement the Representational State Transfer (REST) architecture. REST-based APIs help you send HTTP requests for resources over the network and receive responses.
Requests

An HTTP request sent by a vCenter Chargeback Manager API can be of the following type: PUT, POST, GET, or DELETE. Table 1-1 shows how each of these request types maps to a standard CRUD operation.

Table 1-1. Request Type Mapping

<table>
<thead>
<tr>
<th>Request Type</th>
<th>CRUD Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>CREATE</td>
</tr>
<tr>
<td>GET</td>
<td>READ</td>
</tr>
<tr>
<td>PUT</td>
<td>UPDATE/CREATE</td>
</tr>
<tr>
<td>DELETE</td>
<td>DELETE</td>
</tr>
</tbody>
</table>

Along with the HTTP requests, you can pass request parameters by using XMLs. An example request XML for the Login API is provided here.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Request xmlns="http://www.vmware.com/vcenter/chargeback/2.0">
  <User>
    <Name>admin</Name>
    <Password>P@ssw0rd</Password>
    <Type>local</Type>
  </User>
</Request>
```

Responses

When an API task is successful, the value of the status parameter in the response XML is set to success as shown in the following example.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Response status="success" isValidLicense="true" xmlns="http://www.vmware.com/vcenter/chargeback/2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Hierarchies>
    <Hierarchy id="1">
      ...
    </Hierarchy>
  </Hierarchies>
</Response>
```
If an API task is unsuccessful, the status parameter is set to failure and the Error element captures all the details.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Response status="failure" isValidLicense="true"
 xmlns="http://www.vmware.com/vcenter/chargeback/2.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <Error majorErrorCode="500" minorErrorCode="2014"
 message="Hierarchy with id '1' does not exist." />
</Response>
```

Table 1-2 explains the components of the Error element.

### Table 1-2. Components of Error Element

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>majorErrorCode</td>
<td>The class of the error. It represents the HTTP Status codes.</td>
</tr>
<tr>
<td>minorErrorCode</td>
<td>The API error code. For example, it can indicate that hierarchy creation failed.</td>
</tr>
<tr>
<td>vendorSpecificErrorCode</td>
<td>A vendor or implementation-specific error code that points to specific modules or parts of the code, and can make problem diagnostics easier. For example, it can indicate if a code snippet is a vCenter Chargeback Manager error code or a database error code.</td>
</tr>
<tr>
<td>message</td>
<td>A one-line, human-readable message that describes the error that occurred.</td>
</tr>
<tr>
<td>ErrorStackTrace</td>
<td>This element is present when the log level in vCenter Chargeback Manager is set to debug.</td>
</tr>
</tbody>
</table>

### Common Elements in the Request and Response XMLs

Table 1-3 presents the common XML elements used by the request and response XMLs.

### Table 1-3. Common Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>An API request starts with this element.</td>
</tr>
<tr>
<td>Response</td>
<td>An API response starts with this element.</td>
</tr>
<tr>
<td>status</td>
<td>Denotes whether API is successfully executed.</td>
</tr>
<tr>
<td>isValidLicense</td>
<td>Indicates the status of the license. Value can be true or false.</td>
</tr>
</tbody>
</table>

For an example request XML, see “Requests.” For an example response XML, see “Responses.”

### vCenter Chargeback Manager API Syntax

Each vCenter Chargeback Manager API has the following syntax:

```xml
<HTTP_request_method> <Base_Url>/<API_signature>
```

For example, GET https://123.123.123/vCenter-CB/api/hierarchies

Table 1-4 describes the components of vCenter Chargeback Manager API syntax.

### Table 1-4. API Syntax Components

<table>
<thead>
<tr>
<th>Syntax Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_request_method</td>
<td>PUT, POST, GET, or DELETE</td>
</tr>
<tr>
<td>Base_Url</td>
<td>The URL of the vCenter Chargeback Manager. The base URL for vCenter Chargeback Manager APIs is: https://&lt;vCenter Chargeback Manager IP&gt;/vCenter-CB/api</td>
</tr>
<tr>
<td></td>
<td>For example, <a href="https://123.123.123/vCenter-CB/api">https://123.123.123/vCenter-CB/api</a></td>
</tr>
</tbody>
</table>
If you want to use an earlier version of the vCenter Chargeback Manager API, you must perform the following tasks:

- In the API URL, specify version=<version_number>.
  
  For example, `<HTTP_request_method> <Base_Url>/<API_signature>?version=1.5.0`

- In the request XML, update the request element with the following tag:
  
  `<Request xmlns="http://www.vmware.com/vcenter/chargeback/1.5.0">

NOTE Ensure that you specify the version in both the API URL and the request XML.

### API Versioning

Every vCenter Chargeback Manager API request and response includes target namespace to denote the API version. For example, the following request XML sends the version with which it is working.

```xml
<?xml version="1.0" encoding="UTF-8"?><Request xmlns="http://www.vmware.com/vcenter/chargeback/2.0">
  <Hierarchies>
    <Hierarchy>
      <Name>Test_Hierarchy</Name>
      <Description>Test Hierarchy</Description>
    </Hierarchy>
  </Hierarchies>
</Request>
```

The request states that vCenter Chargeback Manager API version 2.0 is being called. If the request is for vCenter Chargeback Manager version 1.5, the call fails. If the request is for vCenter Chargeback Manager 2.0, the call succeeds and gets a response similar to the following:

```xml
<?xml version="1.0" encoding="UTF-8"?><Response xmlns="http://www.vmware.com/vcenter/chargeback/2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" status="success" isValidLicense="true">
  <Hierarchies>
    <Hierarchy id="1174">
      <Name>Test_Hierarchy</Name>
      <Description>Test Hierarchy</Description>
      <CreatedOn>1275033342320</CreatedOn>
      <InSync>true</InSync>
      <LastUpdatedTime>1275033342350</LastUpdatedTime>
      <LastUpdatedUser>local\superuser</LastUpdatedUser>
      <Entities>
        <Entity id="1175">
          <Name>Test_Hierarchy</Name>
          <Description>Root</Description>
          <Type>100</Type>
          <Shares>
            <Share>
              <Percentage>100</Percentage>
              <Parent id="-1" />
              <StartTime>0</StartTime>
            </Share>
          </Shares>
        </Entity>
      </Entities>
    </Hierarchy>
  </Hierarchies>
</Response>
```
If you specify the API version, be sure to specify it with both the request XML (wherever applicable) and the URL. If it is specified only in the request XML or the URL and not both, then the API results in an error. If the API version is absent in both the URL and the request XML (wherever applicable), then the API returns the output corresponding to the latest version.
This chapter explains how to perform some of the basic vCenter Chargeback Manager tasks using the APIs. You can create a hierarchy and add entities, add fixed cost and generate a report on resource utilization.

The chapter includes the following topics:

- “Requirements for Code Examples” on page 13
- “Log In to vCenter Chargeback Manager” on page 16
- “Add vCenter Server Information” on page 17
- “Add a Custom Chargeback Hierarchy” on page 18
- “Add a vCenter Server Entity to the Chargeback Hierarchy” on page 19
- “Add a Fixed Cost” on page 21
- “Modify a Fixed Cost Value” on page 22
- “Generate a Report” on page 23

Requirements for Code Examples

This chapter provides code examples to explain how you can call the APIs. To run these code examples, you require the following JAR and helper class files.

JAR Files

The code examples need the following JAR files.

- commons-httpclient-3.1.jar
- commons-logging-1.1.1.jar
- jdom.jar

You must place these JAR files in the classpath.
CommonUtil Class

The code examples use the following CommonUtil class.

```java
public final class CommonUtil {

/**
 * This method writes out a formatted XML document to the Writer out.
 * @param doc
 * @param out
 * @throws IOException
 */
public static void printXML(Document doc, Writer out) throws IOException {
    XMLOutputter o = new XMLOutputter();
    Format newFormat = o.getFormat();
    newFormat.setIndent(" ");
    o.setFormat(newFormat);
    o.output(doc, out);
}

/**
 * This method returns the XML document as a string object
 * @param doc JDOM document representation of the XML
 * @return String representation of the XML document
 * @throws IOException
 */
public static String xmlAsString(Document doc) throws IOException {
    Writer w = new StringWriter();
    printXML(doc, w);
    return w.toString();
}

/**
 * This method reads the XML from a file and returns its representation as a
 * JDOM document
 * @param filePath path to the XML file
 * @return JDOM document representation of the XML
 * @throws IOException
 * @throws JDOMException
 */
public static Document getXMLDocument(String filePath) throws IOException, JDOMException {
    FileInputStream fis = null;
    try {
        fis = new FileInputStream(filePath);
        return getXMLDocument(fis);
    } finally {
        if (fis != null) {
            fis.close();
        }
    }
}

/**
 * This method reads an XML document from the input Stream and returns its
 * JDOM document representation
 * @param is input stream that has the XML content
 * @return jDOM document representation for the XML
 * @throws IOException
 * @throws JDOMException
 */
private static Document getXMLDocument(InputStream is) throws IOException, JDOMException {
    Document xml = null;
    SAXBuilder builder = new SAXBuilder();
    xml = builder.build(is);
    return xml;
}
}
FakeSSLCertificateSocketFactory Class

The code examples use the following class to access resources over the HTTPS protocol.

```java
/**
 * Helper class to accept self-signed certificate.
 */
public class FakeSSLCertificateSocketFactory implements SecureProtocolSocketFactory {
    private SSLContext sslContext;
    public FakeSSLCertificateSocketFactory() throws NoSuchAlgorithmException,
            KeyManagementException {
        sslContext = SSLContext.getInstance("SSL");
        sslContext.init(null, new TrustManager[]{new X509TrustManager() {
            @Override
            public void checkClientTrusted(X509Certificate[] ax509certificate, String s) throws
                    CertificateException {
                // Allow.
            }
            @Override
            public void checkServerTrusted(X509Certificate[] ax509certificate, String s) throws
                    CertificateException {
                // Allow.
            }
            @Override
            public X509Certificate[] getAcceptedIssuers() {
                return new X509Certificate[0];
            }
        }}, null);
    }
    @Override
    public Socket createSocket(String s, int i) throws IOException, UnknownHostException {
        return sslContext.getSocketFactory().createSocket(s, i);
    }
    @Override
    public Socket createSocket(String s, int i, InetAddress inetaddress, int j) throws
            IOException, UnknownHostException {
        return sslContext.getSocketFactory().createSocket(s, i, inetaddress, j);
    }
    @Override
    public Socket createSocket(String s, int i, InetAddress inetaddress, int j,
            HttpConnectionParams httpconnectionparams) throws IOException,
            UnknownHostException, ConnectTimeoutException {
        return sslContext.getSocketFactory().createSocket(s, i, inetaddress, j);
    }
    @Override
    public Socket createSocket(Socket socket, String s, int i, boolean flag) throws IOException,
            UnknownHostException {
        return sslContext.getSocketFactory().createSocket(socket, s, i, flag);
    }
}
```
Log In to vCenter Chargeback Manager

To start using vCenter Chargeback Manager APIs, you must log in to vCenter Chargeback Manager.

To log in to vCenter Chargeback Manager

1. Call the Login API by using the following syntax.
   ```
   <HTTP_request_method> <Base_URL>/login
   
   For example, you can define a call as follows:
   ```
   ```
   ```
2. (Optional) Use the URL parameter version to specify the API version that you want to call.
3. In the request XML for this API, you can specify the following login details.
   - User type
   - User name
   - Password
   - LDAP server ID or LDAP server name

   The following is an example request XML file.
   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <Request xmlns="http://www.vmware.com/vcenter/chargeback/2.0">
     <Users>
       <User>
         <Type>userType</Type>
         <Name>administrator</Name>
         <Password>vmware</Password>
         <!-- For Ldap User you can either provide LdapServer id or ldap server name -->
         <LdapServer id="1"/>
       </User>
     </Users>
     <!-- For Ldap Users -->
     <LdapServers>
       <LdapServer>
         <LdapServerName>vmw</LdapServerName>
       </LdapServer>
     </LdapServers>
   </Request>
   ```

   If the login succeeds, the API returns a response XML that indicates whether the user is authenticated.

   The following is an example program that calls the API.
   ```java
   /**
    * This method is for Login to vCenter-ChargeBack
    *
    * @param requestFilePath
    * @param baseURL
    * @param version
    * @throws IOException
    * @throws JDOMException
    * @throws NoSuchAlgorithmException
    * @throws KeyManagementException
    * @throws HttpException
    */
   public static void sampleLoginMethod(String requestFilePath, String baseURL, String version)
   throws IOException, JDOMException, NoSuchAlgorithmException, KeyManagementException, HttpException {
      PostMethod post = null;
      NameValuePair[] parameters = {new NameValuePair("version", version)};
      Document requestDocument = CommonUtil.getXMLDocument(requestFilePath);
      String bodyString = CommonUtil.xmlAsString(requestDocument);
      HttpClient client = new HttpClient();
   }
   ```
Add vCenter Server Information

You can add the vCenter Server instances in your virtualized environment to vCenter Chargeback Manager. This helps determine the computing resource utilization for the virtual machines and calculate the total costs.

To add vCenter Server information

1. Call the API by using the following syntax.

   `<HTTP_request_method> <Base_URL>/vCenterServer`

   For example, you can define a call as follows:

   ```
   POST https://123.123.123.123/vCenter-CB/api/vCenterServer?vesrion=2.5
   ```

2. In the request XML, specify the following information:

   - vCenter Server URL
   - vCenter Server Name
   - vCenter Server Description
   - vCenter Server Username
   - vCenter Server Password
   - PluginRegistered
   - StatsSync
   - Datasource URL
   - Data Source Name
   - Data Source User Name
   - Data Source Password
   - Data Source Type
   - DataSourceAuthType

   The following is an example request XML.

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <Request xmlns="http://www.vmware.com/vcenter/chargeback/2.0">
     <VCenterServers>
       <VCenterServer>
         <Url>101.102.107.21:443</Url>
         <Name>vc1</Name>
         <Description>vc1</Description>
         <UserName>administrator</UserName>
         <Password>xxxx</Password>
         <PluginRegistered>true</PluginRegistered>
         <StatsSync>true</StatsSync>
         <DataSourceUrl>10.112.107.21\sqlexp_vim</DataSourceUrl>
       </VCenterServer>
     </VCenterServers>
   </Request>
   ```
If the vCenter Server is successfully added, the API returns an XML response that provides the vCenter Server ID.

The following is an example program that calls the API.

```java
/**
 * This method is to add the vCenter-Server in vCenter-Chargeback application
 *
 * @param requestFilePath
 * @param baseURL
 * @throws IOException
 * @throws JDOMException
 * @throws NoSuchAlgorithmException
 * @throws KeyManagementException
 * @throws HttpException
 */

public static void sampleAddVCenterServerMethod(String requestFilePath, String baseURL)
    throws IOException, JDOMException, NoSuchAlgorithmException, KeyManagementException, HttpException {
    // Rest of the code...
}
```

---

**Add a Custom Chargeback Hierarchy**

Use the Add a Chargeback Hierarchy API to create a hierarchy with the given name and description.

**To add a custom Chargeback hierarchy**

1. Call the API by using the following syntax.

   ```xml
   <HTTP_request_method> <Base_URL>/hierarchy
   ```

   For example, you can define a call like this:

   ```xml
   POST https://123.123.123.123/vCenter-CB/api/hierarchy
   ```

2. In the request XML, specify a name and a description for the hierarchy.
The following is an example request XML.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Request>
  <Hierarchies>
    <Hierarchy>
      <Name>Test_Hierarchy</Name>
      <Description>Test Hierarchy</Description>
    </Hierarchy>
  </Hierarchies>
</Request>
```

If successful, the API returns details of the new hierarchy.

The following is an example program that calls the API.

```java
/**
 * This method is to add a vCenter-ChargeBack hierarchy in
 * vCenter-ChargeBack
 *
 * @param requestFilePath
 * @param baseURL
 * @throws IOException
 * @throws JDOMException
 * @throws NoSuchAlgorithmException
 * @throws KeyManagementException
 * @throws HttpException
 */
public static void sampleAddHierarchyMethod(String requestFilePath, String baseURL)
  throws IOException, JDOMException, NoSuchAlgorithmException,
       KeyManagementException, HttpException {
  PostMethod post = null;
  Document requestDocument = CommonUtil.getXMLDocument(requestFilePath);
  String bodyString = CommonUtil.xmlAsString(requestDocument);
       FakeSSLCertificateSocketFactory(), 443));
  HttpClient client = new HttpClient();
  String uri = "https://" + baseURL + "/vCenter-CB/api/hierarchy";
  System.out.println(uri);
  System.out.println(bodyString);
  try {
    post = new PostMethod(uri);
    post.setRequestBody(bodyString);
    client.executeMethod(post);
    System.out.println(post.getResponseBodyAsString());
  } finally {
    if (post != null) {
      post.releaseConnection();
    }
  }
}
```

### Add a vCenter Server Entity to the Chargeback Hierarchy

This task helps you add a vCenter Server entity under a specified parent entity in a Chargeback hierarchy.

**To add a vCenter Server entity to the Chargeback hierarchy**

1. Call the API by using the following syntax.

   `<HTTP_request_method> <Base_URL>/hierarchy/{hierarchyId}/entity/{parentEntityId}`

   For example, you can define a call like this:

   ```
   POST https://123.123.123.123/vCenter-CB/api/hierarchy/11/entity/101
   ```

2. In the request XML, specify the hierarchy ID, entity name, and description for the hierarchy.
The following is an example request XML.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Request>
  <VCenterServers>
    <VCenterServer id="502">
      <VCenterServerView id = "2"/>
      <Entities>
        <Entity id="1062"/>
      </Entities>
    </VCenterServer>
  </VCenterServers>
</Request>
```

If successful, the API returns an XML file that indicates the status.

The ID of the vCenter Server and the vCenter Server entities can be obtained by fetching the list of the vCenter Servers and the vCenter Server entities available in vCenter Chargeback Manager by using the following APIs:

- GET <Base_URL>/vCenterServers
- GET <Base_URL>/vcHierarchy/{vCenterServerId}

The following is an example program that calls the Add Entity API.

```java
/**
 * This method is for adding a new vCenter-Server entity under vCenter-Chargeback Hierarchy entity
 *
 * @param requestFilePath
 * @param baseURL
 * @param hierarchyId
 * @param CBEntityId
 * @throws IOException
 * @throws JDOMException
 * @throws NoSuchAlgorithmException
 * @throws KeyManagementException
 * @throws HttpException
 */
public static void sampleAddNewVCenterServerEntity(String requestFilePath, String baseURL, int hierarchyId, int CBEntityId, long startTime)
    throws IOException, JDOMException, NoSuchAlgorithmException, KeyManagementException, HttpException {
    PostMethod post = null;
    Document requestDocument = CommonUtil.getXMLDocument(requestFilePath);
    NameValuePair[] parameters = {new NameValuePair("startTime", String.valueOf(startTime))};
    String bodyString = CommonUtil.xmlAsString(requestDocument);
    String uri = "https://" + baseURL + "/vCenter-CB/api/hierarchy/" + hierarchyId + "/entity/" + CBEntityId;
    HttpClient client = new HttpClient();
    System.out.println(uri);
    System.out.println(bodyString);
    try {
        post = new PostMethod(uri);
        post.setQueryString(parameters);
        post.setRequestBody(bodyString);
        client.executeMethod(post);
        System.out.println(post.getResponseBodyAsString());
    } finally {
        if (post != null) {
            post.releaseConnection();
        }
    }
}
Add a Fixed Cost

A fixed cost is a definite cost that can be charged on an entity. Using the Add Fixed Cost API, you can create fixed costs for entities.

To add a fixed cost

1. Call the API using the following syntax.

   `<HTTP_request_method> <Base_URL>/fixedCost`

   For example, you can define a call like this:

   **POST https://123.123.123.123/vCenter-CB/api/fixedCost**

2. In the request XML, specify a name and a description for the fixed cost.

   The following is an example request XML.

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <Request>
     <FixedCosts>
       <FixedCost>
         <Name>Fixed Cost 1</Name>
         <Description>Fixed Cost 1 description</Description>
         <Currency id="1"/>
       </FixedCost>
     </FixedCosts>
   </Request>
   ```

   If the task is successful, the API returns an XML file that provides details of the new fixed cost.

   The following is an example program that calls the API.

   ```java
   /**
    * This method is to add a fixed cost in vCenter-ChargeBack
    *
    * @param requestFilePath
    * @param baseURL
    * @throws IOException
    * @throws JDOMException
    * @throws NoSuchAlgorithmException
    * @throws KeyManagementException
    * @throws HttpException
    */
   public static void sampleAddFixedCost(String requestFilePath, String baseURL) throws
   IOException, JDOMException, NoSuchAlgorithmException,
   KeyManagementException, HttpException {
     PostMethod post = null;
     Document requestDocument = CommonUtil.getXMLDocument(requestFilePath);
     String bodyString = CommonUtil.xmlAsString(requestDocument);
     HttpClient client = new HttpClient();
     FakeSSLCertificateSocketFactory(), 443));
     String uri = "https://" + baseURL + "/vCenter-CB/api/fixedCost";
     try {
       post = new PostMethod(uri);
       post.setRequestBody(bodyString);
       client.executeMethod(post);
       System.out.println(post.getResponseBodyAsString());
     } finally {
       if (post != null) {
         post.releaseConnection();
       }
     }
   }
   ```
Modify a Fixed Cost Value

Using the Modify Fixed Cost API, you can update the ID, value, and duration for a fixed cost.

To modify a fixed cost value

1. Call the API by using the following syntax.

   `<HTTP_request_method> <Base_URL>/fixedCost/{fixedCostId}/values`

   For example, you can define a call like this:

   **PUT https://123.123.123.123/vCenter-CB/api/fixedCost/{fixedCostId}/values**

2. In the request XML, specify a name and a description for the fixed cost. The following is an example request XML.

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <Request>
   <FixedCosts>
   <FixedCost id="1">
   <Values>
   <Value>
   <Cost>3.1415</Cost>
   <Duration id="1"/>
   </Value>
   </Values>
   </FixedCost>
   </FixedCosts>
   </Request>
   ```

   The response XML indicates whether the fixed cost value was successfully modified.

The following is an example program that calls the API.

```java
/**
 * This method is to modify the values of an existing fixedCost
 *
 * @param requestFilePath
 * @param baseURL
 * @param startTime
 * @param endTime
 * @throws IOException
 * @throws JDOMException
 * @throws NoSuchAlgorithmException
 * @throws KeyManagementException
 * @throws HttpException
 */
public static void sampleModifyFixedCostValues(String requestFilePath, String baseURL,
int fixedCostId, long startTime, long endTime) throws IOException,
JDOMException, NoSuchAlgorithmException, KeyManagementException, HttpException {
    PutMethod put = null;
    Document requestDocument = CommonUtil.getXMLDocument(requestFilePath);
    String bodyString = CommonUtil.xmlAsString(requestDocument);
FakeSSLCertificateSocketFactory(), 443));
    HttpClient client = new HttpClient();
    String uri = "https://" + baseURL + "/vCenter-CB/api/fixedCost/" + fixedCostId + 
"/values";
    System.out.println(uri);
    NameValuePair[] parameters = {new NameValuePair("startTime",
            String.valueOf(startTime)), new NameValuePair("endTime",
            String.valueOf(endTime))};
    try {
        put = new PutMethod(uri);
        put.setRequestBody(bodyString);
        put.setQueryString(parameters);
        client.executeMethod(put);
        System.out.println(put.getResponseBodyAsString());
    }
}
Generate a Report

Use the Generate Report API to send a request to create a report.

To generate a report

1. Call the API by using the following syntax.
   `<HTTP_request_method> <Base_URL>/report`
   For example, you can define a call like this:
   `POST https://123.123.123.123/vCenter-CB/api/report`

2. In the request XML, specify the following details about the hierarchy, entity, resource counters, and the pricing model.
   - Name of the report.
   - Description of the report.
   - Type of the report that you want to generate. You can generate a cost report, cost comparison report, or a usage report.
   - Start and end times of the period for which the report is to be generated.
   - IDs of the hierarchy, entity, pricing model, and computing resource for which you want to generate the report.

The following is an example request XML.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Request>
  <Reports>
    <Report>
      <MetaData>
        <Name>SampleReport</Name>
        <Description>Sample report description</Description>
        <ReportType>COST_REPORT</ReportType>
        <OwnedByName></OwnedByName>
        <ReportPeriod>
          <From>1320085800000</From>
          <To>1322418599999</To>
        </ReportPeriod>
        <Hierarchies>
          <Hierarchy id="1023">
            <Entities>
              <Entity id="1024" costModelId="30" />
            </Entities>
          </Hierarchy>
        </Hierarchies>
        <ComputingResources>
          <ComputingResource id="1" />
          <ComputingResource id="2" />
          <ComputingResource id="5" />
          <ComputingResource id="6" />
          <ComputingResource id="9" />
          <ComputingResource id="10" />
        </ComputingResources>
      </MetaData>
      <Configuration>
        <Computation>
          <Settings type="costVariance"/>
        </Computation>
      </Configuration>
    </Report>
  </Reports>
</Request>
```
vCenter Chargeback Manager queues this report request as a task and returns an XML that indicates the status of the task. You can use the Get Queued Task Status API to track the progress of the task. For more information on the Get Queued Task Status API, see the vCenter Chargeback Manager API Reference.

The following is an example program that calls the API.

```java
/**
 * This method is to generate report for a vCenter-ChargeBack hierarchy
 * entity in vCenter-ChargeBack
 *
 * @param requestFilePath
 * @param baseURL
 * @throws IOException
 * @throws JDOMException
 * @throws NoSuchAlgorithmException
 * @throws KeyManagementException
 * @throws HttpException
 */
public static void sampleGenerateReportMethod(String requestFilePath, String baseURL)
    throws IOException, JDOMException, NoSuchAlgorithmException,
    KeyManagementException, HttpException {
    PostMethod post = null;
    Document requestDocument = CommonUtil.getXMLDocument(requestFilePath);
    String bodyString = CommonUtil.xmlAsString(requestDocument);
        FakeSSLCertificateSocketFactory(), 443));
    String uri = "https://" + baseURL + "/vCenter-CB/api/report";
    HttpClient client = new HttpClient();
    System.out.println(uri);
    System.out.println(bodyString);
    try {
        post = new PostMethod(uri);
        post.setRequestHeader(bodyString);
        client.executeMethod(post);
        System.out.println(post.getResponseBodyAsString());
    } finally {
        if (post != null) {
            post.releaseConnection();
        }
    }
}
This chapter explains some of the tasks that you can perform by using vCenter Chargeback Manager APIs in an environment where vCenter Chargeback Manager is integrated with a third party billing system. You can use vCenter Chargeback Manager to perform cost measurement and generate resource utilization reports, which are provided to the billing system to charge customers.

The chapter includes the following topics:

- “Requirements for Code Examples” on page 25
- “Add Pricing Models” on page 28
- “Add a Billing Policy” on page 29
- “Retrieve List of Hierarchies” on page 30
- “Get Details of a Hierarchy” on page 31
- “Add Report Schedule for Hierarchy” on page 32
- “Get Report Schedule by Hierarchy Name” on page 33
- “Reschedule a Report” on page 35
- “Delete Report Schedule” on page 36
- “Get List of Archived Reports for a Hierarchy” on page 37
- “Get a Report as XML” on page 38

Requirements for Code Examples

This chapter provides code examples to explain how you can call the APIs. To run these code examples, you require the following JAR and helper class files.

JAR Files

The code examples need the following JAR files.

- commons-httpclient-3.1.jar
- commons-logging-1.1.1.jar
- jdom.jar

You must place these JAR files in the classpath.
CommonUtil Class

The code examples use the following CommonUtil class.

```java
public final class CommonUtil {

/**
 * This method writes out a formatted XML document to the Writer out.
 * @param doc
 * @param out
 * @throws IOException
 */
public static void printXML(Document doc, Writer out) throws IOException {
    XMLOutputter o = new XMLOutputter();
    Format newFormat = o.getFormat();
    newFormat.setIndent("  ");
    o.setFormat(newFormat);
    o.output(doc, out);
}

/**
 * This method returns the XML document as a string object
 * @param doc JDOM document representation of the XML
 * @return String representation of the XML document
 * @throws IOException
 */
public static String xmlAsString(Document doc) throws IOException {
    Writer w = new StringWriter();
    printXML(doc, w);
    return w.toString();
}

/**
 * This method reads the XML from a file and returns its representation as a
 * JDOM document
 * @param filePath path to the XML file
 * @return JDOM document representation of the XML
 * @throws IOException
 * @throws JDOMException
 */
public static Document getXMLDocument(String filePath) throws IOException, JDOMException {
    FileInputStream fis = null;
    try {
        fis = new FileInputStream(filePath);
        return getXMLDocument(fis);
    } finally {
        if (fis != null) {
            fis.close();
        }
    }
}

/**
 * This method reads an XML document from the input Stream and returns its
 * JDOM document representation
 * @param is input stream that has the XML content
 * @return jDOM document representation for the XML
 * @throws IOException
 * @throws JDOMException
 */
private static Document getXMLDocument(InputStream is) throws IOException, JDOMException {
    Document xml = null;
    SAXBuilder builder = new SAXBuilder();
    xml = builder.build(is);
    return xml;
}
}
```
FakeSSLCertificateSocketFactory Class

The code examples use the following class to access resources over the HTTPS protocol.

```java
/**
 * Helper class to accept self-signed certificate.
 */
public class FakeSSLCertificateSocketFactory implements SecureProtocolSocketFactory {
    private SSLContext sslContext;
    public FakeSSLCertificateSocketFactory() throws NoSuchAlgorithmException,
            KeyManagementException {
        sslContext = SSLContext.getInstance("SSL");
        sslContext.init(null, new TrustManager[] {new X509TrustManager() {
            @Override
            public void checkClientTrusted(X509Certificate[] ax509certificate, String s) throws
                    CertificateException {
                // Allow.
            }
            @Override
            public void checkServerTrusted(X509Certificate[] ax509certificate, String s) throws
                    CertificateException {
                // Allow.
            }
            @Override
            public X509Certificate[] getAcceptedIssuers() {
                return new X509Certificate[0];
            }
        }}, null);
    }
    @Override
    public Socket createSocket(String s, int i) throws IOException, UnknownHostException {
        return sslContext.getSocketFactory().createSocket(s, i);
    }
    @Override
    public Socket createSocket(String s, int i, InetAddress inetaddress, int j) throws
            IOException, UnknownHostException {
        return sslContext.getSocketFactory().createSocket(s, i, inetaddress, j);
    }
    @Override
    public Socket createSocket(String s, int i, InetAddress inetaddress, int j, HttpConnectionParams httpconnectionparams) throws IOException, UnknownHostException, ConnectTimeoutException {
        return sslContext.getSocketFactory().createSocket(s, i, inetaddress, j);
    }
    @Override
    public Socket createSocket(Socket socket, String s, int i, boolean flag) throws IOException, UnknownHostException {
        return sslContext.getSocketFactory().createSocket(socket, s, i, flag);
    }
}
```
Add Pricing Models

You can add multiple pricing models in vCenter Chargeback Manager. Defining multiple pricing models enables you to charge different sets of entities or hierarchies differently. It also enables you to compare the costs calculated using different pricing models for a hierarchy or a set of entities.

To add a pricing model

1. Call the Add Pricing Model API by specifying the following URL in your program.
   
   POST https://<ipaddress>/vCenter-CB/api/costModel

2. In the request XML, specify the following information.
   - Name: Specify Networking as the name of the pricing model.
   - Description: Provide a brief description for the Networking pricing model.
   - Currency ID: (Optional) Specify the ID of the currency to be set for this pricing model. If not specified, the currency ID will automatically be set to US Dollars (USD) or the global currency ID that was set during vCenter Chargeback Manager upgrade. For a list of currencies supported by vCenter Chargeback Manager, see the Appendix of the vCenter Chargeback Manager API Reference.

   The following is an example request XML.

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <Request>
     <CostModels>
       <CostModel>
         <Name>Networking</Name>
         <Description>Networking Pricing Model</Description>
         <Currency id="1"/>
       </CostModel>
     </CostModels>
   </Request>
   ```

3. Add the following pricing models.
   - Allocation Pool
   - Reservation Pool
   - Pay as you go

   The following is an example program that calls the Add Pricing Models API. This program assumes that the request XML is populated with the required information.

   ```java
   /**
   * This method is to add a pricing model in vCenter-ChargeBack
   * 
   * @param requestFilePath
   * @param baseURL
   * @throws IOException
   * @throws JDOMException
   * @throws NoSuchAlgorithmException
   * @throws KeyManagementException
   * @throws HttpException
   */
   public static void sampleAddCostModel(String requestFilePath, String baseURL) throws IOException, JDOMException, NoSuchAlgorithmException, KeyManagementException, HttpException {
     PostMethod post = null;
     Document requestDocument = CommonUtil.getXMLDocument(requestFilePath);
     String bodyString = CommonUtil.xmlAsString(requestDocument);
     HttpClient client = new HttpClient();
     FakeSSLCertificateSocketFactory(), 443));
     String uri = "https://" + baseURL + "/vCenter-CB/api/costModel;"
try {
    post = new PostMethod(uri);
    post.setRequestBody(bodyString);
    client.executeMethod(post);
    System.out.println(post.getResponseBodyAsString());
} finally {
    if (post != null) {
        post.releaseConnection();
    }
}

Add a Billing Policy

You can define custom billing policies as per your requirements. A billing policy defines an expression that is used for identifying the computing resources units to be considered for calculating the costs.

To add a billing policy

1. Call the Add Billing Policy API by specifying the following URL in your program:
   
   POST https://<ipaddress>/vCenter-CB/api/billingPolicy

2. In the request XML, provide the following details:
   
   - Name: Name of the billing policy.
   - Description: A brief description of the billing policy.
   - Expression: The expression defines attribute values that identify the computing resource units to be considered for cost calculation. For a list of computing resources and their associated attribute values, see the Appendix of the vCenter Chargeback Manager API Reference. For a computing resource the expression can be the maximum of available attributes. It can also account for the fixed costs, linked clones and the state of the virtual machine.

   The following is an example request XML.

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <Request>
     <BillingPolicy>
       <Name>New Billing Policy</Name>
       <Description>New Flexible Billing Policy Expression</Description>
       <Expression>cpu=max(usage, reservation);memory=max(usage,reservation);rest=usage;</Expression>
     </BillingPolicy>
   </Request>
   
   The following is an example program that calls the API. This program assumes that the request XML is populated with the required information.

   /**
    * This method is to add a billing policy in vCenter-ChargeBack
    * @param requestFilePath
    * @param baseURL
    * @throws IOException
    * @throws JDOMException
    * @throws NoSuchAlgorithmException
    * @throws KeyManagementException
    * @throws HttpException
    */
   public static void sampleAddBillingPolicy(String requestFilePath, String baseURL) throws
     IOException, JDOMException, NoSuchAlgorithmException, KeyManagementException, HttpException {
     PostMethod post = null;
     Document requestDocument = CommonUtil.getXMLDocument(requestFilePath);
     String bodyString = CommonUtil.xmlAsString(requestDocument);
     HttpClient client = new HttpClient();
String uri = "https://" + baseURL + "/vCenter-CB/api/billingPolicy";
try {
    post = new PostMethod(uri);
    post.setRequestBody(bodyString);
    client.executeMethod(post);
    System.out.println(post.getResponseBodyAsString());
} finally {
    if (post != null) {
        post.releaseConnection();
    }
}

Retrieve List of Hierarchies

Typically, you create new hierarchies in vCenter Chargeback Manager whenever new customer accounts are created and schedule reports to track resource usage for these accounts. To retrieve a list of hierarchies based on their names or on the time of their creation, you can use the Search API.

To retrieve a list of hierarchies

1. Call the Search API by using the following URL in your program.
   
   POST <API base URL>/search

2. In the request XML, use the following search criteria to get the hierarchy by name.
   
   <Criteria type="AND">
       <Filter name="hierarchyName" type="EQUAL" value="TestHierarchy" />
   </Criteria>

   For more information on all the available search filters, refer vCenter Chargeback Manager API Reference.

   The following is an example request XML.
   
   <?xml version="1.0" encoding="UTF-8"?>
   <Request>
       <SearchQueries>
           <SearchQuery id="hierarchy">
               <Criteria type="AND">
                   <Filter name="name" type="LIKE" value="%hierarchy1%"/>
                   <Filter name="desc" type="LIKE" value="%hierarchyDesc1%"/>
                   <Filter name="createdOn" type="BETWEEN" from="1230748200000" to="1295548200000"/>
               </Criteria>
               <SortBy>
                   <Params>
                       <Param index="1" order="DESC">createdOn</Param>
                   </Params>
               </SortBy>
               <Pagination>
                   <FirstResultCount>0</FirstResultCount>
                   <MaxResultCount>100</MaxResultCount>
               </Pagination>
           </SearchQuery>
       </SearchQueries>
   </Request>
   
   If the operation is successful, the response XML provides details of the retrieved hierarchies.
The following is an example program that calls the API. This program assumes that the request XML is populated with the required information.

```java
/**
 * This method get list of hierarchies from vCenter-ChargeBack
 *
 * @param requestFilePath
 * @param baseURL
 * @throws IOException
 * @throws HttpException
 * @throws NoSuchAlgorithmException
 * @throws KeyManagementException
 * @throws JDOMException
 */
public static void sampleGetHierarchies(String requestFilePath, String baseURL) throws HttpException, IOException, KeyManagementException, NoSuchAlgorithmException, JDOMException {
    PostMethod post = null;
    Document requestDocument = CommonUtil.getXMLDocument(requestFilePath);
    String bodyString = CommonUtil.xmlAsString(requestDocument);
    HttpClient client = new HttpClient();
    String uri = "https://" + baseURL + "/vCenter-CB/api/search";
    try {
        post = new PostMethod(uri);
        post.setRequestBody(bodyString);
        client.executeMethod(post);
        System.out.println(post.getResponseBodyAsString());
    } finally {
        if (post != null) {
            post.releaseConnection();
        }
    }
}
```

Get Details of a Hierarchy

After the Search API retrieves the new hierarchies, you can get more details about a specific hierarchy by calling the Get Hierarchy API.

To get details of a hierarchy

1. Call the Get Hierarchy API by using the following URL in your program.

   ```
   GET <API base URL>/hierarchy/{hierarchyId}
   ```

   If the operation is successful, the response XML contains details of the hierarchy.

The following is an example program that calls the API. This program assumes that the request XML is populated with the required information.

```java
/**
 * This method get hierarchy from vCenter-ChargeBack
 *
 * @param hierarchyId
 * @param baseURL
 * @throws IOException
 * @throws HttpException
 * @throws NoSuchAlgorithmException
 * @throws KeyManagementException
 * @throws JDOMException
 */
public static void sampleGetHierarchy(int hierarchyId, String baseURL) throws HttpException, IOException, KeyManagementException, NoSuchAlgorithmException, JDOMException {
    GetMethod get = null;
    HttpClient client = new HttpClient();
```
String uri = "https://" + baseURL + "/vCenter-CB/api/hierarchy/" + hierarchyId;
try {
    get = new GetMethod(uri);
    client.executeMethod(get);
    System.out.println(get.getResponseBodyAsString());
} finally {
    if (get != null) {
        get.releaseConnection();
    }
}

Add Report Schedule for Hierarchy

You can schedule reports to be generated at regular intervals. Add a report schedule for the hierarchy by using the Report Schedule API.

To add a report schedule for a hierarchy

1. Call the Report Schedule API by using the following URL in your program.

   POST <API base URL>/reportSchedule

   For example,

   POST https://123.123.123.123/vCenter-CB/api/reportSchedule

   The following is an example request XML.

   <?xml version="1.0" encoding="UTF-8"?>
   <Request>
   <ReportSchedules>
   <ReportSchedule>
   <MetaData>
   <Name>SampleReportSchedule</Name>
   <Description></Description>
   <ReportType>COST_REPORT</ReportType>
   <OwnedByName></OwnedByName>
   <ReportPeriodType>WEEKLY</ReportPeriodType>
   <ReportPeriod>
   <Weekly>
   <StartDayOfWeek>1</StartDayOfWeek>
   </Weekly>
   </ReportPeriod>
   <Hierarchies>
   <Hierarchy id="1023">
   <Entities>
   <Entity id="1024" costModelId="30">
   <Attributes operator="OR">
   <Attribute id="10" criteria="EQUALS">
   <Value>TestValue</Value>
   </Attribute>
   </Attributes>
   </Entity>
   </Entities>
   </Hierarchy>
   </Hierarchies>
   </ComputingResources>
   </ReportSchedule>
   </MetaData>
   <Configuration>
<Computation>
  <Settings type="costVariance">
    <Property name="enabled" value="true"/>
    <Property name="granularity" value="DAILY"/>
  </Settings>
</Computation>
</Configuration>
</ScheduleDetail>
</ReportSchedule>
</Request>

If the operation is successful, the response XML provides details of the report schedule.

The following is an example program that calls the API. This program assumes that the request XML is populated with the required information.

```java
/**
 * This method schedules report in vCenter-ChargeBack
 *
 * @param requestFilePath
 * @param baseURL
 * @throws IOException
 * @throws HttpException
 * @throws NoSuchAlgorithmException
 * @throws KeyManagementException
 * @throws JDOMException
 */
public static void sampleScheduleReport(String requestFilePath, String baseURL) throws
  HttpException, IOException, KeyManagementException, NoSuchAlgorithmException, JDOMException {
  PostMethod post = null;
  Document requestDocument = CommonUtil.getXMLDocument(requestFilePath);
  String bodyString = CommonUtil.xmlAsString(requestDocument);
  HttpClient client = new HttpClient();
    FakeSSLCertificateSocketFactory(), 443));
  String uri = "https://" + baseURL + "/vCenter-CB/api/reportSchedule";
  try {
    post = new PostMethod(uri);
    post.setRequestHeader(bodyString);
    client.executeMethod(post);
    System.out.println(post.getResponseBodyAsString());
  } finally {
    if (post != null) {
      post.releaseConnection();
    }
  }
}
```

**Get Report Schedule by Hierarchy Name**

Depending on how you charge each customer, you might have mapped entities in the hierarchy to appropriate pricing models. You can call the Search API to get the report schedule for the hierarchy and verify the entity to pricing model mappings.
To get schedule by hierarchy name

1. Call the Search API by using the following URL in your program.
   
   POST <API base URL>/search

2. In the request XML, specify the following parameters to search schedules. You can use any of these parameters alone or in combination with the others.
   - `name`
   - `desc`
   - `hierarchyName`
   - `costModelName`

3. You can use search operators such as `EQUAL`, `NOT_EQUAL`, `BETWEEN`, `GT`, `LT`, `GT_EQUAL`, `LT_EQUAL`, `NULL`, `NOT_NULL`, `LIKE`, `NOT_LIKE` to filter your search.

   The following is an example request XML.

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <Request>
   <SearchQueries>
       <SearchQuery id="reportSchedule">
           <Criteria type="AND">
               <Filter name="name" type="LIKE" value="%reportSchedule1%" />
               <Filter name="desc" type="LIKE" value="%reportScheduleDesc1%" />
               <Filter name="costModelName" type="EQUAL" value="costModel1"/>
           </Criteria>
           <SortBy>
               <Params>
                   <Param index="1" order="DESC">name</Param>
               </Params>
           </SortBy>
           <Pagination>
               <FirstResultCount>0</FirstResultCount>
               <MaxResultCount>100</MaxResultCount>
           </Pagination>
       </SearchQuery>
   </SearchQueries>
   </Request>
   
   The following is an example program that calls the API. This program assumes that the request XML is populated with the required information.

   ```java
   public static void sampleGetScheduleByHierarchyName(String requestFilePath, String baseURL) throws HttpException, IOException, KeyManagementException, NoSuchAlgorithmException, JDOMException {
       PostMethod post = null;
       Document requestDocument = CommonUtil.getXMLDocument(requestFilePath);
       String bodyString = CommonUtil.xmlAsString(requestDocument);
       HttpClient client = new HttpClient();

       String uri = "https://" + baseURL + "/vCenter-CB/api/search";
       try {
           post = new PostMethod(uri);
           post.setRequestBody(bodyString);
           client.executeMethod(post);
   ```
System.out.println(post.getResponseBodyAsString());
} finally {
    if (post != null) {
        post.releaseConnection();
    }
}

### Reschedule a Report

You can update a report schedule using the Reschedule Report API. The Reschedule Report API updates the schedule information. If you want to update the entity to pricing model mapping, you must delete the schedule and create a new schedule.

**To reschedule a report**

1. Call the Reschedule Report API by using the following URL in your program.

   ```
   PUT <API base URL>/reportSchedule/{scheduleId}
   ```

   For example,

   ```
   PUT https://123.123.123.123/vCenter-CB/api/reportSchedule/5
   ```

   The following is an example request XML.

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <Request xmlns="http://www.vmware.com/vcenter/chargeback/2.0">
     <ReportSchedules>
       <ReportSchedule>
         <MetaData>
           <ReportPeriodType>DAILY</ReportPeriodType>
           <ReportPeriod>
             <Daily>
               <DaysBefore>1</DaysBefore>
             </Daily></ReportPeriod>
         </MetaData>
         <ScheduleDetail>
           <FireTime>
             <Hour>13</Hour>
             <Minute>59</Minute>
           </FireTime>
           <Recurrence type="DAILY">
             <RepeatInterval>1</RepeatInterval>
           </Recurrence>
           <Range>
             <StartDate>1321295400000</StartDate>
             <Count>-1</Count>
           </Range>
         </ScheduleDetail>
       </ReportSchedule>
     </ReportSchedules>
   </Request>
   ```

   The following is an example program that calls the API. This program assumes that the request XML is populated with the required information.

   ```
   /**
    * This method Re-schedule report in vCenter-ChargeBack
    *
    * @param requestFilePath
    * @param baseURL
    * @param scheduleId
    * @throws IOException
    * @throws HttpException
    * @throws NoSuchAlgorithmException
    * @throws KeyManagementException
    * @throws JDOMException
    */
   ```
public static void sampleRescheduleReport(String requestFilePath, String baseURL, int scheduleId) throws HttpException, IOException, KeyManagementException, NoSuchAlgorithmException, JDOMException {
    PutMethod put = null;
    Document requestDocument = CommonUtil.getXMLDocument(requestFilePath);
    String bodyString = CommonUtil.xmlAsString(requestDocument);
    HttpClient client = new HttpClient();
    String uri = "https://" + baseURL + "/vCenter-CB/api/reportSchedule/" + scheduleId;
    try {
        put = new PutMethod(uri);
        put.setRequestBody(bodyString);
        client.executeMethod(put);
        System.out.println(put.getResponseBodyAsString());
    } finally {
        if (put != null) {
            put.releaseConnection();
        }
    }
}

Delete Report Schedule

Currently, the Reschedule Report API updates only the schedule information and not the entity to pricing model mapping. So, you need to delete the report schedule and create a new one if you want to add, modify, or delete any entity to pricing model mappings if any new customer is getting added.

To delete a report schedule

1. Call the Delete Schedule API by using the following URL in your program.

   DELETE <API base URL>/reportSchedule/{scheduleId}

   For example,

   DELETE https://123.123.123.123/vCenter-CB/api/reportSchedule/5

   The response XML indicates that the delete operation is successful.

   The following is an example program that calls the API. This program assumes that the request XML is populated with the required information.

   /**
   * This method is to delete report schedule in vCenter-ChargeBack
   *
   * @param baseURL
   * @param scheduleId
   * @throws IOException
   * @throws HttpException
   * @throws NoSuchAlgorithmException
   * @throws KeyManagementException
   * @throws JDOMException
   */
   public static void sampleDeleteReportSchedule(String baseURL, int scheduleId) throws HttpException, IOException, KeyManagementException, NoSuchAlgorithmException, JDOMException {
       DeleteMethod delete = null;
       HttpClient client = new HttpClient();
       String uri = "https://" + baseURL + "/vCenter-CB/api/reportSchedule/" + scheduleId;
       try {
           delete = new DeleteMethod(uri);
           client.executeMethod(delete);
           System.out.println(delete.getResponseBodyAsString());
       } finally {
           if (delete != null) {
               delete.releaseConnection();
           }
       }
   }
Get List of Archived Reports for a Hierarchy

For a specific hierarchy, you can get a list of archived reports that were generated after a specific time. To do this, you can use the Search API.

To get list of archived reports for a hierarchy

1. Call the Search API by using the following URL in your program.

   POST <API base URL>/search

2. In the request XML, use the following section to search for reports with the specific hierarchy name and created after or equal to a time passed in milliseconds since January 1, 1970.

   <Criteria type="AND">
     <Filter name="hierarchyName" type="EQUAL" value="TestHierarchy" />
     <Filter name="createdOn" type="GT_EQUAL" value="1272639780140" />
   </Criteria>

   The following is an example request XML.

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <Request>
     <SearchQueries>
       <SearchQuery id="report">
         <Criteria type="AND">
           <Filter name="name" type="LIKE" value="%report1%" />
           <Filter name="desc" type="LIKE" value="%reportDesc1%" />
           <Filter name="costModelName" type="EQUAL" value="costModel1"/>
         </Criteria>
         <SortBy>
           <Params>
             <Param index="1" order="DESC">totalCost</Param>
           </Params>
         </SortBy>
         <Pagination>
           <FirstResultCount>0</FirstResultCount>
           <MaxResultCount>100</MaxResultCount>
         </Pagination>
       </SearchQuery>
     </SearchQueries>
   </Request>
   ```

   The response XML contains a list of reports that match the search criteria.

   The following is an example program that calls the API. This program assumes that the request XML is populated with the required information.

   ```java
   public static void sampleGetArchivedReports(String requestFilePath, String baseURL) throws HttpException, IOException, KeyManagementException, NoSuchAlgorithmException, JDOMException {
     PostMethod post = null;
     Document requestDocument = CommonUtil.getXMLDocument(requestFilepath);
     String bodyString = CommonUtil.xmlAsString(requestDocument);
   }
   ```
Get a Report as XML

Get a report in the XML format using the Export Report API.

1. Use the following syntax to call the Export Report API.

   ```java
   GET <API base URL>/report/{reportId}/export?exportFormat=XML
   ```

   For example,

   ```java
   ```

   The API queues the export report request as a task and returns the status of the queued task. You can check the status of the queued task using the Get Queued Task Status API. After the exported report is ready, the Export Report API returns the report in XML format.

2. In the response XML, check the ComputeData section for the cost details.

   For a detailed description of all the elements in the response XML, see the vCenter Chargeback Manager API Reference Guide.

   The following is an example program that calls the API. This program assumes that the request XML is populated with the required information.

   ```java
   /*
   * This method get report as XML vCenter-ChargeBack
   * ...
   */
   public static void sampleGetReportAsXml(int reportId, String baseURL) throws
   IOException, HttpException, KeyManagementException, NoSuchAlgorithmException,
   JDOMException {
   GetMethod get = null;
   HttpClient client = new HttpClient();
   FakeSSLCertificateSocketFactory(), 443));
   String uri = "https://" + baseURL + "/vCenter-CB/api/search";
   try {
   post = new PostMethod(uri);
   post.setRequestBody(bodyString);
   client.executeMethod(post);
   System.out.println(post.getResponseBodyAsString());
   } finally {
   if (post != null) {
   post.releaseConnection();
   }
   }
   }
   ```
Index

A
about,
vCenter Chargeback Manager APIs 7
add
  billing policy 29
  entity 19
  fixed cost 21
  hierarchy 18
  pricing model 28
  report schedule 32
  vCenter Server 17
add vCenter Server
  example program 18
  request XML 17
add vCenter Server API
  add vCenter Server 17
Apache Tomcat 7
API
  add billing policy 29
  add vCenter Server entity 19
  login 16
  request 8
  response 8
  syntax 9
  versioning 10
API syntax 17
  components 9
API versioning,
  about 10
  example 10
APIs
  export report 38
  get hierarchy 31
  Reschedule Report 35
  Search 33
  search 30
architecture,
  REST-based 7
  vCenter Chargeback Manager APIs 7
archived reports
  example program 37
  get 37
  request XML 37

B
billing policy
  add 29
  example program 29
  request XML 29

C
code examples
  requirements 13, 25
common
  XML elements 9
CommonUtil class
  example 14, 26
components
  API syntax 9
  Error element 9

D
delete
  report schedule 36

E
entity
  add 19
Error
  components 9
  response 9
example program
  add report schedule 33
  add vCenter Server 18
log in API 16
examples
  class files 13, 25
  CommonUtil class 14, 26

F
fixed cost
  add 21
  modify value 22

G
generate
  report 23
get
  archived reports 37
  hierarchies 30
  hierarchy 31
get details
  hierarchy 31
H
   helper class
   for examples 13, 25
   hierarchy
   add 18
   get details 31
J
   JAR files
   for examples 13, 25
L
   list of hierarchies
   get 30
   log in
   request XML 16
   log in API
   example program 16
   login 13
   API 16
M
   modify
   value of fixed cost 22
P
   pricing model
   add 28
R
   report
   generate 23
   in XML 38
   reschedule 35
   XML format 38
   report schedule
   add 32
   delete 36
   example program 33
   for hierarchy 33
   request XML 32
   request
   API 8
   request XML
   add vCenter Server 17
   archived reports 37
   login 16
   login API 16
   report schedule 32
   requirements
   for code examples 13, 25
   reschedule
   report 35
   response
   Error 9
   XML 8
S
   search
   APIs 30
   syntax 17
   API 9
T
   target namespace 10
V
   vCenter Chargeback Manager API
   about 7
   vCenter Server
   add 17
X
   XML
   report 38
   XML elements
   common 9