Getting Started with the Prime Cable Provisioning API

This chapter describes how to start the API clients and process a batch.

Startup Process for API Client

The startup process for an API client interaction involves:

- Configuring the System, page 6-1.
- Executing the API Client, page 6-2.

Configuring the System

Before executing a simple client, ensure that you have completed the tasks listed in this section.

Note These tasks are part of an initial configuration workflow that you must complete before executing a simple client for the first time. Thereafter, you can execute any number of simple clients.

Table 6-1 System Configuration Workflow

Task	Refer to
1. Install Java Development Kit version 1.6.	Sun Microsystems support site
2. Ensure that files bpr.jar, bacbase.jar, and bac-common.jar are available in the classpath. These *.jar* files are located in the *BPR_HOME/lib* directory.	—
3. Access the Prime Cable Provisioning administrator user interface and ensure that the password that you set for the default **admin** username matches the password that you set on the RDU. The default password is **changeme**.	*Cisco Prime Cable Provisioning 6.2 User Guide*

Executing the API Client

To execute a simple API client:

Note This procedure uses the *AddDeviceExample.java* classfile as an example.

Step 1 Compile the API classfile using the following code:

```
javac -classpath .:bpr.jar:bacbase.jar:bac-common.jar java_source_file
```

For example:

```
javac -classpath .:bpr.jar:bacbase.jar:bac-common.jar AddDeviceExample
```

Note This example assumes that the bpr.jar, bacbase.jar and bac-common.jar files exist in the local directory.

Step 2 Execute the API classfile using the following code:

```
java -cp .:bpr.jar:bacbase.jar:bac-common.jar class_file
```

For example:

```
java -cp .:bpr.jar:bacbase.jar:bac-common.jar AddDeviceExample
```

Step 3 Verify the results.

For example, the AddDeviceExample print success or failure messages. If there is no error, the following message appears:

```
Successfully provisioned device with identifier [OUI-serial-12345]
```

You can also verify the results for the device record from the administrator user interface from the **Devices > Manage Device** page. For more information, see the *Cisco Prime Cable Provisioning 6.2 User Guide*.

Processing a Batch

This section describes how you can connect to the RDU, create a batch, post the batch to the RDU, and verify the result.

Note This procedure uses the *AddDeviceExample.java* classfile as an example.

Step 1 Create a connection to the Provisioning API Command Engine (PACE).

```
// The PACE connection to use throughout the example.  When
// executing multiple batches in a single process, it is advisable
// to use a single PACE connection that is retrieved at the start
// of the application.  When done with the connection, YOU MUST
// explicitly close the connection with the releaseConnection()
// method call.
```

```
PACEConnection connection = null;

// ----------------------------------------------------------------
//
// 1) Connect to the Regional Distribution Unit (RDU).
//
//    The parameters defined at the beginning of this class are
//    used here to establish the connection.  Connections are
//    maintained until releaseConnection() is called.  If
//    multiple calls to getInstance() are called with the same
//    arguments, you must still call releaseConnection() on each
//    connection you received.
//
//    The call can fail for one of the following reasons:
//    - The hostname / port is incorrect.
//    - The authentication credentials are invalid.
//
// ----------------------------------------------------------------
try
{
    connection = PACEConnectionFactory.getInstance(
        // RDU host
        rduHost,
        // RDU port
        rduPort,
        // User name
        userName,
        // Password
        password);
}
catch (PACEConnectionException pce)
{
    // failed to get a connection
    System.out.println("Failed to establish a PACEConnection to ["
            + userName + "@" + rduHost + ":" + rduPort + "]; " +
            pce.getMessage());
    throw new RuntimeException(pce.getMessage());
}
catch (RDUAuthenticationException bae)
{
    // failed to get a connection
    System.out.println("Failed to establish a PACEConnection to ["
            + userName + "@" + rduHost + ":" + rduPort + "]; " +
              bae.getMessage());
       throw new RuntimeException(bae.getMessage());
}
// ----------------------------------------------------------------
```

Step 2 Get a new batch instance.

```
// ----------------------------------------------------------------
//
// 2) Get a new batch instance.
//
//    To perform any operations in the Provisioning API, you must
//    first start a batch.  As you make commands against the batch,
//    nothing  actually start until you post the batch.
//    Multiple batches can be started concurrently against a
//    single connection to the RDU.
//
// ----------------------------------------------------------------
Batch myBatch = connection.newBatch(
                        // No reset
                        ActivationMode.NO_ACTIVATION,
                        // No need to confirm activation
                        ConfirmationMode.NO_CONFIRMATION,
                        // No publisining to external database
                        PublishingMode.NO_PUBLISHING);
// -----------------------------------------------------------------
```

Step 3 Register the `AddDeviceExample`( ) call with the batch.

```
// -----------------------------------------------------------------
//
// 3) Register the add(...) call with the batch.
//
//    Add to the batch the add(...) call.  This  make
//    the batch add the device during the post() operation.  If
//    multiple methods are added to a batch, they  be executed
//    in the order they are registered.  For example, you could
//    add a device and then modify it successfully in a batch.
//
//    The host name and domain name only needs to be specified if the
//    device should have an explicit name assigned to it -- and this is
//    only really useful if you have dynamic DNS enabled in CNR.
//    Properties can be used to store additional information that
//    should be maintained by BPR.  This data  be returned as a
//    response to a query for device details.
//
// -----------------------------------------------------------------
myBatch.add(
// Device type
DeviceType.DOCSIS,
// deviceID list with MACAddress
deviceIDList,
// Host name - Not used in this example
null,
// Domain Name - Not used in this example
null,
// ownerID
accountNumber,
// classOfService - Use default COS
null,
// dhcpCriteria - Use default DHCP Criteria
null,
// properties
null);

        // -----------------------------------------------------------------
```

Step 4 Post a batch to the RDU.

```
//
// 4) Post the batch to the server.
//
//    Executes the batch against the RDU.  All of the
//    methods are executed in the order entered and the data
//    changes are applied against the embedded database in RDU.
//
// -----------------------------------------------------------------
BatchStatus batchStatus = null;
try
{
    batchStatus = myBatch.post();
}
catch (ProvisioningException pe)
{
    System.out.println("Failed to provision device with identifer ["
        + deviceId + "]; " + pe.getMessage());

    throw new RuntimeException(pe.getMessage());
}

// -----------------------------------------------------------------
```

Step 5 Verify the result of the connection.

```
//
// 5) Check to see if the batch was successfully posted.
//
//    Verify if any errors occurred during the execution of the
//    batch.  Exceptions occur during post() for truly exception
//    situations such as failure of connectivity to RDU.
//    Batch errors occur for inconsistencies such as no lease
//    information for a device requiring activiation.  Command
//    errors occur when a particular method has problems, such as
//    trying to add a device that already exists.
//
// -----------------------------------------------------------------
if (batchStatus.isError())
{
    // Batch error occurred.
    // we need to determine if it was a batch error or a
    // command error that caused this failure

    if (batchStatus.getFailedCommandIndex() == -1)
    {
        // this is a batch only error
        // get the error code and get the error message
        final StringBuilder msg = new StringBuilder(128);
        msg.append("Batch with ID [");
        msg.append(batchStatus.getBatchID());
        msg.append("] failed with error code [");
        msg.append(batchStatus.getStatusCode());
        msg.append("]. [");
        msg.append(batchStatus.getErrorMessage());
        msg.append("].");


        // throw an exception or log the message
        System.out.println("Failed to add device with identifier ["
                + deviceId + "]; " +  msg.toString());
    }
    else
    {
        // this is a batch error caused by a command
        final CommandStatus commandStatus =
            batchStatus.getFailedCommandStatus();

        // get the error code and get the error message
        final StringBuilder msg = new StringBuilder(128);
        msg.append("Batch with ID [");
        msg.append(batchStatus.getBatchID());
        msg.append("] failed with command error code [");
        msg.append(commandStatus.getStatusCode());
        msg.append("]. [");
        msg.append(commandStatus.getErrorMessage());
        msg.append("].");

        // throw an exception or log the message
        System.out.println("Failed to add device with identifier ["
               + deviceId + "]; " + msg.toString());
    }
```

```
}
else
{
    // Successfully added device
    System.out.println("Successfully added device with identifier ["
        + deviceId + "]");
}
```

Step 6 Release the connection to the RDU.

```
// ----------------------------------------------------------------
//
// 6) Release the connection to the RDU.
//
//    Once the last batch has been executed, the connection can
//    be closed to the RDU.  It is important to explictly
//    close connections since it helps ensure clean shutdown of
//    the Java virtual machine.
//
// ----------------------------------------------------------------
connection.releaseConnection();
```