Batches and Commands

This chapter provides an overview of a batch, the commands contained in a batch, and how a batch is processed in Prime Cable Provisioning.

Overview

A batch object:

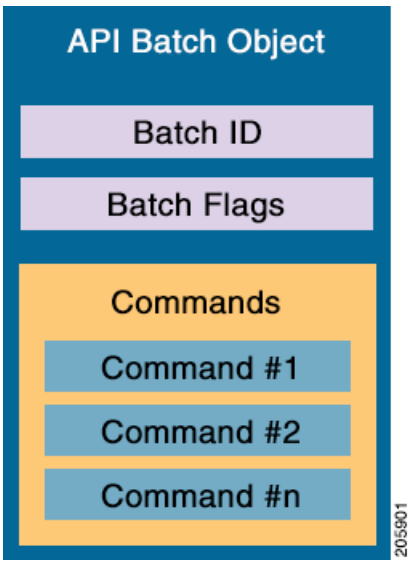
- Is a container for commands that the RDU must execute.
- Contains methods that control how the RDU executes the commands and returns results.

A command represents an operation that is performed on an object in the RDU database. For example, to add a new device, the client issues an add command from the API to the RDU.

The batch lifecycle (create, post, execute, return results) demands two entities to communicate over a network. For this communication, a provisioning client in Prime Cable Provisioning submits API requests to the RDU in the form of batches that contain single or multiple commands.

Figure 4-1 illustrates the concept of batch processing.

Figure 4-1 API Batch Object



Batches are atomic units; either all the commands in the batch succeed or none of the commands succeeds. If the batch fails, the RDU restores changes that were made to its database. The RDU executes the commands in the same sequence in which they are added to the batch. For more information on batch identification, see Identifying a Batch, page 4-2. For more information on batch flags, see Batch Processing Flags, page 4-3.

Batch Rules

To execute a batch successfully, ensure that you follow rules listed below:

- A batch must contain between 1 and 100 commands. You cannot execute a batch with no commands, or one with more than 100 commands.
- Commands in a batch must either be read or write. You cannot combine read and write commands in a batch. For example, the same batch cannot contain a get device details command (read) as well as an add device command (write).

  Note Commands that perform device operations are write commands.

- Batch commands must relate to device or system configuration. You cannot combine device-related and system-related commands in a batch. For example, you cannot combine a modify Class of Service command (system) and an add device command (device) in the same batch.
- When a batch includes a command that interacts with a device record in the RDU through a device operation or an automatic activation flag, all commands in the batch must relate to the same device record in the RDU.
- If you have multiple device operations, each device operation should be submitted in a single batch.

Identifying a Batch

Every batch that the RDU executes has a unique batch identifier. The batch identifier that the RDU Java client library generates includes the hostname of the local client server and a random number that increments.

The batch identifier helps you to:

- Retrieve batch status from the RDU.
- Correlate the respective batch events in the RDU.

While the RDU Java client library automatically generates a batch identifier, you can specify your own batch identifier based on your requirements.

Note We recommend that you use the batch identifiers that the RDU Java client library generates for you.

If you generate your own batch identifier, ensure that you clearly identify the local client server.

Tip If you have a global transaction identifier, it can be a good idea to include it in the batch identifier in order to monitor the transaction throughout the entire system.

If the RDU detects a duplicate batch identifier, it rejects that batch. Submitting batches with batch identifiers that have already been processed may lead to failure and unexpected results.

You can generate a batch identifier in one of two following ways:

- Using the RDU Java client library —To use the RDU Java client library, use the newBatch methods on the Provisioning API Command Engine (PACE) connection object for a batch without the batch identifier parameter.

  Use the following code to generate a batch identifier using a RDU Java client library:

```
public Batch newBatch()

public Batch newBatch(ActivationMode activation)

public Batch newBatch(PublishingMode publishing)

public Batch newBatch(ActivationMode activation, ConfirmationMode confirmation)

public Batch newBatch(ActivationMode activation, ConfirmationMode confirmation,
                            PublishingMode publishing)

public Batch newBatch(ActivationMode activation,
                            PublishingMode publishing)
```

- By specifying your own identifier — To generate your own batch identifier, use the newBatch methods on the PACE connection object containing the batch identifier parameter.

  Use the following code to generate a batch identifier by specifying your own identifier:

```
public Batch newBatch(String batchId)

public Batch newBatch(String batchId,
                            ActivationMode activation)

public Batch newBatch(String batchId,
                            PublishingMode publishing)

public Batch newBatch(String batchId,
                            ActivationMode activation,
                            ConfirmationMode confirmation)

public Batch newBatch(String batchId,
                            ActivationMode activation,
                            ConfirmationMode confirmation,
                            PublishingMode publishing)

public Batch newBatch(String batchId,
                            ActivationMode activation,
                            PublishingMode publishing)
```

Batch Processing Flags

Batch processing flags control:

- Batch interaction with a device.

- Notifications of batches to external systems. These notifications detail the changes that are made by various operations in a batch.

Prime Cable Provisioning supports the following processing flags, each of which is described in subsequent sections:

- Reliable, see Setting the Reliable Flag, page 4-4.
- Activation, see Setting the Activation Flag, page 4-5.
- Confirmation, see Setting the Confirmation Flag, page 4-6.
- Publishing, see Setting the Publishing Flag, page 4-6.
- Optimistic Locking, see Setting the Optimistic Locking Flag, page 4-7.

Setting the Reliable Flag

Communication between the client and the RDU breaks if:

- The client restarts after posting a batch.
- The RDU restarts after receiving a batch.
- The network connection breaks when the results are being sent. Subsequently, the results are lost.

To handle such issues, Prime Cable Provisioning provides a reliable batch flag. When you enable the reliable flag for a batch, the RDU stores the batch on receiving it, and even if the RDU restarts, the batch is guaranteed to be executed after the restart.

Note You can enable the reliable batch flag for batches that contain write commands, such as add, change, or delete. The get operation is not supported in the reliable mode.

After the batch is executed, the RDU stores the results in its database. Subsequently, the client can obtain results for the batches even after an RDU restart. To obtain the results, the client uses a join operation and the thread blocks till the results are returned or a timeout occurs. If the RDU did not receive the batch, or cleared the results from its database, an error appears. At a time, the RDU stores the results of 1000 reliable batches that were last executed.

Note We recommend that you store all batch identifiers of reliable batches to the disk, before you post a batch. By storing the batch identifiers, the RDU Java client library can query for results even if a client restart occurs.

- To join a reliable batch with a batch identifier using the PACE Connection object:
  - With a timeout:

```
final BatchStatus batchStatus = connection.join(batchId, 5000);
```

Note We recommend that you use a timeout value when using the join feature for reliable batches. Also, because reliable batches add a significant load to the RDU, use it only when client and network reliability outweigh the performance impact.

  - Without a timeout:

```
final BatchStatus batchStatus = connection.join(batchId);
```

- To force a batch to be reliable before submitting a synchronous or asynchronous post, use the following code:

```
// make it reliable
batch.forceBatchReliable();
```

For information on synchronous and asynchronous batches, see Batch Processing Modes, page 4-9.

Setting the Activation Flag

You can use the activation flag in batches that contain write commands and operate on a single device. The activation flag is of two types:

- No Activation—Executes by updating the RDU database and the appropriate DPE caches.

  Batches that include commands for on-connect device operations must use the no-activation flag.

- Automatic Activation—Executes by persisting the changes in the RDU database and by trying to establish contact with the device to obtain the latest configuration.

  Batches that include commands for all immediate device operations must use the automatic-activation flag.

You can mark a batch using the no-activation flag or the automatic-activation flag.

For example, consider a batch that contains a change Class of Service command for a device. If you execute the batch with the no-activation flag, the Class of Service of the device is changed, and the resulting new configuration is sent to the DPEs in the provisioning group. The new data is available in the appropriate DPEs for the next device session. On the other hand, if you execute the same batch with an automatic-activation flag, the RDU sends the new configuration to the provisioning group.

Activation does not verify if the configuration was successfully applied on the device. When you execute a batch with the automatic-activation flag, the batch becomes reliable. Also, activation involves updating the RDU database and pushing the updated configuration for the device to the DPE, automatically. For details on controlling this behavior using the Confirmation flag, see Setting the Confirmation Flag, page 4-6.

Note You can augment or replace the activation logic in the RDU during deployment using an extension. For more information, see the *Cisco Prime Cable Provisioning 5.1 User Guide*.

- You can create a batch with no activation in one of two following ways:
  - Without specifying the flag. Because no-activation is the default, batches are created with the no-activation flag.

```
final Batch batch = connection.newBatch();
```

  - By explicitly setting the flag.

```
final Batch batch = connection.newBatch(
                          ActivationMode.NO_ACTIVATION);
```

- You can create a batch with automatic activation using the following code:

```
final Batch batch = connection.newBatch(
                          ActivationMode.AUTOMATIC);
```

Setting the Confirmation Flag

You can use the confirmation flag to control the behavior of batch activation. You must use the confirmation flag only in batches that have the automatic-activation flag set.

The confirmation flag communicates with the RDU on how the processing of a batch should proceed if there are warnings or errors during activation. For more information on warnings or errors during activation, see Batch Warnings, page 4-16.

Prime Cable Provisioning supports two types of confirmation flags:

- No confirmation
- Custom confirmation.

Unless you specify otherwise, a batch is created with the no confirmation flag.

When you execute a batch with the no-confirmation flag, warnings or errors during activation do not cause the batch to fail. Instead, the batch results contain a warning indicating that activation issues occurred. The batch proceeds and database updates are committed.

When you execute a batch with the custom-confirmation flag and a warning occurs during activation, the batch results contain the warning. The batch proceeds, committing the database updates. However, if an error occurs during activation, and the batch results contain the error, the batch fails, and the database updates get rolled back.

Note You can replace or augment the activation code in the RDU so that the errors or warnings that appear depend on the code in use.

You can create a batch with a no-confirmation flag or a custom-confirmation flag.

- You can create a batch with the no-confirmation flag, using the following code:

```
final Batch batch = connection.newBatch(
                                ActivationMode.AUTOMATIC);
```

- You can create a batch with the custom-confirmation flag, using the following code:

```
final Batch batch = connection.newBatch(
                          ActivationMode.AUTOMATIC,
                          ConfirmationMode.CUSTOM_CONFIRMATION);
```

Setting the Publishing Flag

You can use publishing plug-ins to include custom code that helps notify the external entities of changes the batch make to the RDU database. For information on creating publishing plug-ins in the RDU, see the *Cisco Prime Cable Provisioning 5.1 User Guide*.

You can set the publishing flag in one of three ways:

- No publishing—The publishing plug-in is not called within the batch.
- Publishing with no confirmation—The publishing plug-in is executed. If an error occurs, the batch proceeds and any database change is updated.
- Publishing with confirmation—The publishing plug-in is executed. If an errors occurs, the batch fails and the database updates are rolled back.

Note When you mark a batch with the publishing with confirmation flag, the batch automatically becomes reliable.

You must explicitly specify if a batch is to be created with publishing; otherwise, batches are created using the no-publishing flag.

- You can create a batch with the no-publishing flag in one of two following ways:
  - Without setting any flag. Because the no-publishing flag is the default setting, a batch is thus created:

```
final Batch batch = connection.newBatch();
```

  - By explicitly setting the no-publishing flag:

```
final Batch batch = connection.newBatch(
                          PublishingMode.NO_PUBLISHING);
```

- You can create a batch with the publishing no-confirmation flag using:

```
final Batch batch = connection.newBatch(
                        PublishingMode.PUBLISHING_NO_CONFIRMATION);
```

- You can create a batch with the publishing-with-confirmation flag using:

```
final Batch batch = connection.newBatch(
                        PublishingMode.PUBLISHING_CONFIRMATION);
```

Setting the Optimistic Locking Flag

Because the API client executes in a client-server model, a time interval occurs between a get and a modify cycle. You can use the optimistic locking flag to prevent inconsistent changes being made to devices by different clients, simultaneously.

When you perform a get operation for an object (such as a device), the details map contains the GenericObjectKeys.OID_REVISION_NUMBER key. The value for this key is an object identifier that is encoded with the current revision number for the object. You can add this revision number to the batch to ensure that the object is not changed before the changes in your batch are applied. If the object has changed, as indicated by a different revision number, the batch returns the following error: `BatchStatusCodes.BATCH_NOT_CONSISTENT`.

For example, consider a batch that retrieves a device and change its Class of Service using optimistic locking:

Note This example uses the MAC address 1,6,00:11:22:33:44:55 as device ID.

```
final DeviceID deviceId = DeviceID.getInstance("1,6,00:11:22:33:44:55",
KeyType.MAC_ADDRESS);

final Batch batchForGet = connection.newBatch();
batchForGet.getDetails(deviceId, null);

final BatchStatus batchStatusForGet = batchForGet.post(10000);

if (batchStatusForGet.isError())
{
    // handle error
```

```
}

// we know that we only submitted one command in the
// batch so we can get the first command status

final CommandStatus commandStatus =
            batchStatusForGet.getCommandStatus(0);

// we know we submitted a get details command so we are
// expecting a result of a map
if (commandStatus.getDataTypeCode != CommandStatus.DATA_MAP)
{
      // throw an exception or log a message
      // we are expecting a map and didn't get one
}

final Map<String, Object> result =
    (Map<String, Object>)commandStatus.getData();

final Object consistencyValue = result.get(
           GenericObjectKeys.OID_REVISION_NUMBER);

// change the class of service
final Batch batchForMod = connection.newBatch();
batchForMod.changeClassOfService(deviceId, "gold");
// now do the optimistic locking
final List<Object> list = new ArrayList<Object>();
list.add(consistencyValue);
batchForMod.ensureConsistency(list);

// now when we post we know the device has not been changed
// since our get and our change
// if it has it  be an error
```

Submitting the Batch

The API client submits batches to the RDU synchronously or asynchronously. The API submits batches to the RDU in two modes:

- Submitting in Synchronous Mode, page 4-8
- Submitting in Asynchronous Mode, page 4-9

Submitting in Synchronous Mode

When the API client submits a synchronous batch, the batch blocks the current thread till:

- The RDU returns the results on the batch.
- The batch times out before the RDU returns results.

If the RDU Java client library does not receive a response from the RDU within the specified timeout, a `ProvTimeoutException` is thrown. The error message in the exception indicates that the RDU Java client library did not receive the batch result in the specified time but that the batch execution did not necessarily fail.

You can submit your batch to the RDU in synchronous mode with or without a timeout.

- You can submit a synchronous batch on a PACE connection object with a timeout, using:

```
// posting with timeout (in milliseconds)
final BatchStatus batchStatus = connection.postBatch(batch, 5000);
```

Note We recommend that you post a batch in synchronous mode with a timeout configured. For batches that read or update the database, you can configure a timeout of 30,000 milliseconds (msec). For batches that perform operations on live devices, you can configure a timeout of 60,000 msec.

- You can submit a synchronous batch on a PACE connection object without a timeout, using:

```
// posting with no timeout
final BatchStatus batchStatus = connection.postBatch(batch);
```

Submitting in Asynchronous Mode

When the client submits an asynchronous batch, the RDU Java client library thread that posts a batch to the RDU becomes active again. The RDU Java client library obtains the results using the batch events or, if preferred, does not obtain results at all.

You can submit an asynchronous batch on a PACE connection object, using:

```
// posting async
connection.postBatchNoStatus(batch);
```

To obtain batch results from batch events, the RDU Java client library registers a listener class that implements batch listener through the PACE connection with an appropriate qualifier. The batch listener interface exposes a completed method that has a batch event as its argument, and this method is called for each qualified batch when it completes. The batch event, in turn, provides access to the batch status object, which contains the results of the batch. To correlate between the submitted batch and the results, use the batch identifier.

To receive the results, ensure that the listener is registered before the batch is submitted. See Events to view the various events posted by Prime Cable Provisioning.

Batch Processing Modes

Depending on the commands contained in the batch, the RDU executes the batch in one of two following modes:

- Concurrent
- Nonconcurrent

The concurrent and nonconcurrent modes provide higher throughput at the RDU, without losing data integrity.

When the RDU receives a batch, the commands in the batch determine the mode in which a batch is executed. The RDU executes most batches in concurrent mode. A batch must include either concurrent or nonconcurrent commands; the RDU does not process a mix of concurrent and nonconcurrent commands in a single batch. When running one concurrent batch, you can execute other concurrent batches as well.

If the RDU has to process a batch in nonconcurrent mode, all the batches currently being run in the RDU must have completed execution, and no new batches must have started. Batches you submit at this time are queued. The RDU executes the new batches in the mode in which they are marked, after completing the processing of the nonconcurrent batch; by so doing, the RDU avoids lock conflicts and consistency issues.

Only a few commands cause a batch to run in nonconcurrent mode. These commands relate to the following system configuration operations:

- Configuring Class of Service objects in the RDU.
- Managing firmware rules, configuration templates and other files.
- Configuring device grouping objects in the RDU.
- Configuring licenses.
- Configuring users.
- Configuring system settings.
- Configuring user groups, roles, and domains.

Batch Results

A batch result is the outcome of a batch that the RDU executes. Results are returned either as exceptions or as batch status objects.

When posting a batch, an exception is thrown if:

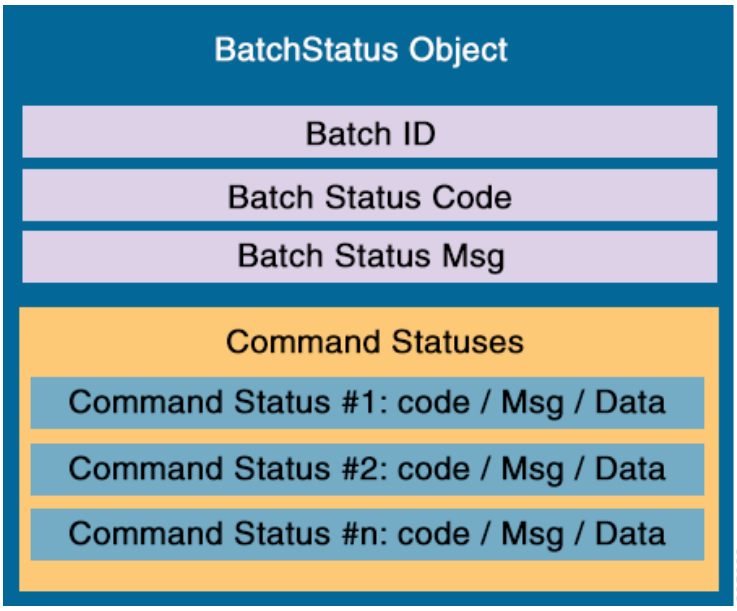
- The batch has already been posted.
- A connection to the RDU cannot be established.
- A timeout occurred when submitting a batch in synchronous mode.

Note These exceptions are rare and are raised as a `ProvisioningException` object.

If there is no `ProvisioningException` thrown, a batch status object is returned. Similar to batches and commands, there are batch status objects and command status objects. A batch status object contains command status entries for each of the commands in the corresponding batch object that was executed. The order of the command status entries matches that of the commands in the batch object.

Figure 4-2 illustrates the structure of a batch status object.

Figure 4-2 Batch Status Object



The batch status object, like a batch, serves as a container. If a single command fails, you can query the batch status to determine if there was a failure and to obtain the command status that contains the details. You can also check the batch status to determine if all the commands succeeded.

Note A batch status object does not always contain a command status. An invalid batch construction, for example, one with a combination of read and write commands, returns a batch status object without command status objects.

- You can query the batch status object to determine:
  - If a single command in a batch failed.
  - The success of all commands in the batch.
- You can query the command status object to determine the details of a command failure. For more information on the status objects, see Batch and Command Errors, page 4-16.

To check whether the batch successfully passes, and to handle errors, if any, use the following code:

```
final BatchStatus batchStatus = connection.post(batch); if (!batchStatus.isError())
{
    // batch passed so all commands passed
}
else
{
    // 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
        }
        else
        {
            // this is a batch error caused by a command
            final CommandStatus commandStatus =
                    batchStatus.getFailedCommandIndex();

            // 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
        }
    }
```

If a batch successfully passed and you want to view the results before retrieving the details of a device, use the following code.

```
final BatchStatus batchStatus = connection.post(batch); if (batchStatus.isError())
 {
     // handle error
 }
 else
 {
     // we know that we only submitted one command in the
     // batch so we can get the first command status

     final CommandStatus commandStatus =
                 batchStatus.getCommandStatus(0);

     // we know we submitted a get details command so we are
     // expecting a result of a map
     if (commandStatus.getDataTypeCode !=
         CommandStatus.DATA_MAP)
     {
         // throw an exception or log a message
         // we are expecting a map and didn't get one
     }
     else
     {
         final Map<String, Object> result =
                 (Map<String, Object>)commandStatus.getData();
         // now handle the result
     }
 }
```

Queuing a Batch

When the RDU receives a batch from a client, it queues the batch for execution. The priority of a batch determines the queue that the RDU uses for a successful execution of the batch. In case the selected queue is full, the batch is dropped, and the client is notified.

There are nine batch queues, each with the capacity to hold 1000 batches in the order that they were received. Each queue has a different priority. Each queue could contain batches that originate internally or externally. Internal batches are those designated from the DPE and the RDU, and the batches submitted to the RDU Java client library. External batches are those designated from the API client.

Of the nine batch queues:

- Three queues are meant for RDU API client batches (for example, those relating to the administrator user interface and the OSS).
- Six queues are meant for internal batches that relate to:
  - Configuration generation of CPNR DHCP extensions
  - Prime Cable Provisioning server registration
  - DPE cache synchronization
  - DPE configuration regeneration
  - Posted by RDU itself to coordinate access to the database
  - User authentication

The RDU has 100 threads dedicated to execute batches. At a time, the server can execute a maximum number of threads as defined in Table 4-1.

PACE also processes batches from the Configuration Regeneration Service (CRS) and a maximum of one CRS batch is executed for every five batches from the RDU batch queues.

Table 4-1 lists the various batch queues, with the maximum executing threads for each queue.

Table 4-1 Batch Queue

Queue	Batch Origin	Maximum Executing Threads
No Activation	External	25
Automatic Activation		50
Search command		1
Configuration Generation	Internal	25
Configuration Regeneration		25
DPE Synchronization		1
Server Registration		1
RDU internal		4
User Authentication		1

Retrying a Batch

If you are unable to receive results, you have to retry the batch posting. You not receive results if:

- A timeout occurred.
- Issues exist in batch submission.
- The client that posted the batch restarts.

Though the RDU Java client library allows you to submit batches only once, you can create a copy of the original batch and re-post it.

There are four basic groups of commands for retrying a batch. Commands that:

- Add new objects to the RDU, such as add a device or a Class of Service.
- Delete objects from the RDU, such as delete a device or a Class of Service.
- Manipulate existing objects in the RDU, such as change the Class of Service for a device, get device details, or get details on a Class of Service.

Note While batches support running commands across groups, mixing commands from different groups adversely impacts batch retrying.

Table 4-2 describes the four different command groups for retrying a batch.

Table 4-2 Command Groups for Retrying a Batch

Command Group	Description
Add new objects to the RDU	For batches that contain commands to add new objects to the RDU, retrying causes issues if the original batch succeeds. You get a command error code that the object already exists. For example, if you try to add objects that already exist, the following batch and command status codes are returned: Batch status code: `BatchStatusCodes.BATCH_FAILED_WRITE` Command status code: `CommandStatusCodes.CMD_ERROR_DEVICEID_EXISTS` **Note** Any other errors that you receive indicates a validate error that is not related to retrying the original batch.
Delete objects in the RDU	For batches that contain commands to delete objects existing in the RDU, retrying is acceptable even if the original batch succeeds. You get a command error code that the object is unknown. For example, if you try to delete an object that has already been deleted, the following batch and command status codes are returned: Batch status code: `BatchStatusCodes.BATCH_FAILED_WRITE` Command status code: `CommandStatusCodes.CMD_ERROR_DEVICEID_UNKNOWN` **Note** Any other errors that you receive indicate a validate error that is not related to retrying the original batch.
Manipulate objects in the RDU	For batches that contain commands that manipulate objects existing in the RDU, retrying does not make any difference. **Note** Any errors that you receive indicate a validate error that is not related to retrying the original batch.
Communicate with live devices	For batches that contain commands that perform operations on live devices, retrying depends on the operation. For example, if an operation adds a new object to the device, deletes an object from the device, or modifies an object from the device, retrying may cause a problem, similar to what an add device command does with the RDU.

Note When retrying a batch for which you created your own batch identifier, ensure that you use the identifier of the original batch. In case you receive a `Duplicate BatchID` error, wait until the original batch has finished execution (for example, using the batch join feature), then submit the batch, if required.

Handling Errors

Troubleshooting integration issues involve handling errors and warnings.

Integration errors may occur because of a:

- Failed RDU Java client library connection to the RDU.
- Failed batch posted in the RDU.

When the connection between the RDU Java client library and the RDU fails, the RDU Java client library tries to reconnect to the RDU. When a batch fails, all database changes are rolled back; a batch status object is returned, indicating that an error occurred.

Batch warnings indicate that the batch succeeded and the changes were committed to the database.

Types of Errors

The two types of errors that occur while integrating the OSS and BSS components to Prime Cable Provisioning are:

- Connection Errors, page 4-16.
- Batch and Command Errors, page 4-16.

Connection Errors

Connection errors are those that occur when the API client library tries to restore a broken connection with the RDU. In general, you can ignore connection errors because the RDU Java client library tries to reconnect to the RDU until the connection is restored. After a connection is restored, processing continues as usual.

You must, however, explicitly address authentication connection errors, such as an `AuthenticationException`. Prime Cable Provisioning does not automatically recover from an authentication error. As an administrator, you must confirm the authentication credentials of the user (username and password).

Batch and Command Errors

To check batch and command errors, see Step 5 in Getting Started with the Prime Cable Provisioning API.

The status objects, BatchStatus and CommandStatus, have methods to return the error code along with a detailed error message. See the API constants *BatchStatusCodes.java* and *CommandStatusCodes.java* in the API Javadocs in the installation directory of the product for the methods that return the error code along with the detailed error message.

Batch Warnings

A warning indicates that:

- The batch has succeeded and the changes have been committed.
- Something of interest has occurred.

The RDU may return warnings for successful batches in two instances:

- When the batch has altered high-level RDU objects, such as a Class of Service or a group. The devices related to these objects must have configurations regenerated from the CRS. The warning indicates the need for configuration regeneration and that this activity occur. The RDU automatically regenerates configurations for these devices.
- During the activation of a batch marked with the default no-confirmation batch flag, if an error occurs, the error appears as a warning, and the batch succeeds.
-

When you execute a batch with the custom-confirmation flag and a warning occurs during activation, the batch results contain the warning. The batch proceeds, committing the database updates. However, if an error occurs during activation, and the batch results contain the error, the batch fails, and the database updates get rolled back.