Getting Started

The Broadband Provisioning Registrar (BPR) software includes a provisioning application programming interface (API) that you can use to automate the provisioning of cable modems, digital set-top boxes (DSTBs), computers, and IP telephony devices on service provider networks. This chapter helps you get started with the development process.

What is Provisioning?

A provisioned device meets two conditions: it is registered, and it is activated.

- Registration means that a provisioning API call (for example, addDOCSISModem) added the device to BPR. Otherwise, the device is unregistered.
- Activation means that the device has booted and received a configuration from BPR. Otherwise, the device is not activated.

In the absence of either of these conditions, the device is not provisioned.

Provisioning Flows

Self-provisioning and preprovisioning are the two most common provisioning flows. Each flow passes through a set of states defined by the attributes registered/unregistered and activated/not activated.

Self-Provisioning

During self-provisioning, device activation precedes device registration. Self-provisioning passes through these states:

1. Unregistered and not activated. A subscriber buys the device.
2. Unregistered and activated. The subscriber connects the device, turns it on, and BPR gives the device a default configuration for an unregistered device.
3. Registered and activated. The subscriber uses a web user interface to register the modem.

Preprovisioning

During preprovisioning, device registration precedes device activation. Preprovisioning passes through these states:

1. Unregistered and not activated. The device arrives at the service provider’s warehouse.
2. Registered and inactivated. The service provider adds the device to BPR.
3. Activated and registered. The device boots and receives the class of service defined when the service provider added the device.

Configuration Details

The first-time configuration of a device is the result of a collaboration between the RDU, DPE, and Network Registrar extensions. Network Registrar extensions send DHCP information to the RDU. The RDU stores this information, then generates a configuration context which contains:

- Stable configuration information
- An instruction set for generating the dynamic configuration information

The RDU also validates rules to decide when the configuration is invalid. It then passes this configuration context to the DPEs. Unless subsequent configuration requests from the device are deemed invalid, they only require the Network Registrar extensions and the DPE.

The RDU

When a device appears on the network for the first time, the RDU builds the initial configuration information using data collected from several sources.

- The DHCP DISCOVER packet adds the MAC address of the device and an options field containing information that can be processed to identify the technology.
- The BPR Network Registrar extension point adds provisioning group information.
- A device detection extension point on the RDU determines the technology and if a computer or custom CPE, the modem in front of the computer.
- A device record, created when a provisioning client added the device to BPR, adds the MAC address, class of service, DHCP criteria, and properties for the device. If a record has not been added, the default Class of Service and DHCP criteria is used for the device's technology.

Having collected the data, the RDU runs an extension point for the technology and generates the configuration information for the device. The RDU then sends the configuration information to all DPEs in the device’s provisioning group.

The DPE

The DPE stores the configuration context that the RDU has sent it. It responds to configuration requests, being from the NR extensions for DHCP configuration for a device, or from the device requesting a TFTP file.

The DPE adds to the initial configuration information by performing realtime mixing of data such as:

- A time stamp from the Time of Day (TOD) server
- The location of the Trivial File Transfer Protocol (TFTP) server
- A shared secret for a DOCSIS device (from the DOCSIS technology defaults)

Network Registrar Extensions

The Network Registrar extensions request a configuration from the best available DPE. This configuration contains operations to perform on the DHCP response that is sent to the device. These operations include: setting the TFTP server as the DPE that returned the configuration, and setting the host and domain name.

Configuration Changes

Changes to configuration occur under these conditions:

- When the device appears on the network for the first time.
- Because of a provisioning API call.
- When the contents of a device's DHCP request packet change (for example, when the relay agent changes).
- When a device roams to a different provisioning group.
- When a computer or custom CPE roams to a different computer.

Provisioning Clients

The goal of the development process is the creation of one or more provisioning clients that use the provisioning API commands. The service provider's application then calls the provisioning client, which in turn, serves as the entry point to the BPR regional distribution unit (RDU).

Note For simplicity, the term provisioning client encompasses any implementation of provisioning API commands, including event driven implementations. Also for simplicity, the term service provider application refers to any application that calls a provisioning client, such as an Operations Support System (OSS), workflow, or billing system.

A single provisioning client can handle all provisioning tasks, or the tasks can be divided among separate clients. High-level provisioning client tasks include:

- Initialization—Seed BPR with the default service definitions needed when a device first appears on the network.
- Modifying Defaults—Modify the default service definitions after a subscriber has signed up for specific services.
- Device Management—Report on, add, and modify devices, or provision subscribers for additional services.
- Provisioning Flows—Enable self-provisioning of broadband devices.

What You Need and Where to Find It

Use the information in this section to help determine how best to use the resources associated with BPR.

Provisioning Clients and the Provisioning Process

This chapter contains information on writing a provisioning client and helps you understand how your provisioning client behaves during the provisioning process.

Use Cases

Chapter 2, "Provisioning API Use Cases," outlines the procedures for common provisioning use cases, such as:

- Changes in services
- Registration of devices
- Addition and deletion of devices
- Bulk provisioning of devices

Each procedure includes provisioning API code segments in pseudocode.

The Demonstration Program

The demonstration program shows how to call the provisioning API to perform various tasks, through a web-based interface.

Caution The demonstration program is intended only to give you an understanding of how to use the provisioning API to provision devices. It is not an out-of-the box solution for your provisioning needs. The installation process automatically installs the demonstration program at this location: <BPR_HOME>/rdu/DemoUI.

Sample Code Directory

You can find BPR code examples and sample technology templates at this location: <BPR_HOME>/rdu/samples.

Provisioning API and JavaDoc

The provisioning API supports several technologies; each with it's own separate interface. The JavaDoc that accompanies this product fully describes the provisioning API.

Device and Technology Support

The provisioning API supports provisioning of these devices:

- Cable modems
- Digital set-top boxes (DSTBs)
- Customer premises equipment (CPE), the computers behind modems

The provisioning API supports these technologies:

- Data-over-Cable Interface Specifications (DOCSIS) high-speed data (HSD) services for broadband cable modems and DSTBs.
- Gateway Control Protocols (xGCPs), such as the Simple Gateway Control Protocol (SGCP) for residential voice services.
- The middleware component in a DSTB.

Note Basic DHCP devices like computers are also supported as technologies.

Interfaces

The provisioning API supports these interfaces:

- Configuration—Get/set server defaults or settings, handle licensing, create custom property definitions.
- Provisioning—Query devices, generate configurations, reset, add/delete a class of service.
- DOCSIS—Manage DOCSIS devices.
- Computer—Manage computers.
- DSTB—Manage DSTB devices.
- XGCP—Manage voice service for DOCSIS modems.
- DeviceSearch—Search for multiple devices by class of service, IP address, MAC address, DHCP criteria, and FQDN.

JavaDoc

The JavaDoc packaged with BPR describes each method in the provisioning API and includes:

- Command descriptions
- Parameters, including all valid keys for property maps
- Sample code
- Events fired
- Returned command status codes
- Returned data type codes

For convenience, this guide includes a listing of the provisioning APIs. See Appendix A, “Provisioning API Reference” for more information.

Migrating Device Provisioning Registrar (DPR) Code

The CSRC DPR provisioning API is a subset of the BPR provisioning API. However, you may need to make changes in your CSRC DPR code because:

- Some methods have been deprecated, meaning that they may be discontinued in the future. See Appendix B, "Deprecated Methods" or the JavaDoc, for a complete list of deprecated methods.

  BPR deprecates the DOCSIS and computer methods from the provisioning interface, but includes new DOCSIS and computer interfaces. In all cases, migration to the new method is recommended.
- BPR defines the class of service object separately.
- BPR validates all properties. The JavaDoc describes the valid property keys that you can use with each method.
- BPR supports additional parameters that may be useful in your previous code. For example, BPR supports authentication by means of user name and password when submitting a batch.
- BPR supports the concept of DHCP criteria.

BPR API methods are signature compatible with the corresponding CSRC DPR 2.0 API methods.

Note The error conditions associated with some methods, have been changed to provide a richer set of error codes for command processing. Additionally, some ambiguous behavior in DPR is more clearly defined in BPR. Both improvements have resulted in a potentially significant code update between the two products.

Desktop Requirements

Your desktop development environment must include:

- The provisioning API
- The Java 2 Software Development Kit and the Java Runtime Environment (JRE)

Note BPR uses version 1.3.1 of the Java 2 Software Development Kit. You can find complete information on the Java 2 Software Development Kit and Java components at the Sun Microsystems website.

Additional Information

In addition, you need:

- Information on accessing the technologies that you need to provision. For example, see the DOCSIS specification for information on DOCSIS options.
- Cisco Network Registrar configuration information. Your administrator initially configures Network Registrar. The commands you initiate will use the values configured for Network Registrar. For more information, see the "DHCP Criteria" section on page 1-17.

You may also need:

- Administrator user interface information. BPR includes an administrator user interface that uses the provisioning API for initial configuration. For information on the administrator user interface, see the *Broadband Provisioning Registrar Administrator's Guide*.

- Logging information. For troubleshooting, the administrator can enable logging. Logging in BPR is dynamic, that is, the administrator or the provisioning API can change the logging settings while the server is running. Each line in the log records an operation with day, date/time, and thread information.
- Information on alerts. BPR includes an alert service for system level conditions. For example, the alert service sends a message when servers are running out of disk space or memory.

Setting Up Your Environment

This procedure outlines a general approach to setting up your environment.

1. Install the Java 2 Software Development Kit.
2. Copy the appropriate provisioning .jar file into your environment. For example:
   ```
   C:\provisioning\bpr.jar
   ```
3. Put the .jar file in the class path.
4. Include the appropriate import statements for the provisioning packages in your Java code. For example:
   ```
   import com.cisco.provisioning.cpe.*;
   ```
5. Compile your code (a check of links). For example:
   ```
   javac -cp c:\provisioning\bpr.jar myProg.java
   ```
6. Run the program. For example:
   ```
   java -cp c:\provisioning\bpr.jar myProg
   ```

Submitting Batches

A provisioning client submits requests to the RDU in the form of batches containing single or multiple commands. The batch is submitted via a connection to the RDU (PACEConnection). The RDU in turn maintains a queue of all batches, including the status of the batch. If the RDU goes down, it can restore all reliable batches in the queue.

Batch support includes:

- Support for atomic batch processing, meaning that either all commands in the batch succeed or no commands succeed. If the batch fails, the RDU rolls back all changes.
- Support for synchronous batch submission.
- Support for asynchronous batch submission with results returns in events
- Support for batch flags that control device activation.
- Support for status flags that monitor results.
- Support for reliable batches.

When you connect to the RDU and then submit a batch, the RDU:

- Receives your request and, if reliable, the batch is also journalled to allow for recovery if the server crashes.
- Puts the batch in a queue.
- Validates and processes the batch.
- Sends back the status and results.

Batch Submission Options

You can choose either synchronous or asynchronous batch execution. Asynchronous batch execution depends on the firing of an event. The service provider's application can register for events such as:

- IP changes
- Device modifications
- Configuration changes
- External file changes
- Batch completions
- Changes in system defaults
- Changes in server properties or defaults

To use events, define and register a listener and qualifier on the PACEConnection for a batch, and then follow the steps for creating batches. See Chapter 2, "Provisioning API Use Cases" for examples. Event notifications return through commands defined for each event. The service provider's application then needs to monitor these commands, and the service provider's application determines the nature of the notification it wishes to receive.

Batch Flags

By default, batches submitted to the RDU include flags that can control device disruption and the progress of configuration generation. Batches that you pass to the RDU include:

- An activation mode that determines the progress of configuration file generation and whether to attempt reset of the device. The AUTOMATIC flag is the only activation mode flag that attempts device reset.
- A confirmation mode that specifies whether BPR should check that the device has taken the configuration changes and that the device has been reset. The confirmation modes are used *only* if the activation mode is AUTOMATIC.

Note The AUTOMATIC flag calls for device reset, but devices can reset on their own when powered off and on. Assuming that a DOCSIS device is up on a network with appropriate access permissions, the default device disruption extension in BPR uses an Object ID (OID) in an SNMP MIB to reset the DOCSIS device. But, the network or the modem could be down thereby stopping BPR from automatically resetting the device.

Activation Modes

The ActivationMode class defines these flags.

- NO_ACTIVATION—The RDU updates the information for the device and the DPE caches its configuration. However, the RDU does not attempt to reset the device.
- AUTOMATIC—The RDU updates the information for the device, the DPE caches its configuration, and the RDU attempts to reset the device. You can use different confirmation modes with the AUTOMATIC activation mode.

Each batch returns a status code. See the ActivationMode class in the JavaDoc for more information.

Confirmation Modes

The ConfirmationMode class defines these flags:

- NO_CONFIRMATION—Does not confirm device reboot and returns with a BatchStatusCode value of BATCH_WARNING if disruption fails. NO_CONFIRMATION is the default.
- CUSTOM_CONFIRMATION—To take advantage of this mode, you must write your own specific device disruption extension point. See the ConfirmationMode class in the JavaDoc for complete information.

Batch Guidelines

Keep these batch guidelines in mind:

- Each batch must have a unique identifier. BPR generates a unique identifier unless you provide one.
- You can only execute a batch once.
- You cannot add commands to a batch after it has been posted to the RDU.
- A batch can contain one or more commands. If a batch contains multiple commands, these batch commands are executed in the order they are added to the batch:
  - You cannot include read (get) and write (add, change, and delete) commands in the same batch.
  - If you do not use the default Activation and Confirmation modes, you must explicitly override them.
  - If the Activation mode is set to NO_ACTIVATION, the batch can affect multiple devices.
  - If the Activation mode is AUTOMATIC, the batch can activate only one device. For example, only one modem at a time can be added using this mode.
- It is up to the provisioning client to control the rate of device provisioning.

Note If the client goes too fast, the RDU will return batch dropped errors.

- There are special cases for handling:
  - For Provisioning.generateConfiguration use NO_ACTIVATION. This command generates new device configurations, but does not attempt to reset the device. Consequently, the command does not support the AUTOMATIC mode.
  - For Provisioning.resetIPDevice use AUTOMATIC. This command does not generate new device configurations, but will reset the device.

Creating a Batch

Follow these steps to create a batch:

1. Open a PACE connection to the RDU.
2. Create a new batch.
3. Get the appropriate API interfaces.
4. Reference commands to the batch.
5. Post the batch to the RDU.
6. Get the status of the batch.
7. Get the requested query data, if it is a read batch.
8. Close the connection.

The following sections discuss each of these steps in more detail.

Step 1—Open a PACE Connection to the RDU

Before you can begin to build batches and send commands to the RDU, you must first open a connection. Open the connection by calling a getInstance() command on the PACEConnectionFactory class. Example 1-1 shows the code that calls a getInstance() command.

Example 1-1 Get a Connection Using an RDU Host Name

```
PACEConnection connection = null;
try
{
     connection =
            PACEConnectionFactory.getInstance(
           "rdu.mycompany.com",             // RDU server host name
           49187,                           // RDU port number
           "admin",                         // username
           "changeme");                     // password
}
catch (PACEConnectionException e)
catch (BPRAuthenticationException e)
{
 // connection failed -- handle exception here
}
```

Note To verify that the connection is established after a call to getInstance(), use the isAlive() call on the PACEConnection to see if the connection is alive.

Alternatively, you can use the IP address of the RDU server. Example 1-2 shows code that calls a getInstance() command using the IP address of the RDU server:

Example 1-2 Get a Connection Using an RDU Server IP Address

```
PACEConnection connection = null;
try
{
     connection =
           PACEConnectionFactory.getInstance(
          "191.14.236.129",                // RDU server IP address
          49187,                           // RDU port number
          "admin",                         // username
          "changeme");                     // password

}
catch (PACEConnectionException e)
catch (BPRAuthenticationException e)
{
 // connection failed -- handle exception here
}
```

Note BPRAuthenticationException is a run-time exception and, therefore, your compiler will not catch it.

Step 2—Get a New Batch

After you open a PACE connection to the RDU, you must get a new batch by calling a newBatch() command on the PACEConnection interface. Each batch must have a unique identifier, an Activation mode, and a Confirmation mode. See the JavaDoc for complete descriptions of the flags.

- If you do not specify a batch identifier, BPR generates a globally unique identifier.
- If you do not specify the Activation mode, the default is NO_ACTIVATION.
- If you do not specify the Confirmation mode, the default is NO_CONFIRMATION.

Example 1-3 shows typical flags on the newBatch() command:

Example 1-3 Typical Batch Flags

```
// get a batch with a BPR-generated identifier, using defaults
Batch myBatch = connection.newBatch();    // NO_ACTIVATION and NO_CONFIRMATION assumed


// get a batch with BPR-generated identifier, AUTOMATIC mode, NO_CONFIRMATION assumed
Batch myBatch2 = connection.newBatch(ActivationMode.AUTOMATIC);
```

Step 3—Reference the Appropriate API Interfaces

After you get a new batch, and before you can add commands to the batch, you must reference the command interface or interfaces that you need. For each new batch, use the API names class to specify the target interface. Example 1-4 shows typical referencing of interfaces.

Example 1-4 Referencing API Interfaces

```
Configuration configurationCommands =
(Configuration)myBatch.getProvAPI(APINames.Configuration);


Provisioning provisioningCommands =
(Provisioning)myBatch.getProvAPI(APINames.Provisioning);


DeviceSearch deviceSearchCommands =
(DeviceSearch)myBatch.getProvAPI(APINames.DeviceSearch);


DOCSIS docsisCommands =
(DOCSIS)myBatch.getProvAPI(APINames.DOCSIS);
```

Step 4—Add Commands to the Batch

Once you have referenced the command interface or interfaces you need for the current batch, you are ready to add commands to the batch. Example 1-6 shows how to queue commands that preprovision a DSTB to myBatch.

Example 1-5 Queue Commands to Preprovision a DSTB

```
docsisCommands.addDOCSISModem(
     "1,6,aa:00:bb:99:cc:55",            // macAddress for DSTB DOCSIS modem component
     null,                               // FQDN: not used in this example
     "012-34-5678",                      // ownerID
     "Gold",                             // classOfService
     "provisionedCM",                    // modem DHCP criteria
     null,                               // cpe DHCP criteria
     null);                              // properties

dstbCommands.addDSTB(
     "1,6,99:cc:22:77:dd:ff",            // macAddress for DSTB middleware component
     null,                               // FQDN: not used in this example
     "012-34-5678",                      // ownerID
     "provisionedDSTB",                  // DHCP criteria
     "1,6,aa:00:bb:99:cc:55",            // relatedDOCSISModemID: macAddress for DOCSIS
     null),                              //  DVB modem component of DSTB
     null),                              //  properties
```

Each of these commands is implicitly added to myBatch because the command accessors (in the example, provisioningCommands and dstbCommands) are implicitly tied to the batch from which they were retrieved (See the "Step 3—Reference the Appropriate API Interfaces" section on page 1-11). You cannot use these command accessors to add commands to another batch (for example, myBatch2). If you wish to perform additional commands, after myBatch has been posted, you must first create a new batch and then re-retrieve the command accessors for the new batch.

Example 1-6 adds a query command to a separate batch. The query gets the device details for a particular device identified by its MAC address:

Example 1-6 Add Command to Get Device Details

```
// queue command to myBatch2
provisioningCommands2.getDetailsForIPDevice(
"1,6,99:cc:22:77:dd:ee");   // macAddress for requested device
```

Step 5—Post the Batch to the RDU

The commands are now queued to the batch, but have not yet executed. To execute these commands, you must post the batch to the RDU. Example 1-7 shows the code that posts a batch.

Example 1-7 Post a Batch

```
BatchStatus bStatus = null;
try
       {
       bStatus = myBatch.post();
       }
catch (ProvisioningException e)
       {
       // batch posting failed -- handle exception here
       }
```

The post() command instructs the RDU to execute the commands that have been queued to the batch. The call waits while the RDU processes the batch which, depending on the number (and complexity) of batches queued at the RDU, could take several minutes to complete. The RDU either returns a BatchStatus or it generates an exception. A BatchStatus details either errors encountered with particular commands or the success status for the posted batch. An exception indicates that the RDU could not process the batch.

Note If getInstance() is unable to contact the RDU, the post method will return a PACEConnectionException or a BPRAuthenticationException to indicate if something is wrong with the connection.

Step 6—Get the Status of the Batch

If the post() command in the previous step did not throw an exception, the batch has been processed and you have the BatchStatus return value.The BatchStatus contains a status code indicating either that the batch executed successfully, or that there were errors. In addition, the BatchStatus provides access to status for the individual commands in the batch. If the batch failed, you can get additional information as to which command caused the batch to fail. And, if the batch succeeded, and the batch consisted of query commands, then the CommandStatus provides access to the requested data (see the "Step 7—Get the Requested Query Data (If Required)" section on page 1-14).

Example 1-8 examines the BatchStatus return value from the previous step:

Example 1-8 Examine the Returned BatchStatus

```
CommandStatus cStatus = null;
String errorText = null;
if (bStatus.isError())
{
     cStatus = bStatus.getFailedCommandStatus();

     if (cStatus != null && cStatus.getErrorMessage() != null)
     {
           errorText =
          + "\tcommand error message = " + cStatus.getErrorMessage();
     }
     else
     {
          errorText = ("BatchID = " + bStatus.getBatchID() + "\n"
          + "\tbatch error message = " + bStatus.getErrorMessage() )+ "\n"
     }
     // do error processing
}
     else (bstatus.isWarning())
     {
          // Process Warning
     }
```

Step 7—Get the Requested Query Data (If Required)

If the posted batch consisted of query commands, use the CommandStatus command to extract the requested data from the BatchStatus returned by Batch.post(). Example 1-9 extracts the returned data from a batch that contained a single query (see Example 1-6):

Example 1-9 Extract the Query Data from the CommandStatus

```
CommandStatus cStatus = null;
String errorText = null;
if (!bStatus.isError())
{
      // get command status for the 0-th command in the batch
      cStatus = bStatus.getCommandStatus(0);

      if (cStatus == null)
      {
            errorText = “Command status is null.”;
      }
      else if (cStatus.getDataTypeCode() == CommandStatus.DATA_MAP)
      {
            Map deviceDetails = (Map)cStatus.getData();
            String ipAddress =
                   (String)deviceDetails.get(DeviceDetailsKeys.IP_ADDRESS));
      }
```

```
}
        else (bstatus.isWarning())
        {
                // Process Warning
        }
```

Step 8—Close the Connection

Repeat Steps 2 through 7 as many times as you want for the given connection. When you are done posting batches, you should close the connection and release system resources. However, if the client is continually connecting and disconnecting from the RDU, it may be more efficient to leave the connection open. Example 1-10 shows how to close the connection.

Example 1-10 Close the Connection

```
connection.releaseConnection();
```

Key Parameters When Adding a Device

When adding a device, a provisioning client passes a number of important parameter values to the RDU.

Device and Owner Identification Information

Device identification information can include:

- MAC address
- FQDN
- Owner ID

MAC Address

The MAC address is a combination of the hardware type, hardware length, and the hardware address of a device. It is the address of the Ethernet interface of the device. BPR uses the MAC address as a key identifier of a device and ensures that the address is unique for each device. For example, if the hardware type is “1”, the hardware address length is “6”, and the hardware address is “00:01:02:nn:nn:nn”, then the MAC address of the device is 1:6:00.01.02:nn:nn:nn, where n is a digit.

MAC addresses are found in commands that:

- Add a device (for example, addDOCSISModem)
- Change the MAC address for a device (for example, changeDOCSISModemMACaddress)
- Get by MAC addresses (for example, getDetailsFor..., getIPDevicesbyMACaddress)

FQDN

The fully qualified domain Name (FQDN) is the full name of a system, that is a host name and domain name in dot notation. For example, if the host name is “foo” and the domain name is “bar.com”or “bar.com.”, then the FQDN is “foo.bar.com” or “foo.bar.com.”

BPR ensures that the FQDN is unique for each device. Your system must enable dynamic DNS (DDNS) on the Network Registrar DHCP server and DNS servers to populate DNS with device FQDNs.

FQDNs are found in commands that:

- Add a device (for example, addDOCSISModem)
- Change device FQDNs (for example, changeComputerFQDN)
- Get by FQDNs (for example, getIPDevicesByFQDN, or getDetailsFor...)

Owner Identifier

The owner identifier (ownerID) correlates all devices belonging to one owner. It can, for example, associate a device with an account number used by a billing system. The owner identifier is a string which can be anything including numbers.

The ownerID is found in commands that:

- Add a device (for example, addComputer)
- Change ownerIDs (for example, changeDOCSISModemOwnerID)
- Get by ownerIDs (for example, getIPDevicesByOwnerID or getDetailsFor...)

Class of Service

A class of service is the template for the device configuration. The service model defines classes of service, such as Gold with a list of options that can include:

- Connection speed
- Upstream an/or downstream bandwidth setting
- Ports open or blocked
- Web server hosting

Initial Configuration

Initial configuration for the classes of service takes place in the administrator application, where classes of service are defined for each technology. For example, both DOCSIS devices and computers will have a default class of service for devices that have not been previously registered with BPR by a command, such as addDOCSISModem or addComputer.

A class of service specification is found in commands that:

- Add or delete a class of service (for example, addClassOfService, deleteClassOfService)
- Get classes of service (for example, getAllClassesOfService, getClassOfServiceProperties)
- Add a device (for example, addDOCSISModem, addComputer)
- Change a devices class of service (for example, changeDOCSISModemClassOfService)
- Change a class of service (for example, changeClassOfServiceProperties)

When specifying classes of service, keep these points in mind:

- If BPR does not already have an existing definition for the class of service you pass to the method, an error is generated.

DOCSIS Class of Service Templates

BPR can store class of service configurations as templates (for example, gold.tmpl or unprov.tmpl). When templates are used, the templates can contain hard-coded specifications for each configuration option or macros that read configuration options from properties passed in by a provisioning client. To understand your current class of service definitions and their implementations, view the individual files. For information on building templates, see the "Building Class of Service Templates" section on page 1-20.

DHCP Criteria

The DHCPCriteria object bundles a set of parameters that targets a certain criteria for IP address selection and may contain additional DHCP options. A provisioning client references a DHCPCriteria object by identifier. Each device has one associated DHCPCriteria object. A DHCP criteria may be a client-class and/or selection-tag and determines which scope a device will be put in when it issues a DHCP discover.

Caution BPR and Network Registrar have configuration items for some of the same options. In particular, DHCP option properties can be configured in both systems. If there is contention between the values configured in Network Registrar and values specified in a provisioning client, the values in BPR will override the values configured in Network Registrar.

All client classes and selection tags must first be set up in Network Registrar before they can be used by a provisioning client. An incorrect association of class of service, client class, and selection tags in your client may mean that the batch cannot succeed or can result in an inappropriately provisioned device.

A Client Class Example

An administrator has created a client class called ccProvComputer in Network Registrar. A scope selection tag (tagProvComputer) has also been created and associated to the client class ccProvComputer and a scope. A provisioning client has added a device to BPR with a DHCPCriteria object that uses the client class ccProvComputer.

Note The scope selection tag tagProvComputer is not added to the DHCP criteria in this case.

The device boots and BPR uses the client class ccProvComputer to a device. Network Registrar now gives the device an IP address from a scope that has tagProvComputer assigned and that has the correct gateway address. Network Registrar also updates the DHCP reply with any policy options that have been configured on the client class ccProvComputer.

A Selection Tag Example

A scope selection tag has been created in Network Registrar and associated tagComputer with a scope. A device has been added to BPR with a DHCPCriteria object that uses tagComputer. The device boots and BPR uses this device the scope selection tag tagComputer. Network Registrar now gives this device an IP address from a scope that has the tagComputer assigned and has the correct gateway address.

Important Programming Points

When specifying DHCP criteria, keep these points in mind:

- DHCP criteria specifications may include selection tags. If the selection tags have been specified in Network Registrar, it is not necessary to specify them in BPR. If a selection tag is specified in both places, then the BPR selection tag overrides the Network Registrar selection tag.
- DHCP criteria specifications are case sensitive and must be identical with the specifications in Network Registrar. The command will fail if you specify an arbitrary string.
- If the Network Registrar configuration definitions do not include the client class or selection tags you use, the device cannot get an IP address.

Properties

Properties are used throughout the provisioning API to provide optional arguments to the commands. They are either predefined in BPR or added through the provisioning API. BPR stores properties as maps containing key/value pairs, and validates each property. See the com.cisco.provisioning.cpe.constants package in Appendix A, "Provisioning API Reference" or the JavaDoc for a listing of all keys.

Property Validation

When BPR is processing an API operation, each command accepting properties validates each predefined or custom property. Use the addCustomPropertyDefinition method to register new properties with BPR. BPR returns an error if the property is not registered or if the specification of the property does not exactly match the registered name of the property.

Order of Precedence for Properties

BPR supports two types of properties, each with an underlying order of precedence. The two types are:

- Device properties
- System properties

Note The two property types do not interact.

Table 1-1 shows the precedence hierarchy for device properties and the methods that return the device properties on each level.

Table 1-1 Device Property Precedence Hierarchy

Precedence Hierarchy	Methods that Return Properties on a Given Level
System (global for device properties)	getSystemDefaults (for example, SNMP community strings)
Technology	getDOCSISDefaults, getXGCPDefaults, getComputerDefaults
DHCP criteria	getDHCPCriteriaDetails
Class of service	getClassOfServiceProperties
Device (lowest level)	getDetailsFor...

Table 1-2 shows the precedence hierarchy for system properties and the methods that return the system properties on each level.

Table 1-2 System Property Precedence Hierarchy

Precedence Hierarchy	Methods that Return Properties on a Given Level
System (global for system properties)	getSystemDefaults (for example, SERVER_TRACE_ENABLE)
Server (lowest level)	getRDUDefaults, getDPEDefaults, getCNRDefaults

For properties that exist on more than one level, the property on the lowest level overrides the property on a higher level. For example, a property set on the IP device level overrides the same property set on the class of service level. See the JavaDoc for specific information on property levels.

All properties supported on a lower level will also be supported on the higher level. But the reverse may not be true. For example, all properties supported on a DOCSIS modem (at the technology level) are supported on the DOCSIS class of service. But all the properties supported on the DOCSIS class of service may not be supported on a DOCSIS modem.

Registration Modes

Registration modes implement the business rules by which the service provider determines the number of interactions with the subscriber. For any registered device, the service provider must be prepared to process any change to the device. So there is a significant quantitative difference between registering 100 cable modems with unregistered computers behind them and registering 100 cable modems each of which has a potentially large number of registered computers behind it.

In order to control device changes, BPR includes these registration modes:

- Standard mode—The modem and all computers behind it are registered. Standard mode has two options.
  - Roaming (the default)—A registered computer can receive its class of service behind any registered modem.
  - Fixed—A registered computer can receive its class of service only when the computer boots behind a specific modem. To invoke fixed standard mode, set the MUST_BE_BEHIND_DEVICE property in the IPDeviceKeys interface to the MAC address of the modem. If the computer reboots behind a different modem, the computer becomes unprovisioned. See the JavaDoc for complete information.

- Fixed Prov Group mode— A registered device can receive its class of service only when it is part of the specified provisioning group. To invoke the Fixed Prov Group mode, set the MUST_BE_IN_PROV_GROUP property, in the IPDeviceKeys interface, to the required provisioning group name. If the device reboots in a different provisioning group, it becomes unprovisioned. Refer to the JavaDoc for complete information.
- Promiscuous mode—Only the modem is registered, and the DHCP server maintains lease information about the computers behind the modem. All computers behind a registered modem receive network access, although the total number of computers given access at any given moment can be controlled by the maxCPEs DOCSIS parameter. To invoke promiscuous mode, enable the PROMISCUOUS_MODE_ENABLED property in the ModemKeys interface and cpeDHCPCriteria. Refer to the JavaDoc for complete information.
- Network Address Translation (NAT) mode—Only the modem is registered and no information is maintained about the computers behind the modem, but all computers receive network access. NAT mode is invoked at the hardware level.

Building Class of Service Templates

BPR includes sample templates that can define classes of service as well as other information specific to a technology implementation. You can use the sample templates, modify them, or write your own.

When working with templates, keep these important points in mind:

- The template must be registered with BPR using the addExternalFile method and the template file must end with a *.tmpl file extension.
- You must add each class of service using the addClassOfService method and the COS_DOCSIS_FILE key to specify the target template.
- If the template specifies a property, the property must be registered with BPR. See the "Properties" section on page 1-18 for more information.
- Macro substitution is possible.

Example 1-11 shows a simple template for a DOCSIS modem.

Example 1-11 Simple DOCSIS Template

```
# silver.tmpl:
#
# Silver class of service
option 3 1
option 4.1 1 [alternatively: option 4.1 ${macro1}]
option 4.2 512000 [alternatively: option 4.2 ${macro11,512000}]
option 4.3 64000
option 4.4 3
option 4.5 0
option 4.5 0
option 4.7 0

# Allow SNMP modem reset
include "modem_reset.tmpl"
# Max CPE's behind modem
option 18 2
```

Provisioning In Detail

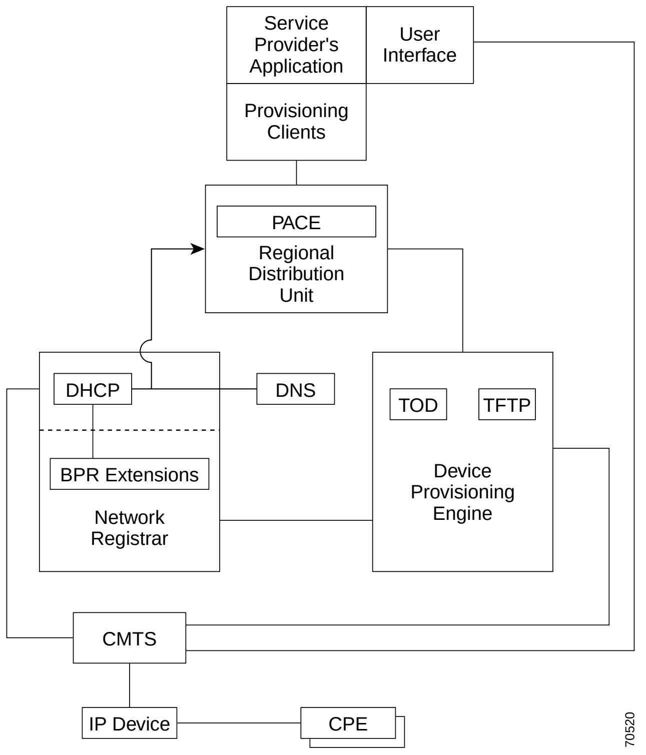
This section describes initial and subsequent activation and shows the process which a DOCSIS modem follows to receive a configuration from BPR. In this context, activation is the process of a device receiving a configuration (DHCP, TFTP, TOD) from BPR.

A Provisioning Reference Model

Figure 1-1 is a reference model for the provisioning process.

Note Some provisioning flows, such as the voice flow, may require additional components not shown here.

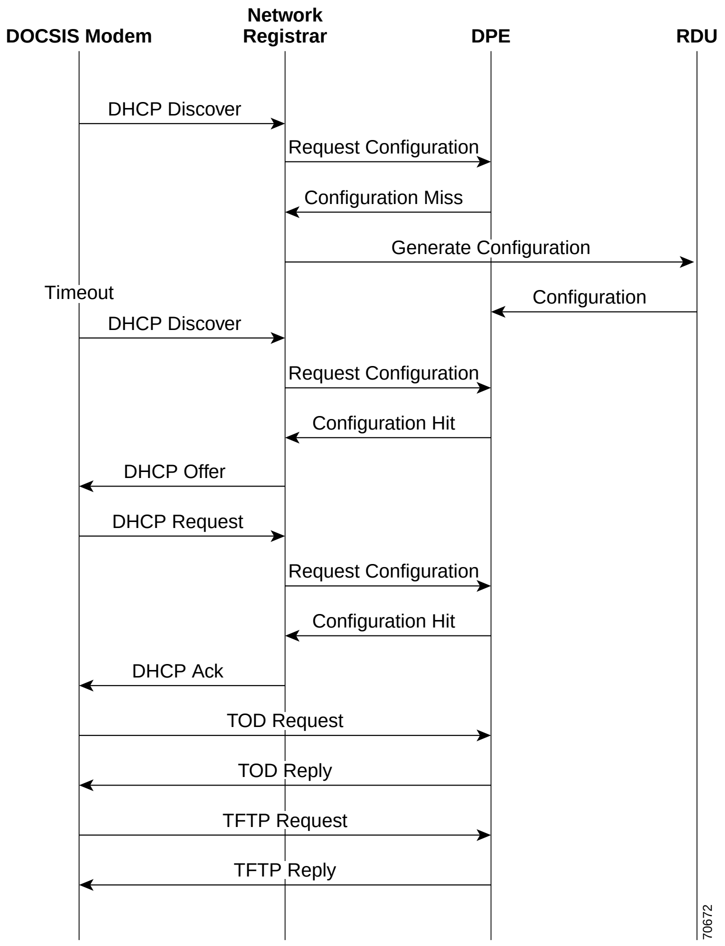
Figure 1-1 Provisioning Reference Model



First Time Activation

First time activation refers to the first time BPR sees network traffic from the device. As shown in Figure 1-2, this initial traffic is the DHCP DISCOVER. The configuration given to the device depends on its registered state. If the device has been registered, in a process known as pre-provisioning, the device will automatically receive its registered level of service. However, if the device is unregistered, it only receives the default level of service. This is useful in self-provisioning where the desired outcome is to provide just enough service to the device to allow the subscriber to complete the provisioning process.

Figure 1-2 Provisioning Flow, First Time On The Network



This is the process by which a DOCSIS modem is activated for the first time.

Step 1—Cable Modem Activity

a. The cable modem boots and sends out a DHCP DISCOVER packet. This packet includes the modem's MAC address which is used as a device identifier.

Step 2—CMTS Interaction

a. The CMTS receives the packet and forwards it to Network Registrar.

Note For simplicity, assume that the traffic involved in all of the remaining steps passes through the CMTS.

Step 3—Network Registrar Interaction

a. Network Registrar invokes a BPR extension point, which contacts a DPE (REQUEST CONFIGURATION).

Step 4—DPE Interaction

The DPE:

a. Checks its cache and determines that it does not have a configuration for the modem based on the modem's MAC address.
b. Returns a negative response (CONFIGURATION MISS) to Network Registrar.

Step 5—Network Registrar Interaction

a. Network Registrar drops the packet and requests the configuration (GENERATE CONFIGURATION) from the RDU. The information included in this requests identifies the devices provisioning group.
b. The cable modem times out and, after a few seconds, the modem sends a new DHCP DISCOVER packet.

Step 6—RDU Interaction

If the device is registered, the RDU:

a. Runs device detection to determine the technology. The information in the request is used to determine if the device has roamed between provisioning groups.
b. Creates a configuration from the information specified when the device was registered. This includes the DHCP criteria and the class of service of the device.
c. Sends the configuration to every DPE in the provisioning group for this device.

If the device is unregistered, the RDU:

a. Runs device detection to determine the technology, using the information in the request to determine what kind of configuration to create (for example, DOCSIS).
b. Creates a configuration from the information generated by device detection, the information passed in by the Network Registrar extensions, and defaults for this technology.
c. Sends the device configuration to every DPE in the provisioning group to which the device is assigned.

Step 7—DPE Interaction

a. The DPE caches the configuration information.

Step 8—Cable Modem Interaction

a. After the modem times out, it resends a DHCP DISCOVER packet. This packet includes the modem's MAC address which is used as a device identifier.

Step 9—Network Registrar Interaction

a. The Network Registrar invokes a BPR extension point, which contacts a DPE.

Step 10—DPE Interaction

The DPE:

a. Checks its cache and finds a configuration for the modem based on the modem's MAC address.
b. Passes the configuration back to the Network Registrar. The configuration includes those DHCP options that must be in the DHCP reply.

Step 11—Network Registrar Interaction

The Network Registrar:

a. Receives the configuration, and executes the commands specified in that configuration.
b. Returns a DHCP OFFER to the DOCSIS modem. This OFFER includes an IP address for the device.

Step 12—Cable Modem Interaction

a. The cable modem sends a DHCP REQUEST packet to the Network Registrar. This packet includes the modem's MAC address, which is used as a device identifier.

Step 13—Network Registrar Interaction

a. The Network Registrar invokes a BPR extension point, which contacts the DPE (REQUEST CONFIGURATION).

Step 14—DPE Interaction

The DPE:

a. Checks its cache and uses the modem's MAC address to locate the proper device configuration.
b. Passes the configuration back to Network Registrar. The configuration includes those DHCP options that must be in the DHCP reply.

Step 15—Network Registrar Interaction

The Network Registrar:

a. Receives the configuration and executes the commands specified in that configuration.
b. Returns a DHCP ACK.

Step 16—Cable Modem Interaction

a. The cable modem sends a TOD REQUEST packet to the DPE.

Step 17—DPE Interaction

a. The DPE sends a TOD REPLY packet to the modem.

Step 18—Cable Modem Interaction

a. The cable modem sends a TFTP REQUEST packet to the DPE.

Step 19—DPE Interaction

a. Mixes in any necessary information.
b. Returns TFTP response containing the device configuration.

Step 20—Cable Modem

a. Validates the configuration file.

Note If this is a DOCSIS cable modem, the cable modem and CMTS may be configured for additional validation.

b. Enters an operational state.

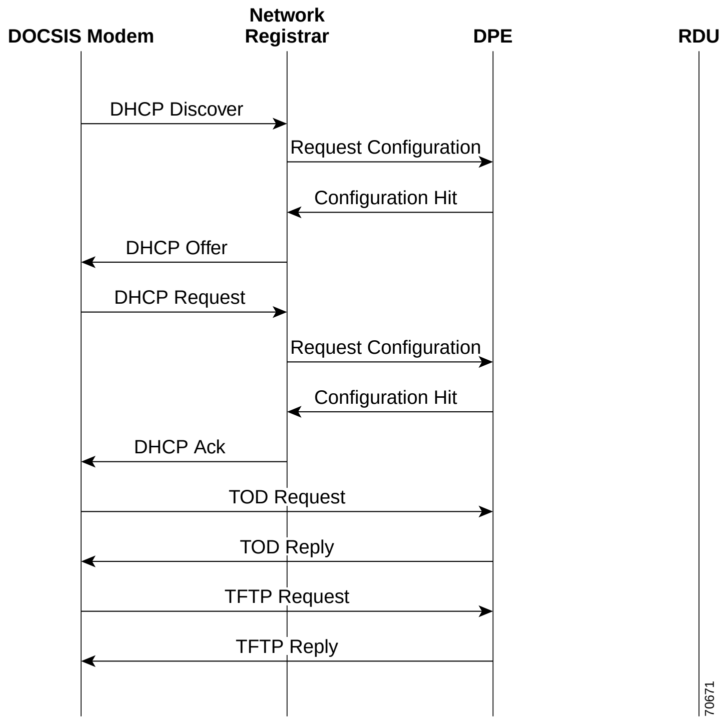
Provisioning Computers

The process by which you provision a computer is identical to steps 1 through 15 provided for a DOCSIS modem. This is because the computer does not have to request ToD or a TFTP file to become provisioned.

Subsequent Activation

After a device is initially activated, as shown in Figure 1-3, subsequent activation does not require communication with the RDU to request a configuration be generated. The RDU generates a new configuration if service selection on the device has changed.

Figure 1-3 Provisioning Flow, Subsequent Activation On The Network



These are the steps a DOCSIS modem takes on subsequent activations.

Step 1—Cable Modem

a. Boots and sends a DHCP DISCOVER packet to the Network Registrar. The packet includes those DHCP options that must be in the DHCP reply.

Step 2—CMTS Interaction

a. Receives and forwards the packet to the Network Registrar.

Note For simplicity, the remaining steps assume that all traffic passes through the CMTS.

Step 3—Network Registrar Interaction

a. Network Registrar invokes a BPR extension point, which contacts the DPE.

Step 4—DPE Interaction

The DPE:

a. Checks its cache and finds a configuration for the modem based on the modem's MAC address.

a. Passes the configuration back to the Network Registrar. The configuration includes those DHCP options that must be in the DHCP reply.

Step 5—Network Registrar Interaction

The Network Registrar:

a. Receives the configuration, and executes the commands in the configuration.
b. Returns a DHCP OFFER, which includes an IP address for the device.

Step 6—Cable Modem Interaction

a. The cable modem sends a DHCP REQUEST packet back to the Network Registrar. The packet includes the modem's MAC address which is used as the device identifier.

Step 7—Network Registrar Interaction

a. The Network Registrar invokes a BPR extension point, which contacts a DPE.

Step 8—DPE Interaction

The DPE:

a. Checks its cache and finds a configuration for the modem based on the modem's MAC address.
b. Passes the configuration back, which includes the DHCP options to be put into the DHCP reply.

Step 9—Network Registrar Interaction

The Network Registrar:

a. Receives the configuration, and executes the commands in the configuration.
b. Returns a DHCP ACK.

Step 10—Cable Modem Interaction

a. The cable modem sends a TOD REQUEST packet to the DPE.

Step 11—DPE Interaction

a. The DPE sends a TOD REPLY packet back to the modem.

Step 12—Cable Modem Interaction

a. The cable modem sends a TFTP REQUEST packet to the DPE.

Step 13—DPE Interaction

The DPE:

a. Mixes in any necessary information.
b. Returns a TFTP REPLY containing the device configuration, to the cable modem.

Step 14—Cable Modem

The cable modem:

a. Validates the configuration file.

Note If this is a DOCSIS cable modem, the cable modem and CMTS may be configured for additional validation.

---

b. Enters an operational state.