



CHAPTER 4

CPE Provisioning Overview

This chapter describes the management of customer premises equipment (CPE) using the technologies that the CPE supports for the Cisco Broadband Access Center (BAC). It features:

- [Overview, page 4-1](#)
- [Device Object Model, page 4-2](#)
- [Discovered Data, page 4-4](#)
- [Configuration Generation and Processing, page 4-6](#)
- [Device Deployment in BAC, page 4-8](#)
- [Promiscuous Access for Devices, page 4-13](#)

Overview

BAC provides provisioning and managing of residential devices, namely DOCSIS cable modems and set-top boxes, PacketCable eMTAs, CableHome devices, and computers.

BAC provisions the following device types:

- Cable modems and STBs compliant with DOCSIS 1.0, 1.1, and 2.0
- Embedded Multimedia Terminal Adapters (eMTAs) compliant with PacketCable versions 1.x
- Devices compliant with CableHome 1.0
- Computers

This release of BAC supports provisioning and managing of:

- IPv6 devices, which include:
 - Cable modems compliant with DOCSIS 3.0
 - Computers
 - Set-top boxes (STBs)
- Video STBs, specifically the RNG-200 STB, which is based on the evolving OpenCable Application Platform.
- Variants of eSAFE (embedded Service/Application Functional Entities) devices, such as mixed-IP mode PacketCable Multimedia Terminal Adapters (MTAs). A mixed-IP mode MTA is an eSAFE device that consists of an IPv6 embedded cable modem and an IPv4 eMTA. This class of devices embeds additional functionality with cable modems, such as packet-telephony, home networking, and video.

Device Object Model

The device object model in BAC is crucial in controlling the configuration that is generated for the DPE to manage devices. The process of generating a device configuration occurs at the RDU, and is controlled through named attributes and relationships.

The main objects in the device object model are:

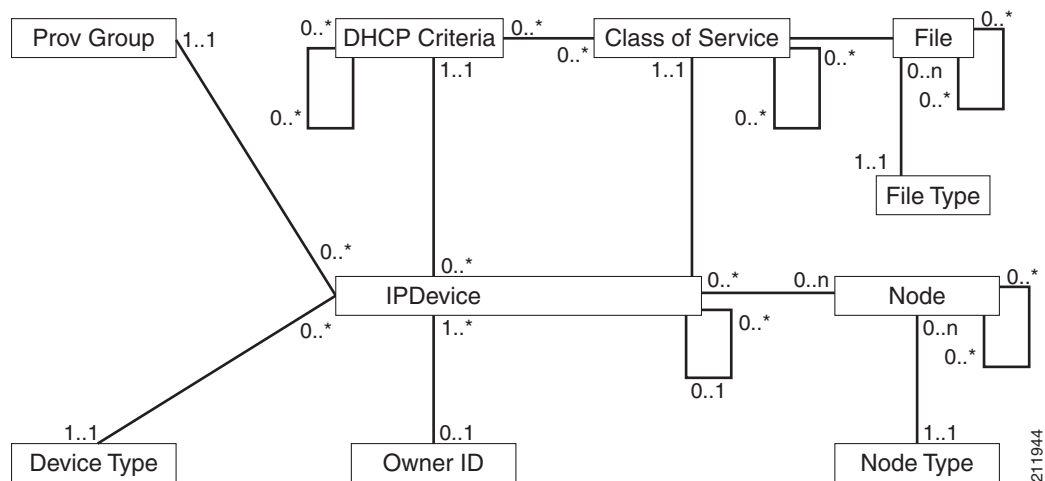
- IPDevice—Represents a network entity that requires provisioning.
- Owner ID—Represents an external identifier for a subscriber.
- Device Type—Represents the type of the device.
- ProvGroup—Represents a logical grouping of devices serviced by a specific set of DPEs.
- Class of Service—Represents the configuration profile to be assigned to a device.
- DHCP Criteria—Represents the criteria for a device to determine the selection of an IP address within the Cisco Network Registrar DHCP server.
- File—Serves as a container for files, including templates, used in provisioning.
- Node—Is a customer-specific mechanism for grouping devices.

Common among the various objects in the BAC device data model are:

- Name—For example, Gold Class of Service.
- Attributes—For example, Device ID and a fully qualified domain name (FQDN).
- Relationships—For example, the relationship of a device to a Class of Service.
- Properties—For example, a property that specifies that a device must be in a provisioning group.

Figure 4-1 illustrates the interaction among the various objects in the device data model.

Figure 4-1 Device Object Model



211944

Table 4-1 describes the attributes and relationships unique to each object in the data model.

Table 4-1 Device Object Relationships

Object	Related to...
<p>IPDevice</p> <ul style="list-style-type: none"> • Could be preprovisioned or self-provisioned (See Device Deployment in BAC, page 4-8). • Attributes include Device ID (MAC address or DUID) and FQDN 	<ul style="list-style-type: none"> • Owner ID • Provisioning Group • Class of Service • DHCP Criteria • Device Type
<p>Owner ID</p> <ul style="list-style-type: none"> • Is associated with devices and, therefore, cannot exist without a device related to it. • Enables grouping; for example, you can group all devices belonging to <i>Joe</i>. 	IPDevice
<p>Device Type</p> <ul style="list-style-type: none"> • Stores defaults common to all devices of a technology. • Enables grouping; for example, you can group all PacketCable devices. 	IPDevice
<p>File</p> <p>Stores files used in provisioning; for example, configuration files and templates.</p>	Class of Service
<p>Class of Service</p> <p>Attributes include Type, Name, and Properties. (For details, see Class of Service, page 4-3.)</p>	<ul style="list-style-type: none"> • IPDevice • File • DHCP Criteria • Configuration Template (optional)
<p>DHCP Criteria</p> <p>Enables grouping; for example, you can group devices within a specific technology to different classes of IP.</p>	<ul style="list-style-type: none"> • IPDevice • Class of Service • Configuration Template (optional)

Class of Service

Class of Service is an RDU abstraction that represents the file configuration to be handed to a device as a static file or as a template file. It enables you to group devices into configuration sets, which are service levels or different packages that are to be provided to the CPE.

The different Classes of Service are:

- **Registered**—Specified by the user when the device is registered. This Class of Service is explicitly added to the device record via the application programming interface (API).
- **Selected**—Selected and returned by an RDU extension.
- **Related**—Related to the device by being registered, selected, or both. This Class of Service is selected by the RDU extensions.

If the selected Class of Service for a device is changed, it regenerates the device configuration. If the registered Class of Service for a device is changed, it regenerates the device configuration even if it is not the selected Class of Service because it could impose a policy that would change the selected Class of Service.

Discovered Data

During the provisioning process, BAC uses a set of properties to detect the device type (whether the device is a cable modem, a computer, and so on) and generate the configuration meant for that device type and technology. The information that BAC discovers using this set of properties is known as discovered data. BAC stores discovered data for each device in the RDU database.

When a device contacts the provisioning server, it provides details about itself, such as its firmware version, MAC address, mode of operation, and so on. In the case of cable modems that contact the provisioning server, these details are made available in the:

- Discover message for IPv4 devices
- Solicit message for IPv6 devices

BAC extensions installed on Network Registrar also retrieve discovered data and send it to the RDU when requesting a configuration for a device. For these devices, the discovered data depends on the Network Registrar settings. If an attribute or an option is configured for use in Network Registrar, then the extensions fetch the value for that attribute or option from the DHCP packet and include it as part of the data discovered for BAC provisioning.

Table 4-2 lists the data that BAC discovers for IPv4 devices.

Table 4-2 Data Discovered from IPv4 Devices

Option	Description
chaddr	Specifies the hardware address of the client
client-id	Identifies a sequence of bytes or a string defined on the client that uniquely identifies the client
client-id-created-from-mac-address	Identifies the client identifier that is created from the MAC address of the client
dhcp-message-type	Specifies the type of DHCP message, such as DHCP Discover, DHCP Ack, and so on
giaddr	Specifies the IP address to which the DHCP server should reply
hlen	Specifies the length of the hardware address
htype	Specifies the hardware type
relay-agent-circuit-id	Encodes an agent local identifier of the circuit from which a DHCP client-to-server packet is received
relay-agent-info	Used in accessing the CableLabs Relay Agent CMTS Capabilities Option
relay-agent-remote-id	Encodes information about the remote host end of a circuit
v-i-vendor-opts	Identifies the options requested by the client from the server

Table 4-2 Data Discovered from IPv4 Devices (continued)

Option	Description
vendor-encapsulated-options	Defines options that are sent encapsulated in a standard DHCP option
vendor-class	Contains a string identifying capabilities of the DHCPv4 client and associated CPE

Table 4-3 lists the data that BAC discovers for IPv6 devices.

Table 4-3 Data Discovered from IPv6 Devices

Option	Description
peer-address	Specifies the IPv6 address of the client that originally sent the message or the previous relay agent that relayed the message
link-address	Specifies the non-link-local address that is assigned to an interface connected to the client subnet
client-identifier	Specifies the DHCP Unique Identifier (DUID) of the client for the lease. Because the client hardware address (chaddr) is not available for DHCPv6 clients, a DUID is used to uniquely identify a device in an IPv6 environment. This information is made available in a DHCP Solicit message.
oro	Identifies the options requested
vendor-opts	Identifies the vendor-specific information option that is used by clients and servers to exchange vendor-specific information. This information is made available in a DHCP Solicit message.
vendor-class	Identifies the vendor that manufactured the hardware on which the client is running. This information is made available in a DHCPv6 Solicit message.

You can view discovered data using the administrator user interface on the Device Details page. For more information on viewing device details, see [Viewing Device Details, page 12-9](#).

For a list of properties that BAC extensions use to discover data for DHCPv4 and DHCPv6, see [Attributes versus Options, page 6-15](#).

DUID versus MAC Address

The DHCPv4 standard uses the client identifier, or the MAC address, as the primary device identifier for DHCP clients. DHCPv6 introduces a new primary device identifier: the DHCP Unique Identifier (DUID).

DHCPv4 uses the hardware address and an optional client identifier to identify the client for assigning an address. DHCPv6 basically follows the same scheme but makes the client identifier mandatory, consolidating the hardware address and the client ID into one unique client identifier.

The client identifier in DHCPv6 consists of:

- DUID—Identifies the client system (rather than just an interface, as in DHCPv4).
- Identity Association Identifier (IAID)—Identifies the interface on that system. As described in RFC 3315, an identity association is the means used for a server and a client to identify, group, and manage a set of related IPv6 addresses.

Each DHCP client and server has a DUID. DHCP servers use DUIDs to identify clients to select configuration information and in the association of IAs with clients. DHCP clients use DUIDs to identify a server in messages where a server needs to be identified.

Configuration Generation and Processing

When a device is activated in a BAC deployment, it initiates contact with the BAC server. Once contact is established, the device's preconfigured policy, based on configuration templates associated with the device, determines the DPE's provisioning and managing of the device. Authoritative provisioning information for the device is forwarded to DPEs from the RDU as a device configuration. The DPE caches the device configuration and uses it to service requests from the device.

Device configurations can include customer-required provisioning information such as:

- DHCP IP address selection
- Bandwidth
- Data rates
- Flow control
- Communication speeds
- Level of service (also known as Class of Service)

A configuration includes an identifier (a MAC address or file name) and a revision number that is incremented each time the configuration is regenerated.

The RDU regenerates the configuration for a device when:

- Certain provisioning API calls, such as changing the device Class of Service, are made.
- Validation for a configuration fails. This occurs, for example, when certain parameters of a DHCP request from a device change from initial request parameters.

Every time the RDU regenerates a configuration for a device, the updated configuration is forwarded to the appropriate DPEs and cached.

This section also describes these related concepts:

- [Static Files versus Template Files, page 4-6](#)
- [Property Hierarchy, page 4-7](#)
- [Templates and Property Hierarchy, page 4-7](#)
- [Custom Properties, page 4-8](#)

Static Files versus Template Files

You can provision devices with BAC using two types of configuration files: static files and template files.

When using static configuration files, you enter them into the BAC system. They are then delivered via TFTP to the specific device to generate its configuration. BAC treats static configuration files like any other binary file. Static files are identified by a *.cm* extension.

Templates are text files containing DOCSIS, PacketCable, or CableHome options and values that, when used with a particular Class of Service, provide dynamic file generation. BAC ships with a configuration file utility that helps you test, validate, and view configuration and template files for DOCSIS, PacketCable, and CableHome. For detailed information on using the configuration file utility, see [Using the Configuration File Utility, page 5-21](#). Template files are identified by a *.tmpl* extension.

For a summary of static provisioning versus dynamic provisioning, see [Table 2-4](#).

Property Hierarchy

BAC properties provide a means to access and store data in BAC via the API. Preprovisioned, discovered, and status data can be retrieved via properties of corresponding objects via the API. Properties also enable configuration of BAC at the appropriate level of granularity (from system level to device group and to individual device).

Device-related properties can be defined at any acceptable point in the BAC property hierarchy. For details on whether you can assign the property at any level, refer to the API Javadoc.

The BAC property hierarchy gives you the flexibility to define properties for individual devices or groups of devices. The properties are looked up on a device and its associated objects until they are found in the following order:

1. Device registered properties—Specifies properties configured via the API or the administrator user interface.
2. Device selected properties—Specifies properties that are stored on the device record by the service-level selection process.
3. Device-detected properties—Specifies properties that are stored on the device record by the device detection process.
4. Provisioning Group—Specifies properties of a device's provisioning group.
5. Class of Service—Specifies properties that are configured on a device's Class of Service. If the service-level selection process determines a Selected Class of Service for a device, the properties from that object are used. Otherwise, the properties are looked up from the Registered Class of Service configured for a device via the API or the administrator user interface.
6. DHCP Criteria—Specifies properties that are configured on a device's DHCP Criteria. If the service-level selection process determines a Selected DHCP Criteria for a device, the properties from that object are used. Otherwise, the properties are looked up from the Registered DHCP Criteria configured for a device via the API or the administrator user interface.
7. Technology Defaults—Specifies the properties that are configured in the device's technology defaults. For example, technology defaults for DOCSIS modems, PacketCable MTAs, or computers.
8. System Defaults—Specifies the properties that are configured in system defaults.

Templates and Property Hierarchy

Generating configurations dynamically involves processing the text description of a device configuration file (which is also known as a template) into a binary device configuration. The binary configuration file is essentially a list of type-length-value (TLV) tuples, each of which contains a device configuration setting. The resulting binary configuration is then forwarded via TFTP to the device.

Dynamic configuration generation offers immense flexibility using a macro capability. Macros allow values from the BAC property hierarchy to be substituted into templates. This substitution is used for values that are commonly overridden, such as:

- Downstream or upstream bandwidth
- Number of devices behind a cable modem

In this way, BAC uses a single template to generate configurations for anywhere from a few devices to hundreds, thousands, or even millions of devices.

Custom Properties

BAC allows you to define new properties within the RDU that can then be stored on any object via the API. These properties enable substitution of values into templates.

Custom properties are variable names defined in the RDU, and must not contain any spaces.

For details on how to create custom properties, see [Configuring Custom Properties, page 13-5](#).

Device Deployment in BAC

A BAC deployment is divided into provisioning groups, with each provisioning group responsible only for a subset of the devices. All services provided by the provisioning group are implemented to provide fault tolerance (see [Provisioning Groups, page 2-14](#)).

BAC provides two device deployment options:

- Preprovisioned—The RDU is populated with configurations and rules for the various device types. When the device record is added to the RDU, it maps to a configuration specific to the device type.
- Self-provisioned—The device makes first contact with the provisioning group before the device record is added to the RDU. The preprovisioned rules, however, determine the configuration of the device.

This section describes:

- [CPE Registration Modes, page 4-8](#)
- [CPE Provisioning Flows, page 4-9](#)

CPE Registration Modes

Registration modes allow the service provider to control the number of interactions with the subscriber. For any registered device, the service provider must be prepared to process any change to the device. There is a significant 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. For this reason, the service provider must carefully choose among the standard, promiscuous, roaming, and mixed modes.

Standard Mode

When operating in the standard mode (sometimes called the fixed mode), a computer is registered and, when it is behind the correct cable modem, it receives registered access. When it is moved behind a different cable modem, however, it receives unprovisioned access.

Promiscuous Mode

When operating in the promiscuous mode, only DOCSIS modems are registered; the DHCP server maintains lease information about a device operating behind another device. All devices of specified types behind a registered device receive network access.

Roaming Mode

When operating in the roaming mode, a registered device receives its assigned service behind any other registered device. For example, this mode permits the use of a laptop moving from location to location and obtaining service from multiple cable modems.

Mixed Mode

When operating in the mixed mode, any mode is used at any time in a single deployment (with different devices).

CPE Provisioning Flows

This section describes the provisioning workflows for devices:

- [Initial Configuration Workflows, page 4-9](#)
- [Configuration Update Workflow, page 4-12](#)

Initial Configuration Workflows

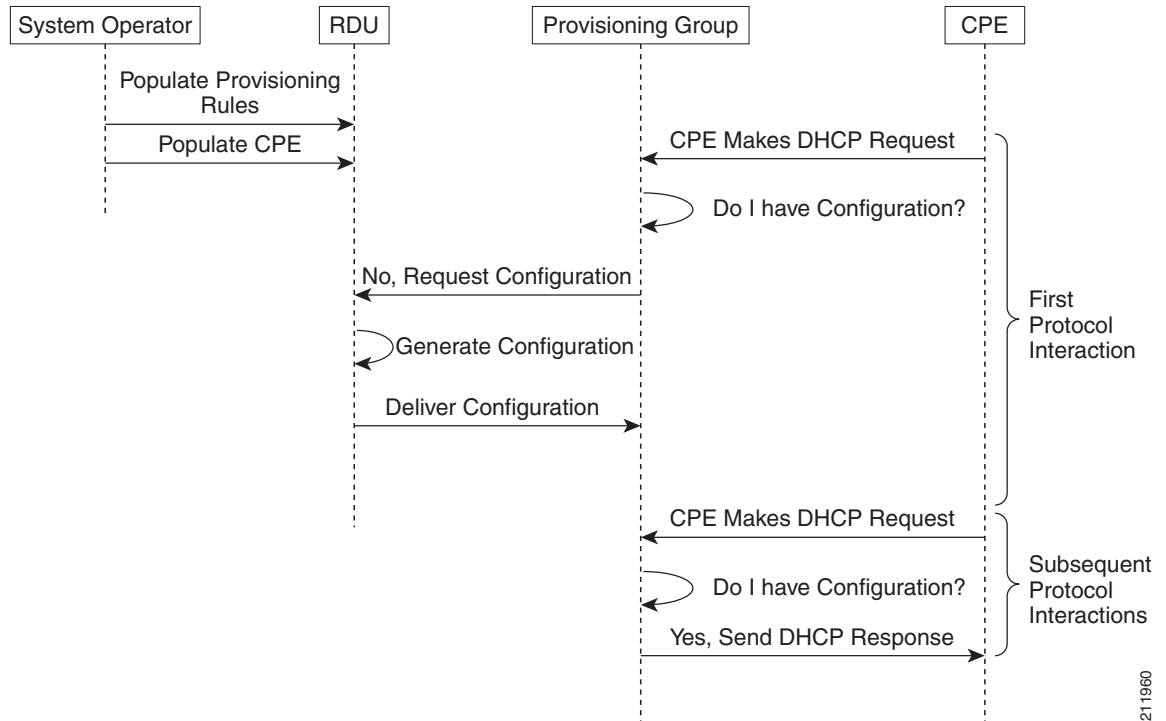
This section describes the configuration workflow when a device is initially installed and booted. The workflows differ based on deployment and registration mode and include:

- [Preprovisioned Device Workflow, page 4-10](#)
- [Self-Provisioned Device Workflow, page 4-11](#)

Preprovisioned Device Workflow

This section describes the workflow for a preprovisioned device. Figure 4-2 shows a common initial configuration workflow.

Figure 4-2 Workflow of Initial Device Configuration – Preprovisioned Mode



1. From the BAC API, the RDU is populated with specifically defined configurations and rules for various types of devices. The device is preconfigured and associated with a Class of Service, and preregistered in the RDU database.



Note Preconfiguring CPE involves populating the device information, such as the MAC address and the Class of Service, in BAC via the API. In the preprovisioned mode, this task occurs before the device has booted on the network and in the self-provisioned mode, this task occurs after the device has booted on the network.

2. When the device is booted, it discovers its provisioning group and initiates its autoprovisioning flow with DPEs in the provisioning group. The cable modem termination system (CMTS) relays broadcast traffic to the DHCP server. It is the DHCP server, or BAC extensions on the Network Registrar DHCP server, that requests configurations from the DPE.



Note When a device roams to a new provisioning group using the roaming mode, it goes through a similar flow except that its old configuration is removed from the provisioning group that it used to belong to.

211960

- The DPE, on receiving the device request, looks up its cache for a configuration for the device. Because the device has never previously contacted the provisioning group, no configuration is found. The Network Registrar extensions in the provisioning group then request the RDU to generate a configuration for the device.

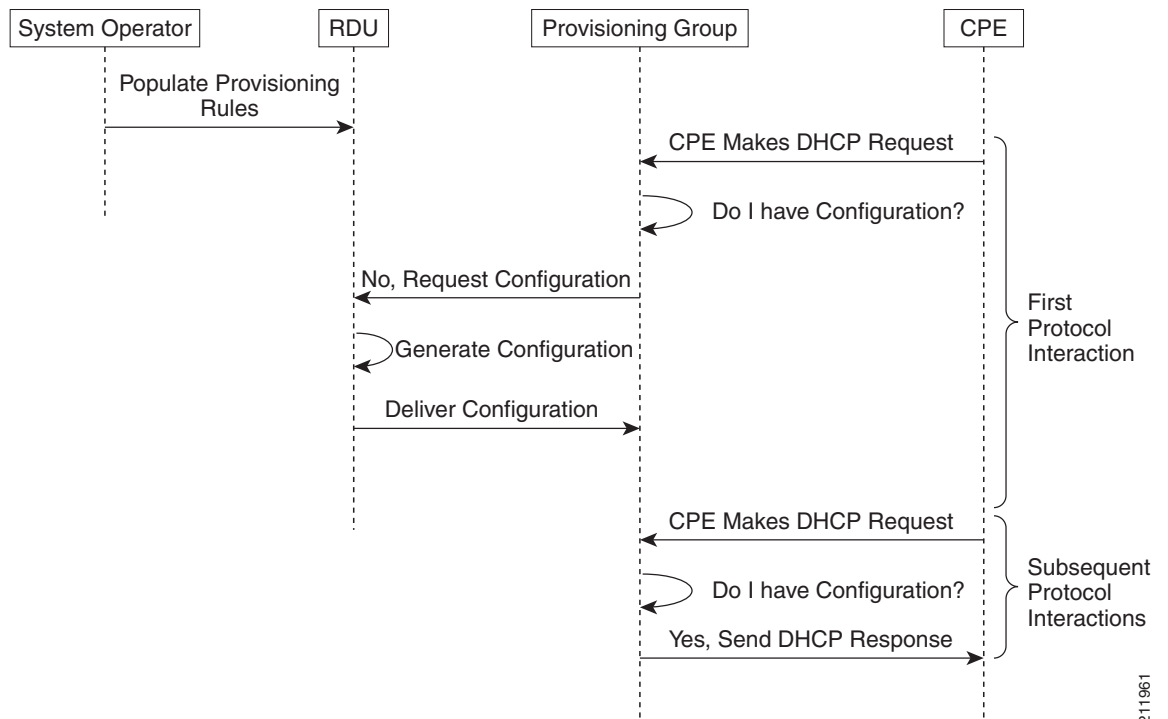
Depending on the time the RDU takes to process the request, the provisioning group may decide not to respond to the device request.

- The RDU generates a configuration appropriate for the device. The resulting device configuration directs DPE responses to various CPE protocol events, such as a DHCP Discover.
- The device configuration is forwarded to the DPE and cached there. Now, the DPE is programmed to handle subsequent CPE protocol interactions for the device autonomously from the RDU. Once the device is added to the network and a configuration is generated for the device, the device boots to allow the DPE to begin its interactions with the preregistered device.
- During interactions with the device, additional information can be discovered and forwarded to the RDU. In this case, the RDU may decide to generate new configurations and forward them to all DPEs.

Self-Provisioned Device Workflow

This section describes the workflow for a self-provisioned device. [Figure 4-3](#) shows a common initial configuration workflow.

Figure 4-3 Workflow of Initial Device Configuration – Self-Provisioned Mode



211961

1. From the BAC API, the RDU is populated with specifically defined configurations and rules for various types of devices.



Note Preconfiguring CPE involves populating the device information, such as the MAC address and the Class of Service, in BAC via the API. In the self-provisioned mode, this task occurs after the device has booted on the network.

2. When the device is booted, it discovers its provisioning group and initiates its autoprovisioning flow with DPEs in the provisioning group. The cable modem termination system (CMTS) relays broadcast traffic to the DHCP server. It is the DHCP server, or BAC extensions on the Network Registrar DHCP server, that requests configurations from the DPE.



Note When a device roams to a new provisioning group using the roaming mode, it goes through a similar flow except that its old configuration is removed from the provisioning group that it used to belong to.

3. The DPE, on receiving the device request, looks up its cache for a configuration for the device. Because the device has never previously contacted the provisioning group, no configuration is found. The Network Registrar extensions in the provisioning group then request the RDU to generate a configuration for the device.

Depending on the time the RDU takes to process the request, the provisioning group may decide not to respond to the device request.

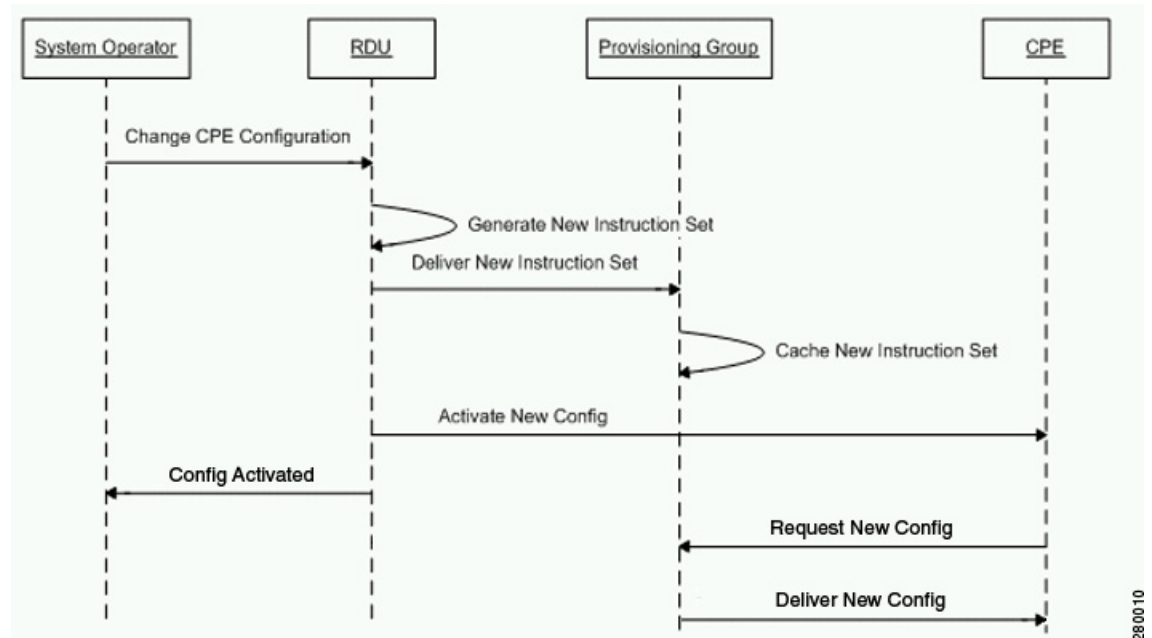
4. The RDU generates a configuration appropriate for the device. The resulting configuration directs DPE responses to various CPE protocol events, such as a DHCP Discover.
5. The device configuration is forwarded to the DPE and cached there. Now, the DPE is programmed to handle subsequent CPE protocol interactions for the device autonomously from the RDU. Once the device is added to the network and a configuration is generated for the device, the device boots to allow the DPE to begin its interactions with the preregistered device.
6. During interactions with the device, additional information can be discovered and forwarded to the RDU. In this case, the RDU may decide to generate new configurations and forward them to all DPEs.

Configuration Update Workflow

This section describes the workflows when a device configuration is updated.

Figure 4-4 shows the common configuration workflow when you change the configuration of a device that was previously configured.

Figure 4-4 Workflow of Device Configuration Update



1. From the BAC API, the device configuration at the RDU is updated.
2. The RDU generates a configuration for the device and delivers it to the DPEs in the provisioning group to which the device belongs.
3. The DPE caches the new configuration.
4. The RDU instructs the DPE to forward the new configuration to the device.
5. In the case of cable, the RDU does an SNMP Set on the modem or MTA, causing the device to reboot.

Promiscuous Access for Devices

This section describes the objects and the properties that are used to control the configuration of devices that are granted promiscuous access.

Devices are said to be given promiscuous access if they are allowed to boot and be configured without being preregistered in BAC. Promiscuous access is typically used for devices, such as computers, that appear behind a registered DOCSIS modem. If promiscuous access is not enabled for unknown devices behind a registered DOCSIS modem, the devices receive the default service level.

To grant promiscuous access to a device, you must:

- Enable or disable the promiscuous policy for unknown devices of a given type. Devices for which promiscuous access is enabled are configured according to the policy, instead of receiving the default configuration.
- Specify the Class of Service meant for unknown devices of a given type if the devices are to be given promiscuous access.

- Specify the DHCP Criteria meant for unknown devices of a given type if the devices are to be given promiscuous access.

Configuring Promiscuous Access

Table 4-4 describes the ways in which you can configure a promiscuous policy for a device.

Table 4-4 Configuring Promiscuous Access for Devices

Configuration Scope	Using API Calls...
Provisioning group of relay agent—For example, you can configure promiscuous access to allow computers only behind any registered relay agent device in a specific provisioning group.	<i>getProvGroupProperties</i> <i>changeProvGroupProperties</i>
Class of Service object of relay agent—For example, you can configure promiscuous access to allow computers only behind DOCSIS modems that are associated with a specific Class of Service.	<i>addClassOfService</i> <i>changeClassOfServiceProperties</i> <i>getClassOfServiceProperties</i>
DHCP Criteria of relay agent—For example, you can configure promiscuous access to allow computers only behind DOCSIS modems that are associated with a specific DHCP Criteria.	<i>addDHCPCriteria</i> <i>changeDHCPCriteriaProperties</i> <i>getDHCPCriteriaDetails</i>
Technology-specific defaults—For example, you can configure promiscuous access for computers behind DOCSIS modems using the technology defaults for DOCSIS modems.	<i>changeDefaults</i> <i>getDefaults</i>
System-wide defaults—Global system defaults	<i>changeSystemDefaults</i> <i>getSystemDefaults</i>



Note

You cannot configure the promiscuous policy directly on devices, such as specific modems.

Promiscuous Access and Property Hierarchy

You can configure a promiscuous policy on a number of objects in BAC. It is, therefore, important to understand the settings that take precedence. While the policy is configured using properties, the precedence of properties is determined by the BAC property hierarchy. The first object in the property hierarchy that has a specific property determines the value that BAC is to use.

BAC looks up the properties of the promiscuous policy in the property hierarchy of the device's relay agent. For example, for a computer, BAC looks up the promiscuous policy settings in the property hierarchy of the cable modem, which functions as a relay for the computer. For more details about property hierarchy, see [Property Hierarchy, page 4-7](#). For more details about promiscuous policy see [Properties for Configuring Promiscuous Policy, page 4-15](#).



Note

When you set the promiscuous policy using technology defaults, the properties must be set on objects associated with the relay agent, not the target device type. For example, to enable promiscuous access for computers behind a DOCSIS modem, you can enable promiscuous access on technology defaults for the DOCSIS modem, but not on technology defaults for computers.

The promiscuous policy properties specify the Class of Service, the DHCP Criteria, and whether promiscuous access is enabled or disabled for each device type. If promiscuous mode is enabled for a device, but a search of the device's relay agent hierarchy does not locate a match of the Class of Service or DHCP Criteria properties, the default Class of Service or DHCP Criteria for non-promiscuous access are used. For example, if BAC is configured to grant promiscuous access to computers, but it cannot locate a promiscuous Class of Service, DHCP Criteria, or both, then it uses the default Class of Service, DHCP Criteria, or both for the computer.

The Class of Service and the DHCP Criteria defaults are configured on the technology defaults of the target device (instead of its relay agent) using these properties:

- Class of Service—*/default/classOfService*
The API constant is `TechnologyDefaultsKeys.DEFAULT_CLASS_OF_SERVICE`.
- DHCP Criteria—*/default/dhcpCriteria*
The API constant is `TechnologyDefaultsKeys.DEFAULT_DHCP_CRITERIA`.

Generating Configurations for Promiscuous Devices

The configuration for promiscuous devices is generated under these conditions:

- The device first appears online and is given promiscuous access.
- An out-of-date DPE is populating its cache and requests configurations for a specific provisioning group.
- Regeneration of the configuration is explicitly requested for the device via the API call *regenConfigs*.
- Configuration of the relay agent device for a promiscuous access device is being regenerated.
- Changes to the promiscuous policy (or other configuration changes) prompt the BAC Configuration Regeneration Service (CRS) service to regenerate configurations of affected devices.

Every time a configuration for a promiscuous device is regenerated, it uses the newly configured promiscuous policy (for example, the Class of Service currently specified for promiscuous computers). However, if the Class of Service or DHCP Criteria of a device is changed via the API after the device appears online as a promiscuous device, then from then on, the device is not considered promiscuous and is unaffected by any changes that you make to the promiscuous policy. The device is henceforth considered registered.

Properties for Configuring Promiscuous Policy

To configure promiscuous access for devices, you must configure the properties associated with specific device types that BAC supports. You can enable or disable promiscuous access for the device types.

- Enabled—Enables promiscuous access for devices within the scope associated with the API call that [Table 4-4](#) describes.
- Disabled—Disables promiscuous access. If the property does not exist, the default is the disabled setting.

See [Table 4-5](#) for a list of properties on which you configure promiscuous access.

Promiscuous policy properties are divided into read-write and read-only properties. This section describes the read-write and read-only properties that you must configure to enable promiscuous access for devices and those that you set to select Class of Service or DHCP Criteria for these devices.

Read-Write Properties



Note Table 4-4 describes the applicable API calls for all the properties that are described in this section.

Table 4-5 describes the properties that you can use to enable promiscuous access.

Table 4-5 Properties for Enabling Promiscuous Access

Property Name	Description
<i>/promiscuousMode/enable/Computer</i>	<p>Sets a Boolean value of “true” or “false” in the relay agent’s property hierarchy:</p> <ul style="list-style-type: none"> • true—Enables promiscuous access for computers behind such a relay. • false—Disables promiscuous access for computers behind such a relay. <p>If the property does not exist in the relay agent's property hierarchy, promiscuous access for computers behind such a relay is disallowed and the devices receive default access.</p> <p>API Constant</p> <p><code>PolicyKeys.COMPUTER_PROMISCUOUS_MODE_ENABLED</code></p>
<i>/promiscuousMode/enable/PacketCableMTA</i>	<p>Sets a Boolean value of “true” or “false” in the relay agent’s property hierarchy:</p> <ul style="list-style-type: none"> • true—Enables promiscuous access for PacketCable MTAs behind such a relay. • false—Disables promiscuous access for PacketCable MTAs behind such a relay. <p>If the property does not exist in the relay agent's property hierarchy, promiscuous access for PacketCable MTAs behind such a relay is disallowed and the devices receive default access.</p> <p>API Constant</p> <p><code>PolicyKeys.PACKET_CABLE_MTA_PROMISCUOUS_MODE_ENABLED</code></p>
<i>/promiscuousMode/enable/STB</i>	<p>Sets a Boolean value of “true” or “false” in the relay agent’s property hierarchy:</p> <ul style="list-style-type: none"> • true—Enables promiscuous access for STBs behind such a relay. • false—Disables promiscuous access for STBs behind such a relay. <p>If the property does not exist in the relay agent's property hierarchy, promiscuous access for STBs behind such a relay is disallowed and the devices receive default access.</p> <p>API Constant</p> <p><code>PolicyKeys.STB_PROMISCUOUS_MODE_ENABLED</code></p>

Table 4-5 Properties for Enabling Promiscuous Access (continued)

Property Name	Description
<p><i>/promiscuousMode/enable/ CableHomeWanData</i></p>	<p>Sets a Boolean value of “true” or “false” in the relay agent’s property hierarchy:</p> <ul style="list-style-type: none"> • true—Enables promiscuous access for CableHome WAN-Data devices behind such a relay. • false—Disables promiscuous access for CableHome WAN-Data devices behind such a relay. <p>If the property does not exist in the relay agent’s property hierarchy, promiscuous access for WAN-Data devices behind such a relay is disallowed and the devices receive default access.</p> <p>API Constant</p> <p><code>PolicyKeys.CABLE_HOME_WAN_DATA_PROMISCUOUS_MODE_ENABLED</code></p>
<p><i>/promiscuousMode/enable/ CableHomeWanMan</i></p>	<p>Sets a Boolean value of “true” or “false” in the relay agent’s property hierarchy:</p> <ul style="list-style-type: none"> • true—Enables promiscuous access for CableHome WAN-MAN devices behind such a relay. • false—Disables promiscuous access for CableHome WAN-MAN devices behind such a relay. <p>If the property does not exist in the relay agent’s property hierarchy, promiscuous access for WAN-MAN devices behind such a relay is disallowed and the devices receive default access.</p> <p>API Constant</p> <p><code>PolicyKeys.CABLE_HOME_WAN_MAN_PROMISCUOUS_MODE_ENABLED</code></p>
<p><i>/promiscuousMode/enable/</i></p>	<p>Use this property to enable or disable promiscuous access for new types of devices by appending the property name with the name of a valid device type.</p> <p>Sets a Boolean value of “true” or “false.”</p> <p>API Constant</p> <p><code>PolicyKeys.PROMISCUOUS_MODE_PREFIX</code></p>

Table 4-6 describes the read-write properties that you must configure to select Class of Service for devices granted promiscuous access.

Table 4-6 Promiscuous Access—Read-Write Properties for Class of Service

Class of Service Property Name	Description
<i>/provisioning/cpeClassOfService/Computer</i>	Specifies the name of an existing Class of Service that will be selected for promiscuous computers API Constant PolicyKeys.COMPUTER_CLASS_OF_SERVICE
<i>/provisioning/cpeClassOfService/PacketCableMTA</i>	Specifies the name of an existing Class of Service that will be selected for promiscuous PacketCable MTAs API Constant PolicyKeys.PACKET_CABLE_MTA_CLASS_OF_SERVICE
<i>/provisioning/cpeClassOfService/STB</i>	Specifies the name of an existing Class of Service that will be selected for promiscuous set-top boxes API Constant PolicyKeys.STB_CLASS_OF_SERVICE
<i>/provisioning/cpeClassOfService/CableHomeWanMan</i>	Specifies the name of an existing Class of Service that will be selected for promiscuous CableHome WAN-Data devices API Constant PolicyKeys.CABLEHOME_WAN_DATA_CLASS_OF_SERVICE
<i>/provisioning/cpeClassOfService/CableHomeWanData</i>	Specifies the name of an existing Class of Service that will be selected for promiscuous CableHome WAN-MAN devices API Constant PolicyKeys.CABLEHOME_WAN_MAN_CLASS_OF_SERVICE
<i>/provisioning/cpeClassOfService/</i>	Specifies an existing Class of Service that will be selected for devices of the specified device type. Use this property name with a valid device type name. You can use this property for custom device types. API Constant PolicyKeys.PROMISCUOUS_COS_PREFIX

Table 4-7 describes the read-write properties that you must configure to select DHCP Criteria for devices granted promiscuous access.

Table 4-7 Promiscuous Access—Read-Write Properties for DHCP Criteria

DHCP Criteria Property Name	Description
<i>/provisioning/cpeDhcpCriteria/Computer</i>	Specifies the name of an existing DHCP Criteria object that will be selected for promiscuous computers API Constant PolicyKeys.COMPUTER_DHCP_CRITERIA

Table 4-7 Promiscuous Access–Read-Write Properties for DHCP Criteria (continued)

DHCP Criteria Property Name	Description
<i>/provisioning/cpeDhcpCriteria/ PacketCableMTA</i>	Specifies the name of an existing DHCP Criteria object that will be selected for promiscuous PacketCable MTAs API Constant PolicyKeys.PACKET_CABLE_MTA_DHCP_CRITERIA
<i>/provisioning/cpeDhcpCriteria/STB</i>	Specifies the name of an existing DHCP Criteria object that will be selected for promiscuous set-top boxes API Constant PolicyKeys.STB_ DHCP_CRITERIA
<i>/provisioning/cpeDhcpCriteria/ CableHomeWanData</i>	Specifies the name of an existing DHCP Criteria object that will be selected for promiscuous CableHome WAN-Data devices API Constant PolicyKeys.CABLEHOME_WAN_DATA_ DHCP_CRITERIA
<i>/provisioning/cpeDhcpCriteria/ CableHomeWanMan</i>	Specifies the name of an existing DHCP Criteria object that will be selected for promiscuous CableHome WAN-MAN devices API Constant PolicyKeys.CABLEHOME_WAN_MAN_ DHCP_CRITERIA
<i>/provisioning/cpeDhcpCriteria/</i>	Specifies an existing DHCP Criteria object that will be selected for devices of the specified device type. Use this property name with a valid device type name. You can use this property for custom device types. API Constant PolicyKeys.PROMISCUOUS_DC_PREFIX

Read-Only Properties

Table 4-8 covers read-only promiscuous properties that you must configure to select the Class of Service and the DHCP Criteria for devices. Together with the read-write properties specified in the previous section, these read-only properties help determine the current system configuration.

Table 4-8 Promiscuous Access–Read-Only Properties

Property Name	Description						
<i>/isSystemWide/default/promiscuous</i>	Returns a “true” value if a given Class Of Service or DHCP Criteria object is referenced as system-wide default for promiscuous devices.						
	<table border="1"> <thead> <tr> <th>Applicable API Calls</th> <th>API Constant</th> </tr> </thead> <tbody> <tr> <td><i>getClassOfServiceProperties</i></td> <td>PolicyKeys.IS_SYSTEM_WIDE_</td> </tr> <tr> <td><i>getDHCPCriteriaDetails</i></td> <td>DEFAULT_PROMISCUOUS</td> </tr> </tbody> </table>	Applicable API Calls	API Constant	<i>getClassOfServiceProperties</i>	PolicyKeys.IS_SYSTEM_WIDE_	<i>getDHCPCriteriaDetails</i>	DEFAULT_PROMISCUOUS
Applicable API Calls	API Constant						
<i>getClassOfServiceProperties</i>	PolicyKeys.IS_SYSTEM_WIDE_						
<i>getDHCPCriteriaDetails</i>	DEFAULT_PROMISCUOUS						

Table 4-8 Promiscuous Access—Read-Only Properties (continued)

Property Name	Description	
<i>/referencedBy/deviceTypes/ forPromiscuousDevices</i>	Returns a list of Device Type object (technology) names that reference a given Class of Service or DHCP Criteria object in promiscuous policy properties	
	Applicable API Calls <i>getClassOfServiceProperties</i> <i>getDHCPCriteriaDetails</i>	API Constant PolicyKeys.REFERENCED_BY_DEVICE_TYPE_FOR_PROMISCUOUS_DEVICES
<i>/related/classesOfService</i>	Returns a list of Class of Service object names that are used by a given Class of Service or DHCP Criteria object in promiscuous policy properties	
	Note You can use this property as a shortcut to obtain the Class of Service list. You can also obtain this list by reading individual promiscuous policy properties set on this object.	
	Applicable API Calls <i>getClassOfServiceProperties</i> <i>getDHCPCriteriaDetails</i>	API Constant PolicyKeys.RELATED_CLASS_OF_SERVICE
	Returns a list of DHCP Criteria object names that are used by a given Class of Service or DHCP Criteria object in promiscuous policy properties	
<i>/related/dhcpCriteria</i>	Returns a list of DHCP Criteria object names that are used by a given Class of Service or DHCP Criteria object in promiscuous policy properties	
	Note You can use this property as a shortcut to obtain the DHCP Criteria list. You can also obtain this list by reading individual promiscuous policy properties set on this object.	
	Applicable API Calls <i>getClassOfServiceProperties</i> <i>getDHCPCriteriaDetails</i>	API Constant PolicyKeys.RELATED_DHCP_CRITERIA

Custom Policy for Promiscuous Devices

You can configure promiscuous policy for a device using the properties specified in the above section. When additional logic is required, however, you can implement custom logic using extensions and custom properties. Custom properties allow for the definition of new properties, which can then be stored on any object via the API.

To augment the promiscuous device policy, you can use these extensions:

- **Device Detection**—Determines the technology type of the device (usually based on DHCP request data). Information that this extension detects is placed in a Device Detection Context that other extensions then use.
- **Service-Level Selection**—Selects the appropriate Class of Service and DHCP Criteria objects for a device. The promiscuous policy properties determine the Class of Service and DHCP Criteria for devices with promiscuous access.

- **Configuration Generation**—Generates the configuration for a device and, if necessary, for the devices behind it. Configurations are regenerated for promiscuous devices behind the relay agent based on the policy that the service-level selection extension selects. You may need to change the extension only if you want to augment the default behavior of regenerating configuration for devices behind a relay agent.

