



APPENDIX **C**

API Use Cases

This appendix presents a series of the most common provisioning application programming interface (API) use cases, including pseudo-code segments that can be used to model typical service provider workflows. The pseudo code used in the use cases resembles java though it is not intended for direct compilation. Please refer to the Broadband Access Center (BAC) 2.7.1 API Javadoc for more details and sample Java code segments explaining individual API calls and features.

These use cases are directly related to device (and/or service) provisioning. Many administrative operations, such as managing Class of Service, DHCP criteria, and licenses are not addressed here. It is highly recommended to go through the API javadoc for more details on the related API calls. You can also use the administrator user interface to perform most of these activities.

Use Cases

This appendix includes these use cases:

- [Self-Provisioned Modem and Computer in Fixed Standard Mode, page C-2](#)
- [Adding a New Computer in Fixed Standard Mode, page C-5](#)
- [Disabling a Subscriber, page C-7](#)
- [Preprovisioning Modems/Self-Provisioned Computers, page C-9](#)
- [Modifying an Existing Modem, page C-11](#)
- [Unregistering and Deleting a Subscriber's Devices, page C-12](#)
- [Self-Provisioning First-Time Activation in Promiscuous Mode, page C-14](#)
- [Bulk Provisioning 100 Modems in Promiscuous Mode, page C-17](#)
- [Preprovisioning First-Time Activation in Promiscuous Mode, page C-19](#)
- [Replacing an Existing Modem, page C-20](#)
- [Adding a Second Computer in Promiscuous Mode, page C-21](#)
- [Self-Provisioning First-Time Activation with NAT, page C-21](#)
- [Adding a New Computer Behind a Modem with NAT, page C-23](#)
- [Move Device to Another DHCP Scope, page C-24](#)
- [Log Device Deletions Using Events, page C-25](#)
- [Monitoring an RDU Connection Using Events, page C-26](#)
- [Logging Batch Completions Using Events, page C-27](#)

- [Getting Detailed Device Information, page C-27](#)
- [Searching Using the Default Class of Service, page C-28](#)
- [Retrieving Devices Matching a Vendor Prefix, page C-30](#)
- [Preprovisioning PacketCable eMTA, page C-32](#)
- [SNMP Cloning on PacketCable eMTA, page C-34](#)
- [Incremental Provisioning of PacketCable eMTA, page C-35](#)
- [Preprovisioning DOCSIS Modems with Dynamic Configuration Files, page C-37](#)
- [Optimistic Locking, page C-39](#)
- [Temporarily Throttling a Subscriber's Bandwidth, page C-40](#)
- [Preprovisioning CableHome WAN-MAN, page C-41](#)
- [CableHome with Firewall Configuration, page C-43](#)
- [Retrieving Device Capabilities for CableHome WAN-MAN, page C-45](#)
- [Self-Provisioning CableHome WAN-MAN, page C-47](#)
- [Lease Reservation Use Cases, page C-49](#)

Self-Provisioned Modem and Computer in Fixed Standard Mode

The subscriber has a computer installed in a single dwelling unit and has purchased a DOCSIS cable modem. The computer has a web browser installed.

Desired Outcome

Use this workflow to bring a new unprovisioned DOCSIS cable modem and computer online with the appropriate level of service.

-
- Step 1** The subscriber purchases and installs a DOCSIS cable modem at home and connects a computer to it.
 - Step 2** The subscriber powers on the modem and the computer, and BAC gives the modem restricted access. The computer and modem are assigned IP addresses from restricted access pools.
 - Step 3** The subscriber starts a web browser, and a spoofing DNS server points the browser to a service provider's registration server (for example, an OSS user interface or a mediator).
 - Step 4** The subscriber uses the service provider's user interface to complete the steps required for registration, including selecting Class of Service.
 - Step 5** The service provider's user interface passes the subscriber's information, such as the selected Class of Service and computer IP address, to BAC, which then registers the subscriber's modem and computer.

```
// First we query the computer's information to find the modem's
// MAC address. We use the computers IP address (the web browser
// received this when the subscriber opened the service providers
// web interface)

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION is the activation mode because this is a query
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the device

LeaseResults computerLease =
    getAllForIPAddress("10.0.14.38");
    // ipAddress: restricted access.

// Derive the modem MAC address from the computer's network
// information. The 1,6, is a standard prefix for an Ethernet
// device. The fully qualified MAC address is required by BACC

String modemMACAddress = "1,6," +
    computerLease.getSingleLease().get(RELAY_AGENT_REMOTE_ID);

// Now let's provision the modem and the computer in the same
// batch. This can be done because the activation mode of this
// batch is NO_ACTIVATION. More than one device can be operated
// on in a batch if the activation mode does not lead to more
// than one device being reset. NO_ACTIVATION will generate a
// configuration for the devices. However it will not attempt
// to reset the devices. The configuration can be generated
// because the devices have booted. NO_CONFIRMATION is the
// confirmation mode because we are not attempting to reset
// the modem. We do not want to reset the modem here because
// we want to notify the user to reset their computer. If we
// reset the modem in this batch, we will not be able to notify
// the user of anything until the modem has come back online.
// To add a DOCSIS modem:

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

add(
    DeviceType: DOCSIS,
        // deviceType: DOCSIS
    modemMACAddress,
        // macAddress: derived from computer lease
    null,
        // hostName: not used in this example
    null,
        // domainName: not used in this example
    "0123-45-6789",
        // ownerID: here, account number from billing system
    "Silver",
        // ClassOfService
    "provisionedCM",
        // DHCP Criteria: Network Registrar uses this to
        // select a modem lease granting provisioned IP address
```

```

        null
        // properties: not used
    );

    // properties: not used
String computerMACAddress = computerLease.
    getSingleLease().get(DeviceDetailsKeys.MAC_ADDRESS);
// Create a Map for the computer's properties

Map properties;

// Setting the property IPDeviceKeys.MUST_BE_BEHIND_DEVICE
// on the computer ensures that when the computer boots it
// will only receive its provisioned access when it is behind
// the given device. If it is not behind the given device,
// it will receive default access (unprovisioned). This makes
// the computer "fixed" behind the specified modem.

properties.put(IPDeviceKeys.MUST_BE_BEHIND_DEVICE,
    modemMACAddress);

add(
    DeviceType.COMPUTER,
        // deviceType: COMPUTER
    computerMACAddress,
        // macAddress: derived from computer lease
    null,
        // hostName: not used in this example
    null,
        // domainName: not used in this example
    "0123-45-6789",
        // ownerID: here, account number from billing system
    null,
        // classOfService : get the default COS
    "provisionedCPE",
        // DHCP Criteria: Network Registrar uses this to
        // select a modem lease granting provisioned IP address
    properties
        // properties:
    );

```

Step 6 The user interface prompts the subscriber to reboot the computer.

Step 7 The provisioning client calls `performOperation(DeviceOperation.RESET, modemMACAddress, null)` to reboot the modem and gives the modem provisioned access.

```
get-new-batch(AUTOMATIC, NO_CONFIRMATION);

// AUTOMATIC is the activation mode because we are attempting
// to reset the modem so that it receives its new class of service.
// NO_CONFIRMATION is the confirmation mode because we don't
// want the batch to fail if we can't reset the modem. The user
// may have power cycled the modem when they rebooted their computer.
// Send a batch to reset the modem now that the user has been
// notified to reboot their computer.

performOperation(
    DeviceOperation.RESET,
        //deviceOperation: Reset operation
    modemMACaddress,
        // macAddress:Modem's MAC address
    null
        // properties: not used
);
```

- Step 8** After rebooting, the computer receives a new IP address, and both cable modem and computer are now provisioned devices. The computer has access to the Internet through the service provider's network.
-

Adding a New Computer in Fixed Standard Mode

A multiple system operator (MSO) lets a subscriber have two computers behind a cable modem. The subscriber has one computer already registered and then brings home a laptop from work and wants access. The subscriber installs a hub and connects the laptop to it.

Desired Outcome

Use this workflow to bring a new unprovisioned computer online with a previously provisioned cable modem so that the new computer has the appropriate level of service.

- Step 1** The subscriber powers on the new computer and BAC gives it restricted access.
- Step 2** The subscriber starts a web browser on the new computer and a spoofing DNS server points it to the service provider's registration server (for example, an OSS user interface or a mediator).
- Step 3** The subscriber uses the service provider's user interface to complete the steps required to add a new computer.
- Step 4** The service provider's user interface passes the subscriber's information, such as the selected Class of Service and computer IP address, to BAC, which then registers the subscriber's modem and computer.

```

// First we query the computer's lease information to its
// MAC address. We use the computers IP address (the web browser
// received this when the subscriber opened the service providers
// web interface)

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

/ NO_ACTIVATION is the activation mode because this is a query
/ NO_CONFIRMATION is the confirmation mode because we are not
/ attempting to reset the device.

LeaseResults computerLease =
    getAllForIPAddress("10.0.14.39");
    // ipAddress: restricted access.

String computerMACAddress = computerLease.
    getSingleLease().get(DeviceDetailsKeys.MAC_ADDRESS);

// We have the MAC address now. Let's add the computer to BACC.

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION will generate a configuration for the computer.
// However it will not attempt to reset the computer (this
// can not be done). The configuration can be generated because
// the device has booted. NO_CONFIRMATION is the confirmation
// mode because we are not attempting to reset the modem.

Map properties; // Map containing the properties for the computer

// Setting the property IPDeviceKeys.MUST_BE_BEHIND_DEVICE on
// the computer ensures that when the computer boots, it will
// only receive its provisioned access when it is behind the
// given device. If it is not behind the given device, it
// will receive default access (unprovisioned) and hence the
// fixed mode.

properties.put(IPDeviceKeys.MUST_BE_BEHIND_DEVICE, modemMACAddress);

// The IPDeviceKeys.MUST_BE_IN_PROV_GROUP property ensures that
// the computer will receive its provisioned access only when
// it's brought up in the specified provisioning group. This prevents
// the computer (and/or the modem) from moving from one locality to
// to another locality.

properties.put(IPDeviceKeys.MUST_BE_IN_PROV_GROUP, "bostonProvGroup");

add(
    DeviceType.COMPUTER,
        // deviceType: COMPUTER
    computerMACAddress,
        // macAddress: derived from computer lease
    null,
        // hostName: not used in this example
    null,
        // domainName: not used in this example
    "0123-45-6789",

```

```
        // ownerID: here, account number from billing system
    null,
        // classOfService: get the default COS
    "provisionedCPE",
        // DHCP Criteria: Network Registrar uses this to
        // select a modem lease granting provisioned IP address
    properties
        // properties:
    );
```

- Step 5** The user interface prompts the subscriber to reboot the new computer so that BAC can give the computer its registered service level.
- Step 6** The computer is now a provisioned device with access to the appropriate level of service.
-

Disabling a Subscriber

A service provider needs to disable a subscriber from accessing the Internet due to recurring nonpayment.

Desired Outcome

Use this workflow to disable an operational cable modem and computer, so that the devices temporarily restrict Internet access for the user. Additionally, this use case can redirect the user's browser to a special page that could announce:

```
You haven't paid your bill so your Internet access has been disabled.
```

- Step 1** The service provider's application uses a provisioning client program to request a list of all of the subscriber's devices from BAC. The service provider's application then uses a provisioning client to individually disable or restrict each of the subscriber's devices.

```
// The service provider's application uses a provisioning client
// to get a list of all of the subscriber's devices.

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION is the activation mode because this is a query.
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the devices.

getAllForOwnerID(
    "0123-45-6789"); // Query all the devices for this account number
// For each device in the retrieved list:
// We are only interested in the modems. If we disable network
// access of the modems, we disable network access for all the
// subscriber's devices. If we are using roaming standard mode,
// we may also change the computer's network access (DHCPCriteria)
// because the subscriber may still be able to access the network
// from a different modem.

DeviceLoop:
{
    if (Device.deviceType == DOCSIS_MODEM)
    {
        get-new-batch(AUTOMATIC, NO_CONFIRMATION)

        // AUTOMATIC is the activation mode because we are
        // attempting to reset the modem so that becomes
        // disabled. NO_CONFIRMATION is the confirmation mode
        // because we do not want the batch to fail if we cannot
        // reset the modem. If the modem is off, it will
        // be disabled when it is turned back on.

        // Let's change the COS of the device so that it will restrict
        // the bandwidth usage of the modem.

        changeClassOfService(
            Device.MAC_ADDRESS,
            // macAddress: unique identifier for this modem
            "DisabledCOS");
            // newClassOfService: restricts bandwidth usage

        // Let's change the DHCP Criteria so that the modem will get an IP
        // address from a disabled CNR scope. This scope also points to
        // a spoofing DNS server so that the subscriber gets a restricted
        // access page.

        changeDHCPCriteria(
            Device.MAC_ADDRESS,
            // macAddress: unique identifier for this modem
            "DisabledDHCPCriteria");
            // newDHCPCriteria: disables Internet access
    }
}
```



```
    }  
  }  
  // end DeviceLoop
```

**Note**

You may need to consider the impact on the CPE behind the modem when defining the characteristics of DisabledCOS and resetting the modem. This is especially important if you have voice end points behind the modem, because disrupting the cable modem might affect the telephone conversation in progress at that time.

The subscriber is now disabled.

Preprovisioning Modems/Self-Provisioned Computers

A new subscriber contacts the service provider and requests service. The subscriber has a computer installed in a single dwelling unit. The service provider preprovisions all its cable modems in bulk.

Desired Outcome

Use this workflow to bring a preprovisioned cable modem, and an unprovisioned computer, online in the roaming standard mode. This must be done so that both devices have the appropriate level of service and are registered.

- Step 1** The service provider chooses a subscriber username and password for the billing system.
- Step 2** The service provider selects services that the subscriber can access.
- Step 3** The service provider's field technician installs the physical cable to the new subscriber's house and installs the preprovisioned device, connecting it to the subscriber's computer.
- Step 4** The technician turns on the modem and BAC gives it a provisioned IP address.
- Step 5** The technician turns on the computer and BAC gives it a private IP address.
- Step 6** The technician starts a browser application on the computer and points the browser to the service provider's user interface.
- Step 7** The technician accesses the service provider's user interface to complete the steps required for registering the computer behind the provisioned cable modem.

```

// To provision a computer:
// First we query the computer's lease information to its
// MAC address. We use the computers IP address (the web browser
// received this when the subscriber opened the service provider's
// web interface)

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION is the activation mode because this is a query
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the device

LeaseResults computerLease =
    getAllForIPAddress("10.0.14.38");
    // ipAddress:

String computerMACAddress = computerLease.
    getSingleLease().get(DeviceDetailsKeys.MAC_ADDRESS);

// MSO admin UI calls the provisioning API to provision a computer.

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION will generate a new configuration for the computer
// however it will not attempt to reset it.
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the computer because this cannot be done.

add(
    DeviceType.COMPUTER,
        // deviceType: Computer
    computerMACAddress,
        // macAddress: derived from computer lease
    null,
        // hostName: not used in this example
    null,
        // domainName: not used in this example
    "0123-45-6789",
        // ownerID: here, account number from billing system
    null,
        // ClassOfService : get the default COS
    "provisionedCPE",
        // DHCP Criteria: Network Registrar uses this to
        // select a modem lease granting provisioned IP address
    null
        // properties: not used
);

```

**Note**

The `IPDeviceKeys.MUST_BE_BEHIND_DEVICE` property is not set on the computer and this allows roaming from behind one cable modem to another.

- Step 8** The technician restarts the computer and the computer receives a new provisioned IP address. The cable modem and the computer are now both provisioned devices. The computer has access to the Internet through the service provider's network.

Modifying an Existing Modem

A service provider's subscriber currently has a level of service known as **Silver** and has decided to upgrade to **Gold** service. The subscriber has a computer installed at home.



Note

The intent of this use case is to show how to modify a device. You can apply this example to devices provisioned in modes other than roaming standard.

Desired Outcome

Use this workflow to modify an existing modem's Class of Service and pass that change of service to the service provider's external systems.

- Step 1** The subscriber phones the service provider and requests to have service upgraded. The service provider uses its user interface to change the Class of Service from **Silver** to **Gold**.
- Step 2** The service provider's application makes these API calls in BAC:

```
get-new-batch(AUTOMATIC, NO_CONFIRMATION)

// AUTOMATIC will generate a configuration for the device
// and will attempt to reset the device
// The configuration will be able to be generated because
// the device has booted. NO_CONFIRMATION is the confirmation
// mode because we don't want the batch to fail if the reset failed.
// This use case is a perfect example of the different
// confirmation modes that could be used instead of
// NO_CONFIRMATION. These confirmation modes give you
// a method to test whether a configuration was taken by
// a device. Also, these modes will take more time because
// the batch has to wait for the modem to reset.

changeClassOfService(
    "1,6,00:11:22:33:44:55",
        // macAddress: unique identifier for this modem
    "Gold"
        // newClassOfService
);
```

The subscriber can now access the service provider's network with the *Gold* service.

Unregistering and Deleting a Subscriber's Devices

A service provider needs to delete a subscriber who has discontinued service.

Desired Outcome

Use this workflow to permanently remove all the subscriber's devices from the service provider's network.

-
- Step 1** The service provider's user interface discontinues service to the subscriber.
- Step 2** The service provider's application uses a provisioning client program to request a list of all the subscriber's devices from BAC, and unregisters and resets each device so that it is brought down to the default (unprovisioned) service level.



Note

If the device specified as the parameter to the "unregister" API is already in unregistered state then the status code from the API call will be set to `CommandStatusCodes .CMD_ERROR_DEVICE_UNREGISTER_UNREGISTERED_ERROR`. This is normal/expected behavior.

```
// MSO admin UI calls the provisioning API to get a list of
// all the subscriber's devices.

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION is the activation mode because this is a query.
// NO_CONFIRMATION is the confirmation mode because we are
// not attempting to reset the devices.

getAllForOwnerID(
    "0123-45-6789");
    // query all the devices for this account number

// We need to unregister all the devices behind each modem(s) or else the
// unregister call for that modem will fail.

// for each computer in the retrieved list:
DeviceLoop:
{

    if (Device.deviceType == COMPUTER)
    {

        get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

        // NO_ACTIVATION is the activation mode because we can't
        // to reset Computers.
        // NO_CONFIRMATION is the confirmation mode because
        // the unregister call will not reset the devices.

        unregister(
            Device.MAC_ADDRESS,
            // macAddress: unique identifier for this device
        );

    }

}
```

```
}  
  
// for each modem in the retrieved list:  
DeviceLoop:  
{  
  
    if (Device.deviceType == DOCSIS_MODEM)  
    {  
  
        get-new-batch(AUTOMATIC, NO_CONFIRMATION)  
  
        // AUTOMATIC is the activation mode because we want  
        // to reset as we unregister the device.  
  
        unregister(  
            Device.MAC_ADDRESS,  
            // macAddress: unique identifier for this device  
        );  
  
    }  
}  
  
// end DeviceLoop.
```

**Note**

The next step is optional as some service providers prefer to keep the cable modem in the database unless it is broken. This step needs to be run only if the devices need to be deleted from the database.

Step 3

The service provider's application uses a provisioning client program to delete each of the subscriber's remaining devices individually from the database.

```

// MSO admin UI calls the provisioning API to get a list of
// all the subscriber's devices.

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION is the activation mode because this is a query.
// NO_CONFIRMATION is the confirmation mode because we are
// not attempting to reset the devices.

getAllForOwnerID(
    "0123-45-6789");
    // query all the devices for this account number
// for each device in the retrieved list:

DeviceLoop:
{
    // get a new batch for each modem being deleted

    if (Device.deviceType == DOCSIS_MODEM)
    {
        get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

        // NO_ACTIVATION is the activation mode because we don't want
        // to reset as we are deleting the device.
        // NO_CONFIRMATION is the confirmation mode because
        // the delete call will not reset the devices.

        delete(
            Device.MAC_ADDRESS,
            // macAddress: unique identifier for this device
            true
            // deleteDevicesBehind: deletes CPEs behind this modem.
        );
    }
}
//end DeviceLoop.

```

Self-Provisioning First-Time Activation in Promiscuous Mode

The subscriber has a computer (with a browser application) installed in a single dwelling unit and has purchased a DOCSIS cable modem.

Desired Outcome

Use this workflow to bring a new unprovisioned DOCSIS cable modem and computer online with the appropriate level of service.

-
- Step 1** The subscriber purchases a DOCSIS cable modem and installs it at home.
 - Step 2** The subscriber powers on the modem, and BAC gives it restricted access.

- Step 3** The subscriber starts a browser application on the computer and a spoofing DNS server points the browser to the service provider's registration server (for example, an OSS user interface or a mediator).
- Step 4** The subscriber uses the service provider's user interface to complete the steps required for registration, including selecting a Class of Service.

The service provider's user interface passes the subscriber's information to BAC, including the selected Class of Service and computer IP address. The subscriber's cable modem and computer are then registered with BAC.

```

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION is the activation mode because this is a
// query. NO_CONFIRMATION is the confirmation mode because
// we are not attempting to reset the device.
// First we query the computer's information to find the
// modems MAC address.
// We use the computer's IP address (the web browser
// received this when the subscriber opened the service
// provider's web interface). We also assume that "bostonProvGroup"
// is the provisioning group used in that locality.

List provGroupList;
provGroupList = provGroupList.add("bostonProvGroup");
Map computerLease = getAllForIPAddress(
    "10.0.14.38",
    // ipAddress: restricted access computer lease
    provGroupList
    // provGroups: List containing provgroup)

// Derive the modem MAC address from the computer's network
// information. The 1,6, is a standard prefix for an Ethernet
// device. The fully qualified MAC address is required by BACC

String modemMACAddress = "1,6," +
    computerLease.GetSingleLease().get(RELAY_AGENT_REMOTE_ID);

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// Now let's provision the modem
// NO_ACTIVATION will generate a configuration for the
// modem, however it will not attempt to reset it
// The configuration will be able to be generated because
// the modem has booted.
// NO_CONFIRMATION is the confirmation mode because we
// are not attempting to reset the modem
// Create a Map for the properties of the modem

Map properties;
// Set the property ModemKeys.PROMISCUOUS_MODE_ENABLED
// to enable promiscuous mode on modem

properties.put(ModemKeys.PROMISCUOUS_MODE_ENABLED, "enabled");

properties.put(ModemKeys.CPE_DHCP_CRITERIA, "provisionedCPE");

add(
    DeviceType.DOCSIS,

```

```

        // deviceType: DOCSIS
modemMACaddress,
        // macAddress: derived from computer lease
null,
        // hostName: not used in this example
null,
        // domainName: not used in this example
"0123-45-6789",
        // ownerID: here, account number from billing system
"Silver",
        // ClassOfService
"provisionedCM",
        // DHCP Criteria: Network Registrar uses this to
        // select a modem lease granting provisioned IP address
properties
        // properties:
);

```

Step 5 The user interface prompts the subscriber to reboot the computer.

Step 6 The provisioning client calls `performOperation()` to reboot the modem and gives the modem provisioned access.

```

get-new-batch(AUTOMATIC, NO_CONFIRMATION)

// AUTOMATIC is the activation mode because we are attempting
// to reset the modem so that it receives the new class of service.
// NO_CONFIRMATION is the confirmation mode because we don't want
// the batch to fail if we can't reset the modem. The user might
// have power cycled the modem when they rebooted their computer
// send a batch to reset the modem now that the user has been
// notified to reboot their computer

performOperation(
    DeviceOperation.RESET,
        //deviceOperation: Reset operation
modemMACaddress,
        // macAddress:Modem's MAC address
null
        // properties : not used
);

```

Step 7 When the computer is rebooted, it receives a new IP address. The cable modem is now a provisioned device. The computer is not registered with BAC, but it gains access to the Internet through the service provider's network. Computers that are online behind promiscuous modems are still available using the provisioning API.

Bulk Provisioning 100 Modems in Promiscuous Mode

A service provider wants to preprovision 100 cable modems for distribution by a customer service representative at a service kiosk.

Desired Outcome

Use this workflow to distribute modem data for all modems to new subscribers. The customer service representative has a list of modems available for assignment.

-
- Step 1** The cable modem's MAC address data for new or recycled cable modems is collected into a list at the service provider's loading dock.
 - Step 2** Modems that are assigned to a particular kiosk are bulk-loaded into BAC and are flagged with the identifier for that kiosk.
 - Step 3** When the modems are distributed to new subscribers at the kiosk, the customer service representative enters new service parameters, and changes the Owner ID field on the modem to reflect the new subscriber's account number.

```

// get a single batch for bulk load or 100 modems

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)
// The activation mode for this batch should be NO_ACTIVATION.
// NO_ACTIVATION should be used in this situation because no
// network information exists for the devices because they
// have not booted yet. A configuration can't be generated if no
// network information is present. And because the devices
// have not booted, they are not online and therefore cannot
// be reset. NO_CONFIRMATION is the confirmation mode because
// we are not attempting to reset the devices.
// Create a Map for the properties of the modem
Map properties;

// Set the property ModemKeys.PROMISCUOUS_MODE_ENABLED to
// enable promiscuous mode on modem.
// This could be done at a system level if promiscuous mode
// is your default provisioning mode.
properties.put(ModemKeys.PROMISCUOUS_MODE_ENABLED, "enabled")

// The ModemKeys.CPE_DHCP_CRITERIA is used to specify the DHCP
// Criteria to be used while selecting IP address scopes for
// CPEs behind this modem in the promiscuous mode.

properties.put(ModemKeys.CPE_DHCP_CRITERIA, "provisionedCPE");

// for each modem MAC-address in list:

ModemLoop:
{
    add(
        DeviceType.DOCSIS,
            // deviceType: DOCSIS
        modemMACaddress,
            // macAddress: derived from computer lease
        null,
            // hostName: not used in this example
        null,
            // domainName: not used in this example
        "0123-45-6789",
            // ownerID: here, account number from billing system
        "Silver",
            // ClassOfService
        "provisionedCM",
            // DHCP Criteria: Network Registrar uses this to
            // select a modem lease granting provisioned IP address
        properties
            // properties:
    );
}
//end ModemLoop.

```

Preprovisioning First-Time Activation in Promiscuous Mode

A new subscriber contacts the service provider and requests service. The subscriber has a computer installed in a single dwelling unit.

Desired Outcome

Use this workflow to bring a new unprovisioned cable modem and computer online with the appropriate level of service.

-
- Step 1** The service provider chooses a subscriber username and password for the billing system.
 - Step 2** The service provider selects the services that the subscriber can access.
 - Step 3** The service provider registers the device using its own user interface.
 - Step 4** The service provider's user interface passes information, such as the modem's MAC address and the Class of Service to BAC. Additionally, the modem gets a CPE DHCP criteria setting that lets Network Registrar select a provisioned address for any computers to be connected behind the modem. The new modem is then registered with BAC.
 - Step 5** The service provider's field technician installs the physical cable to the new subscriber's house and installs the preprovisioned device, connecting it to the subscriber's computer.

```
// MSO admin UI calls the provisioning API to pre-provision
// an HSD modem.

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// The activation mode for this batch should be NO_ACTIVATION.
// NO_ACTIVATION should be used in this situation because no
// network information exists for the modem because it has not
// booted. A configuration cannot be generated if no network
// information is present. And because the modem has not booted,
// it is not online and therefore cannot be reset.
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the modem.
// Create a map for the properties of the modem.

Map properties;

// Set the property ModemKeys.PROMISCUOUS_MODE_ENABLED to enable
// promiscuous mode on modem

properties.put(ModemKeys.PROMISCUOUS_MODE_ENABLED, "enabled")

properties.put(ModemKeys.CPE_DHCP_CRITERIA, "provisionedCPE");

add(
    DeviceType.DOCSIS,
        // deviceType: DOCSIS
        "1,6,00:11:22:33:44:55",
        // macAddress: derived from computer lease
    null,
```

```

        // hostName: not used in this example
    null,
        // domainName: not used in this example
    "0123-45-6789",
        // ownerID: here, account number from billing system
    "Silver",
        // ClassOfService
    "provisionedCM",
        // DHCP Criteria: Network Registrar uses this to
        // select a modem lease granting provisioned IP address
    properties
        // properties:
    );

```

Step 6 The technician powers on the cable modem and BAC gives it provisioned access.

Step 7 The technician powers on the computer and BAC gives it provisioned access. The cable modem and the computer are now both provisioned devices. The computer has access to the Internet through the service provider's network.

Replacing an Existing Modem

A service provider wants to replace a broken modem.



Note

If the computer has the option restricting roaming from one modem to another, and the modem is replaced, the computer's MAC address for the modem must also be changed.

Desired Outcome

Use this workflow to physically replace an existing cable modem with a new modem without changing the level of service provided to the subscriber.

Step 1 The service provider changes the MAC address of the existing modem to that of the new modem.

```

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ ACTIVATION is the activation mode because we will
// not be able to reset as the new modem has not booted
// on the network.
// NO_CONFIRMATION is the confirmation mode because we are
// not trying to reset the modem
// To change the MAC address of a DOCSIS modem:

changeMACAddress (
    "1,6,00:11:22:33:44:55"
    // old macAddress: unique identifier for the old modem

```

```

"1,6,11:22:33:44:55:66"
    /// new macAddress: unique identifier for the new modem
);

```

- Step 2** The service provider replaces the cable modem and turns it on. The computer must also be turned on.
- Step 3** The cable modem is now a fully provisioned device with the appropriate level of service, as is the computer behind the cable modem.
-

Adding a Second Computer in Promiscuous Mode

A subscriber wishes to connect a second computer behind an installed cable modem.

Desired Outcome

Use this workflow to ensure that the subscriber's selected service permits the connection of multiple CPE, and that the subscriber has network access from both connected computers.



Note This case does not require calls to the provisioning API.

- Step 1** The subscriber connects a second computer behind the cable modem.
- Step 2** The subscriber turns on the computer.
- Step 3** If the subscriber's selected service permits connecting multiple CPE, BAC gives the second computer access to the Internet.
-

Self-Provisioning First-Time Activation with NAT

A university has purchased a DOCSIS cable modem with network address translation (NAT) and DHCP capability. The five occupants of the unit each have a computer installed with a browser application.

Desired Outcome

Use this workflow to bring a new unprovisioned cable modem (with NAT) and the computers behind it online with the appropriate level of service.

- Step 1** The subscriber purchases a cable modem with NAT and DHCP capability and installs it in a multiple dwelling unit.
- Step 2** The subscriber turns on the modem and BAC gives it restricted access.
- Step 3** The subscriber connects a laptop computer to the cable modem, and the DHCP server in the modem provides an IP address to the laptop.
- Step 4** The subscriber starts a browser application on the computer and a spoofing DNS server points the browser to the service provider's registration server (for example, an OSS user interface or a mediator).

- Step 5** The subscriber uses the service provider's user interface to complete the steps required for cable modem registration of the modem. The registration user interface detects that the modem is using NAT and registers the modem, making sure that the modem gets a Class of Service that is compatible with NAT.

```

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION is the activation mode because this is a query.
// NO_CONFIRMATION is the confirmation mode because we are not
// attempting to reset the device.
// First we query the computer's information to find the modems
// MAC address.

// With NAT, the computer is never seen by the Network Registrar
// DHCP server. Instead, the modem has translated the IP address
// it assigned to the computer, so the web browser sees the
// modem's IP address. When the lease data is examined, the MAC
// address for the device and the relay-agent-remote-id are the
// same, indicating that this is an unprovisioned modem device,
// which is therefore presumed to be performing NAT.

LeaseResults modemLease =
    getAllForIPAddress("10.0.14.38");
    // ipAddress: restricted access.

String modemMACAddress = modemLease.
    getSingleLease().get(DeviceDetailsKeys.MAC_ADDRESS);
// MSO client registration program then calls the provisioning
// API to provision the NAT modem (with an appropriate class of
// service for NAT).

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION will generate a configuration for the modem
// however it will not attempt to reset it.
// The configuration will be able to be generated because the
// modem has booted. NO_CONFIRMATION is the confirmation mode
// because we are not attempting to reset the modem

add(
    DeviceType.DOCSIS,
        // deviceType: DOCSIS
    modemMACAddress,
        // macAddress: derived from its lease
    null,
        // hostName: not used in this example
    null,
        // domainName: not used in this example
    "0123-45-6789",
        // ownerID: here, account number from billing system
    "Silver",
        // ClassOfService
    "provisionedCM",
        // DHCP Criteria: Network Registrar uses this to
        // select a modem lease granting provisioned IP address

```

```

        null
        // properties: not used
    };

```

Step 6 The user interface prompts the subscriber to reboot the computer.

Step 7 The provisioning client calls `performOperation()` to reboot the modem and gives the modem provisioned access.

```

get-new-batch(AUTOMATIC, NO_CONFIRMATION)

// AUTOMATIC is the activation mode because we are attempting
// to reset the modem so that it receives its new class of service.
// NO_CONFIRMATION is the confirmation mode because we don't want
// the batch to fail if we can't reset the modem. The user might
// have power cycled the modem when they rebooted their computer
// send a batch to reset the modem now that the user has been
// notified to reboot their computer

performOperation(
    DeviceOperation.RESET,
        //deviceOperation: Reset operation
    "1,6,00:11:22:33:44:55",
        // macAddress:Modem's MAC address
    null
        // properties : not used
    );

```

Step 8 The cable modem is now fully provisioned and the computers behind it have full access to the service provider's network.



Note

Certain cable modems with NAT may require you to reboot the computer to get the new Class of Service settings. If the cable modem and NAT device are separate devices, the NAT device must also be registered similarly to registering a computer.

Adding a New Computer Behind a Modem with NAT

The landlord of an apartment building has four tenants sharing a modem and accessing the service provider's network. The landlord wants to provide Internet access to a new tenant, sharing the building's modem. The modem has NAT and DHCP capability. The new tenant has a computer connected to the modem.

Desired Outcome

Use this workflow to bring a new unprovisioned computer online with a previously provisioned cable modem so that the new computer has the appropriate level of service.

**Note**

This case does not require calls to the provisioning API.

Step 1 The subscriber turns on the computer.

Step 2 The computer is now a provisioned device with access to the appropriate level of service.

**Note**

The provisioned NAT modem hides the computers behind it from the network.

Move Device to Another DHCP Scope

A service provider is renumbering its network causing a registered cable modem to require an IP address from a different Network Registrar scope.

Desired Outcome

A provisioning client changes the DHCP criteria, and the cable modem receives an IP address from the corresponding DHCP scope.

Step 1 Change the DOCSIS modem's DHCP criteria to "newmodemCriteria".

```
get-new-batch(AUTOMATIC, NO_CONFIRMATION);
// AUTOMATIC is the Activation mode because we are attempting
// to reset the modem so that a phone line is disabled
// NO_CONFIRMATION is the Confirmation mode because we don't
// want the batch to fail if we can't reset the modem.

// This use case assumes that the DOCSIS modem has been
// previously added to the database

changeDHCPCriteria(
    "1,6,ff:00:ee:11:dd:22"
    // Modem's MAC address or FQDN
    "newmodemCriteria"
);
```

Step 2 The modem gets an IP address from the scope targeted by "newmodemCriteria."

Log Device Deletions Using Events

A service provider has multiple provisioning clients and wants to log device deletions.

Desired Outcome

When any provisioning client deletes a device, the provisioning client logs an event in one place.

- Step 1** Create a listener for the device deletion event. This class must extend the DeviceAdapter abstract class or, alternatively, implement the DeviceListener interface. This class must also override the `deletedDevice(DeviceEvent ev)` method in order to log the event.

```
public DeviceDeletionLogger
    extends DeviceAdapter
    //Extend the DeviceAdapter class.
{
    public void deletedDevice(DeviceEvent ev)
        //Override deletedDevice.
    {
        logDeviceDeletion(ev.getDeviceID());
        //Log the deletion.
    }
}
```

- Step 2** Register the listener and the qualifier for the events using the PACEConnection interface.

```
DeviceDeletionLogger deviceDeletionLogger =
    new DeviceDeletionLogger();
    // Modem's MAC address or FQDN
    "newmodemCriteria"
qualifier = new DeviceEventQualifier();
// We are interested only in device deletion.
qualifier.setDeletedDevice ();
// Add device listener using PACEConnection
connection.addDeviceListener(deviceDeletionLogger, qualifier
);
```

- Step 3** When a device is deleted from the system, the event is generated, and the listener is notified.

Monitoring an RDU Connection Using Events

A service provider is running a single provisioning client and wants notification if the connection between the provisioning client and the RDU breaks.

Desired Outcome

Use this workflow to have the event interface notify the service provider if the connection breaks.

-
- Step 1** Create a listener for the messaging event. This class must extend the `MessagingAdapter` abstract class or, alternatively, implement the `MessagingListener` interface. This class must override the `connectionStopped(MessagingEvent ev)` method.

```
// Extend the service provider's Java program using the
// provisioning client to receive Messaging events.
public MessagingNotifier
    extends MessagingAdapter
    //Extend the MessagingAdapter class.
{
    public void connectionStopped(MessagingEvent ev)
        //Override connectionStopped.
    {
        doNotification(ev.getAddress(), ev.getPort());
        //Do the notification.
    }
}
```

- Step 2** Register the listener and the qualifier for the events using the `PACEConnection` interface.

```
MessagingQualifier qualifier =
    new MessagingQualifier();
qualifier.setAllPersistentConnectionsDown();
MessagingNotifier messagingNotifier = new MessagingNotifier();
connection.addMessagingListener(messagingNotifier, qualifier
);
```

- Step 3** If a connection breaks, the event is generated, and the listener is notified.
-

Logging Batch Completions Using Events

A service provider has multiple provisioning clients and wants to log batch completions.

Desired Outcome

When any provisioning client completes a batch, an event is logged in one place.

-
- Step 1** Create a listener for the event. This class must extend the `BatchAdapter` abstract class or implement the `BatchListener` interface. This class must override the `completion(BatchEvent ev)` method in order to log the event.

```
public BatchCompletionLogger
    extends BatchAdapter
    //Extend the BatchAdapterclass.
{
    public void completion(BatchEvent ev)
        //Override completion.
    {
        logBatchCompletion(ev.BatchStatus().getBatchID());
        //Log the completion.
    }
}
```

- Step 2** Register the listener and the qualifier for the events using the `PACEConnection` interface.

```
BatchCompletionLogger batchCompletionLogger =
    new BatchCompletionLogger();
Qualify All qualifier = new Qualify All();
connection.addBatchListener(batchCompletionLogger , qualifier
);
```

- Step 3** When a batch completes, the event is generated, and the listener is notified.
-

Getting Detailed Device Information

A service provider wants to allow an administrator to view detailed information for a particular device.

Desired Outcome

The service provider's administrative application displays all known details about a given device, including MAC address, lease information, provisioned status of the device, and the device type (if known).

Step 1 The administrator enters the MAC address for the device being queried into the service provider's administrative user interface.

Step 2 BAC queries the embedded database for the device details.

```
get-new-batch(AUTOMATIC, NO_CONFIRMATION);

// MSO admin UI calls the provisioning API to query the details
// for the requested device. Query may be performed based on MAC
// address or IP address, depending on what is known about the
// device.
Map deviceDetails =
    getDetails(
        "1,6,00:11:22:33:44:55",
        // macORFqdn: unique identifier for the device
        true
        // needLeaseInfo: yes we need it
    );
```

Step 3 The service provider's application presents a page of device data details, which can display everything that is known about the requested device. If the device was connected to the service provider's network, this data includes lease information (for example, IP address and relay agent identifier). The data indicates whether the device was provisioned, and if it was, the data also includes the device type.

```
// extract device detail data from the map
String deviceType = (String)deviceDetails.get(DEVICE_TYPE);
String macAddress = (String)deviceDetails.get(MAC_ADDRESS);
String ipAddress = (String)deviceDetails.get(IP_ADDRESS);
String relayAgentID = (String)deviceDetails.get(RELAY_AGENT_ID);
Boolean isProvisioned = (Boolean)deviceDetails.get(IS_PROVISIONED);
// The admin UI now formats and prints the detail data to a view page
```

Searching Using the Default Class of Service

A service provider wants to allow an administrator to view data for all modems with the **default** Class of Service for DOCSIS device type.

Desired Outcome

The service provider's administrative application returns a list of DOCSIS devices with the default Class of Service.

Step 1 The administrator selects the search option in service provider's administrative user interface.

Step 2 BAC queries the embedded database for a list of all MAC addresses for the devices that match the requested default Class of Service.

```

get-new-batch(AUTOMATIC, NO_CONFIRMATION);

// Create a MACAddressSearchType to indicate search by
// default class of service.

MACAddressSearchType mst =
    new MACAddressSearchType.getByDefaultClassOfService(
        DeviceType.DOCSIS);

// Create a MACAddressSearchFilter to get 20 devices
// at a time. This indicates that we have a page size of 20.

MACAddressSearchFilter mySearchFilter =
    new MACAddressSearchFilter(
        mst,
        // type: MAC address search type
        false,
        // isInclusive:
        20
        // maximumReturned:
    );

// MAC address to start the search from.
String startMac = null;

// A list containing the MAC addresses returned from
// search.
List deviceList = null;

// If the size of deviceList is equal to 20 then
// there may be devices matching the search criteria.
while ((deviceList == null) ||
        (deviceList.size() == 20))
{
    // Use the provisioning API call to search BACC
    // database. The search starts from the MAC address
    // "startMac". If "startMac" is null then the search
    // starts from the very beginning of the index.

    deviceList = search (mySearchFilter, startMac);

    // See Step 3 for the definition of processMACAddressList

    startMac = processMACAddressList(deviceList);
}

```

- Step 3** The service provider's application requests details on these devices from BAC, and presents a page of device data. For each device, the code provides for display of the device type, MAC address, client class, and provisioned status of the device, one device per line.

```

processMACAddressList (List deviceList)
{
    Iterator iter = deviceList.iterator();

    String startMac = null;

    while (iter.hasNext())
    {
        startMac = (String) iter.next();
        // Get details for this device
        Map detailMap = getDetails (
            startMac,
            // MAC of current device
            true
            // yes, we need lease info
        );

        // extract device detail data from each map
        // this can be used for displaying in UI
        String deviceType = (String)detailMap.get (DEVICE_TYPE);
        String macAddress = (String)detailMap.get (MAC_ADDRESS);
        String clientClass = (String)detailMap.get (CLIENT_CLASS);
        Boolean isProvisioned = (Boolean)detailMap.get (IS_PROVISIONED);
        // format and print above data in output line
    }

    // We return the last MAC address in the list
    // so that the next search can be started from here

    return startMAC;
}

```

Retrieving Devices Matching a Vendor Prefix

A service provider wants to allow an administrator to view data for all devices matching a particular vendor prefix.

Desired Outcome

The service provider's administrative application returns a list of devices matching the requested vendor prefix.

-
- Step 1** The administrator enters the substring matching the desired vendor prefix into the service provider's administrator user interface.
 - Step 2** BAC queries the embedded database for a list of all MAC addresses for the devices that match the requested vendor prefix.

```
// Create a MACAddressPattern corresponding to the requested
// vendor prefix

MACAddressPattern pattern =
    new MACAddressPattern(
        "1,6,ff:00:ee:*",
        // macAddressPattern: the requested vendor prefix
    );

// Create a MACAddressSearchType to indicate search by
// MAC address pattern

MACAddressSearchType mst =
    new MACAddressSearchType.getDevices(pattern);

// Create a MACAddressSearchFilter to get 20 devices
// at a time. This indicates that we have a page size of 20.

MACAddressSearchFilter mySearchFilter =
    new MACAddressSearchFilter(
        mst,
        // type: MAC address search type
        false,
        // isInclusive:
        20
        // maximumReturned:
    );

// MAC address to start the search from.
String startMac = null;

// A list containing the MAC addresses returned from
// search.
List deviceList = null;

// If the size of deviceList is equal to 20 then
// there may be devices matching the search criteria.

while ((deviceList == null) ||

        (deviceList.size() == 20))
{

    // Use the provisioning API call to search BACC
    // database. The search starts from the MAC address
    // "startMac". If "startMac" is null then the search
    // starts from the very beginning of the index.

    deviceList = search(mySearchFilter, startMac);

    // See Step 3 for the definition of processMACAddressList

    startMac = processMACAddressList(deviceList);
}
```

Step 3 The service provider's application requests details on these devices from BAC, and presents a page of device data. For each device, the code displays the device type, MAC address, client class, and provisioned status of the device. One device is identified per line.

```
processMACAddressList (List deviceList)
{
    Iterator iter = deviceList.iterator();

    String startMac = null;

    while (iter.hasNext())
    {
        startMac = (String) iter.next();
        // Get details for this device
        Map detailMap = getDetails (
            startMac,
            // MAC of current device
            true
            // yes, we need lease info
        );

        // extract device detail data from each map
        // this can be used for displaying in UI
        String deviceType = (String)detailMap.get (DEVICE_TYPE);
        String macAddress = (String)detailMap.get (MAC_ADDRESS);
        String clientClass = (String)detailMap.get (CLIENT_CLASS);
        Boolean isProvisioned = (Boolean)detailMap.get (IS_PROVISIONED);
        // format and print above data in output line
    }

    // We return the last Mac address in the list
    // so that the next search can be started from here

    return startMac;
}
```

Preprovisioning PacketCable eMTA

A new customer contacts a service provider to order PacketCable voice service. The customer expects to receive a provisioned embedded MTA.

Desired Outcome

Use this workflow to preprovision an embedded MTA so that the modem MTA component has the appropriate level of service when brought online.



Note This use case skips the call agent provisioning that is required for making telephone calls from eMTAs.

Step 1 The service provider chooses a subscriber username and password for the billing system.

- Step 2** The service provider chooses the appropriate Class of Service and DHCP criteria for the modem component and adds it to BAC.

```

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// Let's provision the modem and the MTA component in the same
// batch. This can be done because the activation mode of this
// batch is NO_ACTIVATION. More than one device can be operated
// on in a batch if the activation mode does not lead to more
// than one device being reset.
// To add a DOCSIS modem:

add(
    DeviceType.DOCSIS,
        // deviceType: DOCSIS
    "1,6,01:02:03:04:05:06",
        // macAddress: scanned from the label
    null
        // hostName: not used in this example
    null
        // domainName: not used in this example
    "0123-45-6789",
        // ownerID: here, account number from billing system
    "Silver",
        // classOfService
    "provisionedCM",
        // DHCP Criteria: Network Registrar uses this to
        // select a modem lease granting provisioned IP address
    null
        // properties: not used
);

```

- Step 3** The service provider chooses the appropriate Class of Service and DHCP criteria for the MTA component and adds it to BAC.

```

// Continuation of the batch in Step2
// To add the MTA component:

add(
    DeviceType.PACKET_CABLE_MTA,
        // deviceType: PACKET_CABLE_MTA
    "1,6,01:02:03:04:05:07",
        // macAddress: scanned from the label
    null,
        // hostName: not used in this example, will be auto generated
    null,

```

```

        // domainName: not used in this example, will be auto generated.
        // The FqdnKeys.AUTO_FQDN_DOMAIN property must be set somewhere in the
        // property hierarchy.
"0123-45-6789",
        // ownerID: here, account number from billing system
"Silver",
        // ClassOfService
"provisionedMTA",
        // DHCP Criteria: Network Registrar uses this to
        // select an MTA lease granting provisioned IP address
null
        // properties: not used
);

```

Step 4 The embedded MTA gets shipped to the customer.

Step 5 The customer brings the embedded MTA online and makes telephone calls using it.

SNMP Cloning on PacketCable eMTA

A customer has an SNMP Element Manager that wishes to gain access to a PacketCable eMTA.

Desired Outcome

An external Element Manager is granted secure SNMPv3 access to the PacketCable eMTA.



Note

Changes made to RW MIB variables are not permanent and are not updated in the BAC configuration for the eMTA. The information written into the eMTA MIB is lost the next time the MTA powers down or resets.

Step 1 Call the provisioning API method, `performOperation()`, passing in the MAC address of the MTA and the username of the new user to create on the MTA. This will be the username used in subsequent SNMP calls by the Element Manager.

```

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION);

// NO_ACTIVATION is the activation mode because we don't want to
// reset the device.
// NO_CONFIRMATION is the confirmation mode because we are
// not attempting to reset the device.

// The goal here is to create a new user on the MTA indicated
// by the MAC address. The other parameter needed here is the new
// user name, which is passed in the Map.
// Create a map that contains one element - the name of
// the new user to be created on the MTA
HashMap map = new HashMap();
map.put( SNMPPPropertyKeys.CLONING_USERNAME, "newUser" );
// The first param is the actual device operation to perform.
performOperation(
    DeviceOperation.ENABLE_SNMPV3_ACCESS,
        // deviceOperation : ENABLE_SNMPV3_ACCESS
    "1,6,00:00:00:00:00:99",
        // macORFqdn : MAC Address of the modem
    map
        // parameters: operation specific parameters
);

```

- Step 2** The provisioning API attempts to perform an SNMPv3 cloning operation to create an entry on the MTA for the new user passed in step 1. The keys used in the new user entry row are a function of two passwords defined within BAC. These passwords will be made available to the customer and the RDU command passes these passwords (the auth and priv password) through a key localization algorithm to create an auth and priv key. These are stored, along with the new user, in the eMTA's user table.

**Note**

The auth and priv passwords mentioned in this step may be changed by setting `SNMPPPropertyKeys.CLONING_AUTH_PASSWORD (/snmp/cloning/auth/password)` and `SNMPPPropertyKeys.CLONING_PRIV_PASSWORD (/snmp/cloning/priv/password)` properties respectively in the `rdu.properties` configuration file.

- Step 3** The customer issues SNMPv3 request using above specified username, passwords, and key localization algorithm to allow for secure communication with the MTA.

Incremental Provisioning of PacketCable eMTA

A customer has a PacketCable eMTA in service with its first line (end point) enabled. The customer wants to enable the second telephone line (end point) on the eMTA and connect a telephone to it.

Desired Outcome

The customer should be able to connect a telephone to the second line (end point) on the eMTA and successfully make phone calls from it without any service interruption.

**Note**

In order to use the second line on the eMTA, the Call Agent need to be configured accordingly. This use case does not address provisioning of call agents.

Step 1

The service provider's application invokes the BAC API to change the Class of Service of the eMTA. The new Class of Service supports two end points on the eMTA. This change in Class of Service does not take effect until the eMTA is reset. Disrupting the eMTA is not desirable; therefore, incremental provisioning is undertaken in the next step.

```
get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION is the activation mode because we don't want to
// reset the device.
// NO_CONFIRMATION is the Confirmation mode because we are not
// disrupting the device.

changeClassOfService(
    "1,6,ff:00:ee:11:dd:22" // eMTA's MAC address or FQDN
    ,"twoLineEnabledCOS" // This COS supports two lines.
);
```

Step 2

The service provider's application uses the BAC incremental update feature to set SNMP objects on the eMTA and thereby enabling the service without disrupting the eMTA.

```
// The goal here is to enable a second phone line, assuming one
// phone line is currently enabled. We will be adding a new
// row to the pktcNcsEndPntConfigTable.

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION is the activation mode because we don't want to
// reset the device.
// NO_CONFIRMATION is the confirmation mode because we are
// not attempting to reset the device.

// Create a map containing one element - the list of SNMP
// variables to set on the MTA
HashMap map = new HashMap();

// Create an SnmpVarList to hold SNMP varbinds
SnmpVarList list = new SnmpVarList();

// An SnmpVariable represents an oid/value/type triple.

// pktcNcsEndPntConfigTable is indexed by the IfNumber, which in this
// case we will assume is interface number 12 (this is the last
// number in each of the oids below.

// The first variable represents the creation of a new row in
// pktcNcsEndPntConfigTable we are setting the RowStatus
// column (column number 26). The value of 4 indicates that
// a new row is to be created in the active state.
SnmpVariable variable = new SnmpVariable(
```

```

        ".1.3.6.1.4.1.4491.2.2.2.1.2.1.1.26.12",
        "4",
        SnmpType.INTEGER );
list.add( variable );

// The next variable represents the call agent id for this new
// interface, which we'll assume is 'test.com'
SnmpVariable variable = new SnmpVariable(
        ".1.3.6.1.4.1.4491.2.2.2.1.2.1.1.1.12",
        "test.com",
        SnmpType.STRING );
list.add( variable );

// The final variable represents the call agent port
SnmpVariable variable = new SnmpVariable(
        ".1.3.6.1.4.1.4491.2.2.2.1.2.1.1.2.12",
        "2728",
        SnmpType.INTEGER );
list.add( variable );

// Add the SNMP variable list to the Map to use in the API call
map.put( SNMPPropertyKeys.SNMPVAR_LIST, list );

// Invoke the BACC API to do incremental update on the eMTA.
performOperation( DeviceOperation.INCREMENTAL_UPDATE // device operation
        , "1,6,00:00:00:00:00:99" // MAC Address
        , map // Parameters for the operation
        );

```

- Step 3** The eMTA is enabled to use the second telephone line. The eMTA continues to receive the same service, after being reset, since the Class of Service was changed in step 1.
-

Preprovisioning DOCSIS Modems with Dynamic Configuration Files

A new customer contacts a service provider to order a DOCSIS modem with high-speed *Gold* data service for two CPE behind it.

Desired Outcome

Use this workflow to preprovision a DOCSIS modem with a Class of Service that uses DOCSIS templates. The dynamic configuration file generated from the templates is used while the modem comes online.

- Step 1** The service provider chooses a subscriber username and password for the billing system.
- Step 2** The service provider chooses Gold Class of Service, and the appropriate DHCP criteria, and then adds the cable modem to BAC.

```

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

Map properties;

// Set the property ModemKeys.PROMISCUOUS_MODE_ENABLED to enable
// promiscuous mode on modem

properties.put(ModemKeys.PROMISCUOUS_MODE_ENABLED, "enabled")

// No CPE DHCP Criteria is specified.
// The CPEs behind the modem will use the default provisioned
// promiscuous CPE DHCP criteria specified in the system defaults.

// This custom property corresponds to a macro variable in the
// DOCSIS template for "gold" class of service indicating the
// maximum number of CPEs allowed behind this modem. We set it
// to two CPEs from this customer.

properties.put("docsis-max-cpes", "2");

// To add a DOCSIS modem:

add(
    DeviceType.DOCSIS,
        // deviceType: DOCSIS
    "1,6,01:02:03:04:05:06",
        // macAddress: scanned from the label
    null,
        // hostName: not used in this example
    null,
        // domainName: not used in this example
    "0123-45-6789",
        // ownerID: here, account number from billing system
    "gold",
        // classOfService:
    "provisionedCM",
        // DHCP Criteria: Network Registrar uses this to
        // select a modem lease granting provisioned IP address
    properties
        // properties:
);

```

Step 3 The cable modem is shipped to the customer.

Step 4 The customer brings the cable modem online and connects the computers behind it.

Optimistic Locking

An instance of the service provider application needs to ensure that it is not overwriting the changes made by another instance of the same application.

Desired Outcome

Use this workflow to demonstrate the optimistic locking capabilities provided by the BAC API.



Note

Locking of objects is done in multi-user systems to preserve integrity of changes, so that one person's changes do not accidentally get overwritten by another. With optimistic locking, you write your program assuming that any commit has a chance to fail if at least one of the objects being committed was changed by someone else since you began the transaction.

Step 1 The service representative selects the search option in the service provider's user interface and enters the cable modem's MAC address.

Step 2 BAC queries the embedded database, gets the details of the device and the MSO user interface displays the information.

```
// MSO admin UI calls the provisioning API to query the details
// for the requested device.
Map deviceDetails =
    getDetails(
        "1,6,00:11:22:33:44:55",
        // macORFqdn: unique identifier for the device
        true
        // needLeaseInfo: yes we need it
    );

// extract device detail data from the map
String deviceType = (String)deviceDetails.get(DEVICE_TYPE);
String macAddress = (String)deviceDetails.get(MAC_ADDRESS);
String ipAddress = (String)deviceDetails.get(IP_ADDRESS);
String relayAgentID = (String)deviceDetails.get(RELAY_AGENT_ID);

Boolean isProvisioned = (Boolean)deviceDetails.get(IS_PROVISIONED);
// service provider admin UI displays this information.

// Let's save the OID_REVISION_NUMBER property so that we can see in
// step 3.
String oidRevisionNumber = (String)deviceDetails.get(OID_REVISION_NUMBER);
```

Step 3 The service representative attempts to change the Class of Service and the DHCP criteria of the modem using the user interface. This in turn invokes the BAC API.

```

// We need a reference to Batch instance so that ensureConsistency()
// method can be invoked on it.
Batch batch = conn.newBatch() ;

List oidList = new ArrayList();
// Add the oid-rev number saved from step 2 to the list
oidList.add(oidRevisionNumber);

// Sends a list of OID revision numbers to validate before processing the
// batch. This ensures that the objects specified have not been modified
// since they were last retrieved.
batch.ensureConsistency(oidList);

    batch.changeClassOfService (
        "1,6,00:11:22:33:44:55",
        // macORFqdn: unique identifier for the device.
        "gold"
        // newCOSName : Class of service name.
    )

batch.changeDHCPCriteria (
    "1,6,00:11:22:33:44:55",
    // macORFqdn: unique identifier for the device.
    "specialDHCPCriteria"
    // newDHCPCriteria : New DHCP Criteria.
)

// This batch fails with BatchStatusCodes.BATCH_NOT_CONSISTENT,
// in case if the device is updated by another client in the mean time.
// If there is a conflict occurs, then the service provider client
// is responsible for resolving the conflict by querying the database
// again and then applying changes appropriately.

```

Step 4 The user is ready to receive Gold Class of Service with appropriate DHCP criteria.

Temporarily Throttling a Subscriber's Bandwidth

An MSO has a service that allows a subscriber to download only 10 MB of data a month. Once the subscriber reaches that limit, their downstream bandwidth is turned down from 10 MB to 56 K. When the month is over they are moved back up to 10 MB.



Note

You may want to consider changing upstream bandwidth as well, since peer-to-peer users and users who run websites tend to have heavy upload bandwidth.

Desired Outcome

Use this workflow to move subscribers up and down in bandwidth according to their terms of agreement.

Step 1 The MSO has a rate tracking system, such as NetFlow, which keeps track of each customer's usage by MAC address. Initially a customer is provisioned at the Gold Class of Service level with 1 MB downstream.

- Step 2** When the rate tracking software determines that a subscriber has reached the 10-MB limit it notifies the OSS. The OSS then makes a call into the BAC API to change the subscriber's Class of Service from Gold to Gold-throttled.

```
get-new-batch(AUTOMATIC, NO_CONFIRMATION)

// AUTOMATIC is the activation mode because we are
// attempting to reset the modem so that it
// receives low bandwidth service.
// NO_CONFIRMATION is the confirmation mode
// because we do not want the batch to fail if we cannot
// reset the modem. If the modem is off, when it will
// be disabled when it is turned back on.

// Let's change the COS of the device so that it restricts
// bandwidth usage of the modem.
    changeClassOfService(
        Device.MAC_ADDRESS,
        // macAddress: unique identifier for this modem
        "Gold-throttled");
        // newClassOfService: restricts bandwidth usage to 56k
```

- Step 3** At the end of the billing period, the OSS calls the BAC API to change the subscriber's Class of Service back to *Gold*.

Preprovisioning CableHome WAN-MAN

A new customer contacts a service provider to order home networking service. The customer expects a provisioned CableHome device to be shipped to him.

Desired Outcome

Use this workflow to preprovision a CableHome device so that the cable modem and WAN-MAN components on it will have appropriate level of service when brought online.

- Step 1** The service provider chooses a subscriber username and password for the billing system.
- Step 2** The service provider chooses the appropriate Class of Service and the DHCP criteria for the modem component, then adds it to BAC.

```
get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// Let's provision the modem and the WAN-Man component in the same
// batch.
// To add a DOCSIS modem:

add(
    DeviceType.DOCSIS,
    // deviceType: DOCSIS
```

```

"1,6,01:02:03:04:05:06",
    // macAddress: scanned from the label
null,
    // hostName: not used in this example
null,
    // domainName: not used in this example
"0123-45-6789",
    // ownerID: here, account number from billing system
"Silver",
    // classOfService
"provisionedCM",
    // DHCP Criteria: Network Registrar uses this to
    // select a modem lease granting provisioned IP address
null
    // properties: not used
);

```

Step 3 The service provider chooses the appropriate Class of Service and DHCP criteria for the WAN-MAN component, then adds it to BAC.

```

// Continuation of the batch in Step2
// To add the WAN-Man component:
add(
    DeviceType.CABLEHOME_WAN_MAN,
        // deviceType: CABLEHOME_WAN_MAN
"1,6,01:02:03:04:05:07",
    // macAddress: scanned from the label
null,
    // hostName: not used in this example.
null,
    // domainName: not used in this example.
"0123-45-6789",
    // ownerID: here, account number from billing system
"silverWanMan",
    // ClassOfService
"provisionedWanMan",
    // DHCP Criteria: Network Registrar uses this to
    // select an MTA lease granting provisioned IP address
null
    // properties: not used
);

```

- Step 4** The CableHome device gets shipped to the customer.
- Step 5** The customer brings the CableHome device online.
-

CableHome with Firewall Configuration

A customer contacts a service provider to order a home networking service with the firewall feature enabled. The customer expects to receive a provisioned CableHome device.

Desired Outcome

Use this workflow to preprovision a CableHome device so that the cable modem and the WAN-MAN components on it, have the appropriate level of service when brought online.

- Step 1** The service provider chooses a subscriber username and password for the billing system.
- Step 2** The service provider chooses the appropriate Class of Service and DHCP criteria for the cable modem component, then adds it to BAC.

```
get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// Let's provision the modem and the WAN-Man component in the same
// batch.
// To add a DOCSIS modem:

add(
    DeviceType.DOCSIS,
        // deviceType: DOCSIS
    "1,6,01:02:03:04:05:06",
        // macAddress: scanned from the label
    null,
        // hostName: not used in this example
    null,
        // domainName: not used in this example
    "0123-45-6789",
        // ownerID: here, account number from billing system
    "Silver",
        // classOfService
    "provisionedCM",
        // DHCP Criteria: Network Registrar uses this to
        // select a modem lease granting provisioned IP address
    null
        // properties: not used
);
```

- Step 3** The service provider chooses the appropriate Class of Service and DHCP criteria for the WAN-MAN component and adds it to BAC.

```

// Continuation of the batch in Step2
// To add the WAN-Man component:

// Create a Map to contain WanMan's properties

Map properties;

// The fire wall configuration for the Wan Man component is specified
// using the CableHomeKeys.CABLEHOME_WAN_MAN_FIREWALL_FILE property.
// This use case assumes that the firewall configuration file named
// "firewall_file.cfg" is already present in the RDU database and the
// firewall configuration is enabled in the Wan Man configuration file
// specified with the corresponding class of service.

properties.put(CableHomeKeys.CABLEHOME_WAN_MAN_FIREWALL_FILE,
"firewall_file.cfg");

add(
    DeviceType.CABLEHOME_WAN_MAN,
        // deviceType: CABLEHOME_WAN_MAN
    "1,6,01:02:03:04:05:07",
        // macAddress: scanned from the label
    null,
        // hostName: not used in this example.
    null,
        // domainName: not used in this example.
    "0123-45-6789",
        // ownerID: here, account number from billing system
    "silverWanMan",
        // ClassOfService
    "provisionedWanMan",
        // DHCP Criteria: Network Registrar uses this to
        // select an MTA lease granting provisioned IP address
    properties
        // properties: contains the firewall config file
);

```

Step 4 The CableHome device gets shipped to the customer.

Step 5 The customer brings the CableHome device online and the cable modem and the WAN-MAN component get provisioned IP addresses and proper configuration files.

Retrieving Device Capabilities for CableHome WAN-MAN

A service provider wants to allow an administrator to view capabilities information for a CableHome WAN-MAN device.

Desired Outcome

The service provider's administrative application displays all known details about a given CableHome WAN-MAN component, including MAC address, lease information, provisioned status, and the device capabilities information.

-
- Step 1** The administrator enters the MAC address of the WAN-MAN being queried into the service provider's user interface.
- Step 2** BAC queries the embedded database for details of the device identified using the MAC address entered.

```
get-new-batch(NO_ACTIVATION, NO_CONFIRMATION);

// MSO admin UI calls the provisioning API to query the details
// for the requested device.

Map deviceDetails =
    getDetails(
        1,6,00:11:22:33:44:55", "
        // macORFqdn: unique identifier for the device
        true
        // needLeaseInfo: yes we need it
    );
```

- Step 3** The service provider's application then presents a page of device data details, which can display everything that is known about the requested device. If the device was connected to the service provider's network, this data includes lease information, such as the IP address or the relay agent identifier. This data indicates whether the device is provisioned. If it is provisioned, the data also includes the device type and device capabilities information.

```

// extract device details information from the map
String deviceType      = (String)  deviceDetails.get(DeviceDetailsKeys.DEVICE_TYPE);
String macAddress      = (String)  deviceDetails.get(DeviceDetailsKeys.MAC_ADDRESS);
String ipAddress       = (String)  deviceDetails.get(DeviceDetailsKeys.IP_ADDRESS);
String relayAgentID    = (String)  deviceDetails.get(
                                   DeviceDetailsKeys.RELAY_AGENT_ID);
Boolean isProvisioned = (Boolean)  deviceDetails.get(
                                   DeviceDetailsKeys.IS_PROVISIONED);
String formation       = (String)  deviceDetails.get(
                                   IPDeviceCapabilities.FORMATION);
String deviceList      = (String)  deviceDetails.get(
                                   IPDeviceCapabilities.DEVICE_LIST);
String serNum          = (String)  deviceDetails.get(
                                   IPDeviceCapabilities.SERIAL_NUMBER);
String hwVer           = (String)  deviceDetails.get(
                                   IPDeviceCapabilities.HARDWARE_VERSION);
String swVer           = (String)  deviceDetails.get(
                                   IPDeviceCapabilities.SOFTWARE_VERSION);
String brVer           = (String)  deviceDetails.get(
                                   IPDeviceCapabilities.BOOT_ROM_VERSION);
String vendorOui       = (String)  deviceDetails.get(
                                   IPDeviceCapabilities.VENDOR_OUI);
String modelNum        = (String)  deviceDetails.get(
                                   IPDeviceCapabilities.MODEL_NUMBER);
String vendorNum       = (String)  deviceDetails.get(
                                   IPDeviceCapabilities.VENDOR_NAME);
String sysDesc         = (String)  deviceDetails.get(
                                   IPDeviceCapabilities.SYSTEM_DESCRIPTION);
String fwVer           = (String)  deviceDetails.get(
                                   IPDeviceCapabilities.FIRMWARE_VERSION);
String fwVer           = (String)  deviceDetails.get(
                                   IPDeviceCapabilities.FIREWALL_VERSION);
// The admin UI now formats and prints the detail data to a view page


```

Self-Provisioning CableHome WAN-MAN

A subscriber has a computer with a browser application installed in a single dwelling unit and has purchased an embedded CableHome device.

Desired Outcome

Use this workflow to bring a new unprovisioned embedded CableHome device online with the appropriate level of service, and give the subscriber Internet access from computers connected to the embedded CableHome device.

-
- Step 1** The subscriber purchases an embedded CableHome device and installs it at home.
- Step 2** The subscriber powers on the embedded CableHome device. BAC gives the embedded cable modem restricted access, allowing two CPE: one for the CableHome WAN-MAN and the other for the computer.
-
-  **Note** This use case assumes an unprovisioned DOCSIS modem allows two CPE behind it. Until configured to do otherwise, BAC supports only a single device behind an unprovisioned DOCSIS modem. You can change this behavior by defining an appropriate Class of Service that supports two CPE and then using it as the default Class of Service for DOCSIS devices.
-
- Step 3** BAC configures the CableHome WAN-MAN, including IP connectivity and downloading the default CableHome boot file. The default CableHome boot file configures the CableHome device in passthrough mode. The CableHome device is still unprovisioned.
- Step 4** The subscriber connects the computer to the CableHome device. The computer gets an unprovisioned (restricted) IP address. The subscriber starts a browser application on the computer. A spoofing DNS server points the browser to the service provider's registration server (for example, an OSS user interface or a mediator).
- Step 5** The subscriber uses the service provider's user interface to complete the steps required for cable modem registration, including selecting a Class of Service. The subscriber also selects a CableHome Class of Service.
- Step 6** The service provider's user interface passes the subscriber's information to BAC, including the selected Class of Service for cable modem and CableHome, and computer IP address. The subscriber is then registered with BAC.

```
get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// NO_ACTIVATION is the activation mode because this is a
// query. NO_CONFIRMATION is the confirmation mode because
// we are not attempting to reset the device.
// First we query the computer's information to find the
// modems MAC address.
// We use the computers IP address (the web browser
// received this when the subscriber opened the service
// providers web interface). We also assume that "bostonProvGroup"
// is the provisioning group used in that locality.

List provGroupList;
provGroupList = provGroupList.add("bostonProvGroup");
Map computerLease = getAllForIPAddress(
    "10.0.14.38",
    // ipAddress: restricted access computer lease
```

```

        provGroupList
        // provGroups: List containing provgroup)
// Derive the modem MAC address from the computer's network
// information. The 1,6, is a standard prefix for an Ethernet
// device. The fully qualified MAC address is required by BPR

String modemMACAddress = "1,6," +

        computerLease.getSingleLease().get(RELAY_AGENT_REMOTE_ID);

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

// Now let's provision the modem
// NO_ACTIVATION will generate a configuration for the
// modem however it will not attempt to reset it
// The configuration will be able to be generated because
// the modem has booted.
// NO_CONFIRMATION is the confirmation mode because we
// are not attempting to reset the modem
// Create a Map for the properties of the modem

Map properties;
// Set the property ModemKeys.PROMISCUOUS_MODE_ENABLED
// to enable promiscuous mode on modem

properties.put (ModemKeys.PROMISCUOUS_MODE_ENABLED, "enabled");

properties.put (ModemKeys.CPE_DHCP_CRITERIA, "provisionedCPE");

add(
    DeviceType.DOCSIS,
        // deviceType: DOCSIS
    modemMACAddress,
        // macAddress: derived from computer lease
    null,
        // hostName: not used in this example
    null,
        // domainName: not used in this example
    "0123-45-6789",
        // ownerID: here, account number from billing system
    "Silver",
        // ClassOfService
    "provisionedCM",
        // DHCP Criteria: Network Registrar uses this to
        // select a modem lease granting provisioned IP address
    properties
        // properties:
);

```

Step 7 The user interface prompts the subscriber to reboot the computer.

Step 8 The provisioning client calls `performOperation()` to reboot the modem and gives the modem provisioned access.


```

get-new-batch(AUTOMATIC, NO_CONFIRMATION)

// AUTOMATIC is the activation mode because we are attempting
// to reset the modem so that it receives its new class of service
// NO_CONFIRMATION is the confirmation mode because we don't want
// the batch to fail if we can't reset the modem. The user might
// have power cycled the modem when they rebooted their computer
// send a batch to reset the modem now that the user has been
// notified to reboot their computer

performOperation(
    DeviceOperation.RESET,
        //deviceOperation: Reset operation
    modemMACAddress,
        // macAddress:Modem's MAC address
    null
        // properties : not used
);

```

Step 9 When the computer is rebooted, it receives a new IP address from the CableHome device's DHCP server. The cable modem and the CableHome device are now both provisioned. Now the subscriber can connect a number of computers to the Ethernet ports of the CableHome device and they have access to the Internet.

**Note**

If the configuration file supplied to the WAN-MAN component enables the WAN-Data component on the box, it will get provisioned in the promiscuous mode. This assumes that the promiscuous mode is enabled at the technology defaults level for the DeviceType.CABLEHOME_WAN_DATA device type.

Lease Reservation Use Cases

This section describes use cases specifically related to the use of the lease reservation feature. Standard administrative operations, such as managing Class of Service, DHCP criteria, licenses, and so on are not addressed here. The lease reservation use cases include:

- [Bringing a Device Online Using IP Address Provided by Service Provider, page C-50](#)
- [Removing and Re-Creating a Reservation, page C-51](#)
- [Assigning a New Device with an Old Device's IP Address, page C-52](#)
- [Removing a Reservation and Assigning a New IP Address, page C-53](#)
- [Rebooting a Device with the Same IP Address, page C-54](#)
- [Removing a Device from BAC, page C-55](#)
- [A Submitted Batch Fails when BAC Uses CCM, page C-56](#)
- [A Submitted Batch Fails when BAC Does Not Use CCM, page C-56](#)

API Calls Affected by the Lease Reservation Feature

The implementation of these API calls supports the lease reservation feature:

- `IPDevice.add(DeviceType DeviceType, String macAddress, String hostName, String domainName, String ownerID, String cosName, String dhcpCriteria, Map Properties)`
- `IPDevice.changeProperties(String macORFqdn, Map newPropToAdd, List propToDelete)`
- `IPDevice.changeMACAddress(String macORFqdn, String newMAC)`
- `IPDevice.delete(String macORFqdn, Boolean deleteDevicesBehind)`

Bringing a Device Online Using IP Address Provided by Service Provider

When a device is added to the BAC system datastore, the service provider configures that device with the specific IP address for the device using a BAC property (`IPDeviceKeys.IP_RESERVATION`).

Desired Outcome

Bring a device online with the exact IP address set in the property by the service provider. The granted IP address is reserved (lease reservation) for that device.

-
- Step 1** The service provider adds a new device to the BAC system. The service provider configures the new device with a specific IP address 10.10.10.1.
- Step 2** The service provider's user interface passes device information to BAC, such as the MAC address, FQDN, and Class of Service. The BAC property (`IPDeviceKeys.IP_RESERVATION`) is used to reserve a specific IP address (10.10.10.1) for this device.

```
Map properties;

// Set the property IPDeviceKeys.IP_RESERVATION to a specific
// IP address

properties.put( IPDeviceKeys.IP_RESERVATION, "10.10.10.1");

// To add a DOCSIS modem:

get-new-batch(NO_ACTIVATION, NO_CONFIRMATION)

add(
    DeviceType.DOCSIS,
        // deviceType: DOCSIS
    "1,6,01:02:03:04:05:06",
        // macAddress: scanned from the label
    null,
        // hostName: not used in this example
    null,
        // domainName: not used in this example
    "0123-45-6789",
        // ownerID: here, account number from billing system
```

```

"gold",
    // classOfService:
"provisionedCM",
    // DHCP Criteria: Network Registrar uses this to
    // select a modem lease granting provisioned IP address
properties
    // properties:
);

```

- Step 3** The new device is then registered with BAC. The reservation of 10.10.10.1 is also created for MAC address “01:02:03:04:05:06” in the BAC/Network Registrar system.
- Step 4** When the device is booted, it receives the exact IP address (10.10.10.1) set in the property by the service provider.

Rollback occurs if needed when the API command results in an error during processing or the change failed to commit to the RDU database. The command implementation removes the lease reservation of 10.10.10.1 for MAC address “01:02:03:04:05:06” from the BAC/Network Registrar if it was made during command processing.

Removing and Re-Creating a Reservation

After a device is registered to the BAC system with a reserved IP address, the service provider reassigns the device with a different IP address using a BAC property (`IPDeviceKeys.IP_RESERVATION`).

Desired Outcome

The device is rebooted with the exact IP address set in the property by the service provider. The reservation will be removed and re-created. The previously assigned IP address is unreserved for that device; and the new IP address is granted and reserved (lease reservation) for that device.

- Step 1** A device is registered to BAC with a reserved IP address of 10.10.10.1.
- Step 2** The service provider’s application makes these API calls in BAC to change the reserved IP to 10.10.10.5.

```

get-new-batch(AUTOMATIC, NO_CONFIRMATION);
// AUTOMATIC is the Activation mode because we are attempting
// to reset the device
// NO_CONFIRMATION is the Confirmation mode because we don't
// want the batch to fail if we can't reset the modem.

// This use case assumes that the DOCSIS modem has been
// previously added to the database

```

```

Map properties;

```

```
// Set the property IPDeviceKeys.IP_RESERVATION to a specific
// IP address

properties.put( IPDeviceKeys.IP_RESERVATION, "10.10.10.5");
// To reassign a different IP to:

    changeProperties(
        "1,6,01:02:03:04:05:06",
        // macAdd
        properties, null
    );
```

Step 3 The reservation of 10.10.10.1 for MAC address “01:02:03:04:05:06” is removed from the BAC/Network Registrar system. The reservation of 10.10.10.5 for MAC address “01:02:03:04:05:06” is created in the BAC/Network Registrar system.

Step 4 When the device is rebooted as the result of device disruption by PACE disruptor (AUTOMATIC is used in the Activation mode), it receives the exact IP address (10.10.10.5) set in the property by the service provider.

Rollback occurs if needed when the API command results in an error during processing or the change failed to commit to the RDU database. The command implementation attempts to roll the device back to the prior working configuration. In this case, reservation of 10.10.10.1 for MAC address “01:02:03:04:05:06” will be re-added to the BAC/Network Registrar system if it was removed; the reservation of 10.10.10.5 for MAC address “01:02:03:04:05:06” will be removed from BAC/Network Registrar system if it was created during command processing.

Assigning a New Device with an Old Device’s IP Address

After a device is registered to the BAC system with a reserved IP address, the service provider needs to replace the broken device with a device with a new MAC address.

Desired Outcome

The new device is booted with the same IP address granted to the broken old device. The reservation will be removed and recreated, as if the IP address of the device changed.

Step 1 A device is registered to BAC with a reserved IP address of 10.10.10.5.

Step 2 The service provider changes the MAC address of the existing device (“01:02:03:04:05:06”) to that of the new device (“01:02:03:04:05:07”) in the BAC system.

```
get-new-batch(AUTOMATIC, NO_CONFIRMATION);
// NO_ACTIVATION is the activation mode because we will
// not be able to reset as the new device has not booted
// on the network.
// NO_CONFIRMATION is the confirmation mode because we are
// not trying to reset the device

// To change the MAC address of a device:

changeMACAddress (
    "1,6,01:02:03:04:05:06",
        // old macAddress: unique identifier for the old device
    "1,6,01:02:03:04:05:07"
        //// new macAddress: unique identifier for the new device
);
```

Step 3 The reservation of 10.10.10.5 for MAC address “01:02:03:04:05:06” is removed from the BAC/Network Registrar system. The reservation of 10.10.10.5 for MAC address “01:02:03:04:05:07” is created in the BAC/Network Registrar system.

Step 4 When the device with MAC address “01:02:03:04:05:07” is turned on, it receives the same IP address (10.10.10.5) granted to the broken old device.

Rollback occurs if needed when the API command results in an error during processing or the change failed to commit to the RDU database. The command implementation attempts to roll the device back to the prior working configuration. In this case, the reservation of 10.10.10.5 for MAC address “01:02:03:04:05:06” will be re-added to BAC/Network Registrar system if it was removed, the reservation of 10.10.10.5 for MAC address “01:02:03:04:05:07” will be removed from BAC/Network Registrar system if it was created during command processing.

Removing a Reservation and Assigning a New IP Address

After a device is registered to the BAC system with a reserved IP address, the service provider needs to remove the reservation of the specific IP address since the device no longer needs a reserved IP assignment.

Desired Outcome

The reservation will be removed from the system and Network Registrar/BAC selects the next available IP address in the appropriate IP pool based on the selected DHCP criteria and assigns it to the device.

Step 1 A device is registered to BAC with a reserved IP address of 10.10.10.5.

Step 2 The service provider's application makes these API calls in BAC to remove the reservation.

```
get-new-batch(AUTOMATIC, NO_CONFIRMATION);
// AUTOMATIC is the Activation mode because we are attempting
// to reset the device
// NO_CONFIRMATION is the Confirmation mode because we don't
// want the batch to fail if we can't reset the modem.

// Add the property IPDeviceKeys.IP_RESERVATION the list to be removed

list.add( IPDeviceKeys.IP_RESERVATION);

// To reassign a different IP to:

changeProperties(
    "1,6,01:02:03:04:05:07",
    // macAdd
    null, list
);
```

- Step 3** The reservation of 10.10.10.5 for MAC address "1,6,01:02:03:04:05:07" is removed from the system.
- Step 4** When the device is rebooted as the result of device disruption (AUTOMATIC is used in the Activation mode), dynamic address assignment occurs. Network Registrar/BAC selects the next available IP address in an appropriated IP address pool based on the selected DHCP criteria set on the device and grants it to the device.

Rollback occurs if needed when the API command results in an error during processing or the change failed to commit to the RDU database. The command implementation attempts to roll the device back to the prior working configuration. In this case, the reservation of 10.10.10.5 for MAC address "01:02:03:04:05:07" will be re-added from the BAC/Network Registrar system if it was removed during command processing.

Rebooting a Device with the Same IP Address

After a device is registered to the BAC system with a provisioned IP address (a dynamic IP assignment, not a reserved IP) granted by BAC/Network Registrar, the service provider reassigns the device with a different IP address using a BAC property (`IPDeviceKeys.IP_RESERVATION`).

Desired Outcome

The device is rebooted with the exact IP address set in the property (`IPDeviceKeys.IP_RESERVATION`) by the service provider. The granted new IP address is now reserved (lease reservation) for that device.

- Step 1** A device is registered to BAC with dynamic IP address (Network Registrar/BAC selected an IP address in an appropriated IP pool based on selected DHCP criteria on the device and granted it to the device).
- Step 2** The service provider's application makes these API calls in BAC to reassign the IP address.

```

get-new-batch(AUTOMATIC, NO_CONFIRMATION);
// AUTOMATIC is the Activation mode because we are attempting
// to reset the device
// NO_CONFIRMATION is the Confirmation mode because we don't
// want the batch to fail if we can't reset the modem.

Map properties;

// Set the property IPDeviceKeys.IP_RESERVATION to a specific
// IP address

properties.put( IPDeviceKeys.IP_RESERVATION, "10.10.10.1");

// To reassign a different IP to:

changeProperties(

    "1,6,01:02:03:04:05:08",
        // macAdd
        properties, null

    ):

```

Step 3 The reservation of 10.10.10.1 for Mac address "01:02:03:04:05:08" is made with BAC/Network Registrar.

Step 4 When the device is rebooted as the result of device disruption (AUTOMATIC is used in the Activation mode), it receives the exact IP address (10.10.10.1) set in the property by the service provider.

Rollback occurs if needed when the API command results in an error during processing or the change failed to commit to the RDU database. The command implementation attempts to roll the device back to the prior working configuration. The reservation of 10.10.10.1 for MAC address "01:02:03:04:05:08" will be removed from BAC/Network Registrar system if it was added during command processing. When the device reboots, BAC/Network Registrar selects the next available IP address in the appropriate IP pool based on the selected DHCP criteria and grants it to the device.

Removing a Device from BAC

A service provider needs to remove the subscriber's device with a reserved IP from the BAC system.

Desired Outcome

Permanently remove the subscriber's device from the BAC system. The granted IP address is unreserved for that device.

Step 1 A device is registered to BAC with a reserved IP address of 10.10.10.5.

Step 2 The service provider uses its own user interface to remove the device from the BAC system. The service provider's user interface, acting as a BAC client, passes the information to BAC. BAC updates the device information and removes the device.

```

delete(
    "1,6,01:02:03:04:05:07",
    // macAdd
    // deleteDevicesBehind res: unique identifier for this device
    true: deletes CPEs behind this modem.
):

```

- Step 3** The reservation of 10.10.10.5 for MAC address “01:02:03:04:05:07” is also removed from BAC/Network Registrar system.

Rollback occurs if needed when the API command results in an error during processing or the change failed to commit to the RDU database. The command implementation attempts to roll the device back to the prior working configuration. The reservation of 10.10.10.5 for MAC address "01:02:03:04:05:07" will be re-added from the BAC/Network Registrar system if it was removed during command processing.

A Submitted Batch Fails when BAC Uses CCM

BAC is configured to use CCM. The OSS submits an API request to add or remove a reservation, but CCM is unavailable (connection down, CCM is incorrectly configured, CCM License issue, and so on).

Desired Outcome:

When a submitted batch fails, no reservation should be added or removed in BAC/Network Registrar; a predefined, well-known error code is correctly returned.

-
- Step 1** The OSS builds and submits a batch containing the API calls listed in [API Calls Affected by the Lease Reservation Feature, page C-50](#), for adding or removing reservations.
- Step 2** The RDU receives the batch and processes it. During the processing, the RDU (via the API commands listed in [API Calls Affected by the Lease Reservation Feature, page C-50](#)) makes an external CCM call for lease reservation related tasks.
- Step 3** CCM is unavailable and an error occurs. Return status code from the API call will be set to `CommandStatusCodes .CMD_ERROR_CCM_UNREACHABLE`.
-

A Submitted Batch Fails when BAC Does Not Use CCM

BAC is *not* configured to use CCM. The OSS submits an API request to add or remove a reservation.

Desired Outcome

Submitted batch failed. No reservation is added/removed in the BAC/Network Registrar system; a predefined, well-known error code is correctly returned.

-
- Step 1** The OSS builds and submits a batch containing the API calls listed in [API Calls Affected by the Lease Reservation Feature, page C-50](#), for adding or removing reservations.

- Step 2** The RDU receives the batch and processes it. During the processing, the RDU (via the API commands listed in [API Calls Affected by the Lease Reservation Feature, page C-50](#)) detects that CCM is not configured for the lease reservation feature.
- Step 3** An error occurs. Return status code from the API call will be set to CommandStatusCodes .CMD_ERROR_CCM_NOT_CONFIGURED.
-

