CPE Management Overview

This chapter describes the management of customer premises equipment (CPE) by using the CPE WAN Management Protocol for Cisco Broadband Access Center (BAC).

This chapter includes the following sections:

CWMP Overview

Cisco BAC communicates with CPE through the CPE WAN Management Protocol (CWMP) according to parameters described in the TR196, TR-069, and other related data model specifications. CWMP encompasses secure management of CPE, including:

- Autoconfiguration and dynamic service provisioning
- Firmware management
- Device diagnostics
- Performance and status monitoring

Cisco BAC supports devices based on the TR-069, TR-098, TR-104, TR-106, TR-181, and TR-196 standards. This support includes Ethernet and ADSL gateway devices, wireless gateways, VoIP ATAs, and other devices compliant with CWMP. This release also provides for runtime-extensible data models to support any upcoming data-model standards or any vendor-specific data models.

Cisco BAC Device Object Model

The Cisco BAC device object model is crucial in controlling the configuration and firmware rules that are generated as instructions for the DPE to manage devices. This process occurs at the RDU, and is controlled through named attributes and relationships.

The main objects in the device object model are:

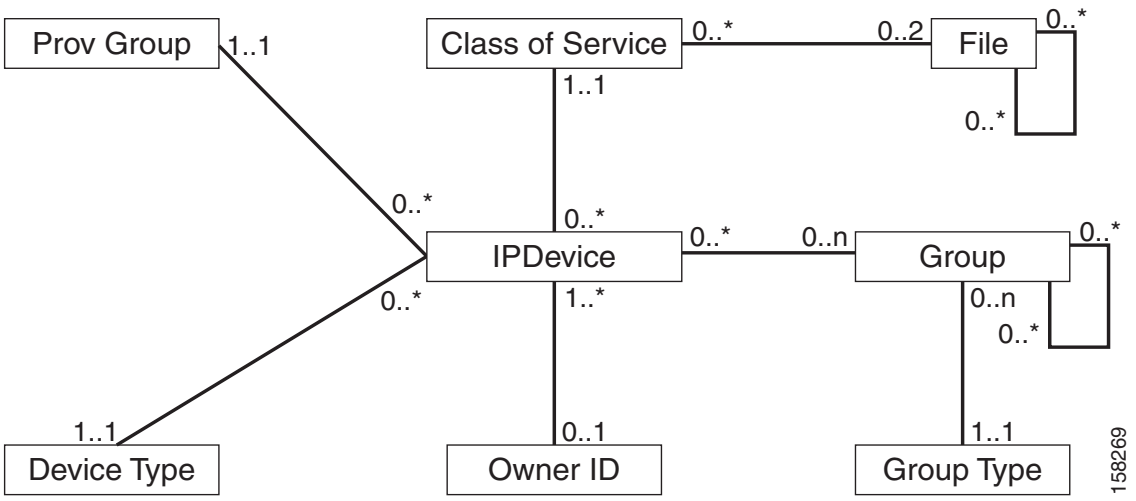
- IPDevice—Represents a network entity that requires provisioning.
- Owner ID—Represents an external identifier for a subscriber.
- Device Type—Represents the type of the device.
- ProvGroup—Represents a logical grouping of devices serviced by a specific set of DPEs.
- Class of Service—Represents the configuration profile to be assigned to a device.
- File—Serves as a container for files used in provisioning that include templates and firmware images.
- Group—Is a customer-specific mechanism for grouping devices.

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

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

See Figure 4-1 for a description of the interaction between the various objects in the device data model.

Figure 4-1 Device Object Model



In the Cisco BAC device object model, the IPDevice is related to the Class of Service, the Provisioning Group, and the Device Type. The Class of Service is then related to the Configuration Template and the Firmware Rules Template. Files can be related to one another, such as the firmware rules template being related to the firmware image.

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

Table 4-1 Device Object Relationships

Object	Related to ...
IP Device • Could be preregistered or unregistered (See Device Deployment in Cisco BAC, page 4-19). • Attributes include Device ID (OUI-Serial) and FQDN	• Owner ID • Provisioning Group • Class of Service • Device Type
Owner ID • Is associated with devices and, therefore, cannot exist without a device related to it. • Enables grouping; for example, you can group all devices belonging to *Joe*.	IP Device
Device Type • Stores defaults common to all devices of a technology, specifically CWMP. • Enables grouping; for example, you can group all CWMP devices.	IP Device
File • Stores files used in provisioning; for example, configuration template and firmware rules template	Class of Service
Class of Service • Attributes include Type, Name, and Properties. (For details, see Class of Service, page 4-3.)	• IP Device • File • Configuration Template (optional) • Firmware Rules Template (optional)

Class of Service

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

The different Classes of Service are:

- Registered—Specified by the user when the device is registered. This Class of Service is explicitly added to the device record using the application programming interface (API).
- Selected—Selected by the RDU for a device that, for one reason or another, cannot retain its registered Class of Service.
- Related—Related to the device by being registered, selected, or both.

If the selected Class of Service for a device is changed, the Instruction Generation Service regenerates instructions for the device configuration.

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

Other concepts related to the device data model are:

- Property Hierarchy, page 4-4
- Custom Properties, page 4-4

Property Hierarchy

While processing configuration templates for substitutable parameters, Cisco BAC searches objects for this property using a certain order called Property Hierarchy.

Cisco BAC properties allow you to access and store data in Cisco BAC using the API. Preprovisioned, discovered, and status data can be retrieved through the properties of corresponding objects, using the API. Properties also enable you to configure Cisco BAC at the appropriate level of granularity (from system level to device groups and to individual devices).

The Cisco BAC property hierarchy gives you the flexibility to define system-wide or service class defaults that can be overridden by individual devices.

Cisco BAC allows you to store any number of properties on objects in its data model. You can reference these properties in configuration templates or firmware rules. You can use properties in this property hierarchy:

1. Device
2. Group (priority is set through the Group Type, see Managing Group Types, page 16-18)
3. Provisioning Group
4. Class of Service
5. Device Type
6. System Defaults

Custom Properties

Custom properties allow for the definition of new properties, which can then be stored on any object via the API.

Custom properties are variable names defined in the RDU, and must not contain any spaces. The template parser works from bottom up when locating properties in the hierarchy and converts the template option syntax. For detailed information, see Custom Properties, page 5-16.

Discovering CPE Parameters

Cisco BAC is enabled to read CPE parameters using Remote Procedure Calls (RPCs) as defined in CWMP. This is made possible through the Data Synchronization Instruction. This instruction discovers data from the CPE device, reports it to the RDU, and keeps the RDU up-to-date when CPE device data changes.

This instruction can be used to keep the RDU up-to-date on such key parameters as the software version and the model name. These parameters may, in turn, be used to generate other instructions, such as configuration instructions specific to a given device type.

Table 4-2 lists the default parameters that Cisco BAC discovers.

Table 4-2 Default Discovered Parameters

Parameter	Description
Inform.DeviceId.Manufacturer	Identifies the manufacturer of the CPE.
Inform.DeviceId.ManufacturerOUI	Identifies the unique identifier of the CPE manufacturer.
Inform.DeviceId.ProductClass	Identifies the product or class of product over which the manufacturer's *SerialNumber* parameter is unique.
InternetGatewayDevice.DeviceInfo.HardwareVersion	Identifies the hardware version of the CPE.
InternetGatewayDevice.DeviceInfo.SoftwareVersion	Identifies the software version currently installed on the CPE.
InternetGatewayDevice.DeviceInfo.ModelName	Identifies the model name of the CPE.
InternetGatewayDevice.ManagementServer.ParameterKey	Specifies the value of the *ParameterKey* from the most recent SetParameterValues, AddObject, or DeleteObject call from the server.

These parameters can be updated using the API (using the /server/acs/discover/parameters property) or via the administrator user interface (see Discovering Data from Devices, page 12-12).

Note Even if the device has a different root object, such as Device, instead of InternetGatewayDevice, the parameters are still discovered.

Multi-Instance Object Support

In this release, the multi-instance object support introduces the ability to discover the multi-instance object parameters from the device without specifying the actual instance number. Multi-instance object support is available as an usability improvement in the following BAC modules to aid deployment:

- Configuration Template
- Firmware Rules Template
- Parameter Discovery
- Display Live Data Operation

This feature will be functional only when the RDU and all the DPEs are upgraded to Cisco BAC 3.8. To configure the configuration template and the firmware template with multi-instance objects the value of the newly introduced *templateVersion* attribute value should be 3.0.

Configuration Template

Earlier to this release of BAC, the scope of configuration template represented only the possible set parameter values and set parameter attributes that could be performed. There was no mechanism to discover object names.

In this release, the template scope is extended to include the following:

- The ability to discover object names
- The ability to discover parameter values
- The ability to delete object instances
- The ability to add object instances

A new attribute sync-method is introduced in the configuration template. The sync-method instructs the configuration sync process to interact with the device to discover the object instances. For the discovered object instances the ACS has total control to configure the parameters available in the object's sub-tree. The following are the available sync-methods.

Sync-method	Description
discovered	If this condition is specified, BAC discovers the object instances based on the instance attribute value specified for the object instance.
replace-all	If this condition is specified, BAC first discovers the parameter value for the specified parameter name from all the object instances. For the discovered object instances the following operations are possible: • If a matching object instance is found with the specified parameter value, it updates the object instance with the new parameter value. • If no matching instance is found, BAC creates an object instance with the new parameter value. • If an object instance is found with the different parameter value, that object instance is deleted.
replace-all-on-change	If this condition is specified, BAC first discovers the parameter value for the specified parameter name from all the object instances. For the discovered object instances the following operations are possible: • If all the object instances in the template match all the instances in the device then BAC updates the object instance with the new parameter value defined in the template. • Even if, one instance in the device does not have matching object instance in the template or if one object instance in the template does not have one matching instance in the device then BAC issues DeleteObect for all the instances in the device. After deleting all the instances, BAC creates object instances with the new parameter value defined in the template.
replace-augment	This operates same as replace-all, except that BAC retains the instances that does not match the parameters configured.
delete-all	If this condition is specified, BAC simply deletes all the available instances at that level.
delete	If this condition is specified, BAC deletes the instance that match the value specified in the instance attribute.

Any one of the following instance attribute must be mandatorily used with *discovered* and *delete* sync-method:

- all() - This option indicates BAC to discover all the instances
- last() - This option indicates BAC to discover the last instance
- compare(parameterName operation matchValue) - This option indicates BAC to discover the object instances in which the parameter and the value are in accordance with the operation.

Note Valid compare operators are: equals, equalsIgnoreCase, notEquals, lessThan, lessThanEquals, greaterThanEquals, greaterThan.

- Index - This option can be used for direct indexing and accepts a valid positive integer. This is the index into the returned instance names. For example, index value 1 returns the first instance.

In configuration template, the expression parameters in pre-requisites can also be configured with multi-instance object parameters, with any of the discovered options. For more information on prerequisites, see Prerequisites, page 5-9.

Example 4-1 Multi-Instance Object Using replace-all in Configuration Template

The following example illustrates usage of multi-instance object using replace-all in configuration template:

```
<Configuration templateVersion="3.0">
    <ParameterDictionaries>
        <ParameterDictionary>tr196-cwmp-dictionary-v2.0.xml</ParameterDictionary>
    </ParameterDictionaries>
    <ObjectInstance name="Device">
        <ObjectInstance name="Services">
            <ObjectInstance name="FAPService">
                <ObjectInstance name="{i}" sync-method="discovered" instance="all()">
                    <ObjectInstance name="CellConfig">
                        <ObjectInstance name="UMTS">
                            <ObjectInstance name="RAN">
                                <ObjectInstance name="NeighborList">
                                    <ObjectInstance name="IntraFreqCell"
sync-method="replace-all" primary-keys="PLMNID,LAC,RNCID">
                                        <ObjectInstance name="{i}">
                                            <Parameter>
                                                <Name>RAC</Name>
                                                <Value>10</Value>
                                            </Parameter>
                                            <Parameter>
                                                <Name>PLMNID</Name>
                                                <Value>VAR(name=FC-PLMN-ID,
defaultValue=123452)</Value>
                                            </Parameter>
                                            <Parameter>
                                                <Name>LAC</Name>
                                                <Value>65532</Value>
                                            </Parameter>
                                            <Parameter>
                                                <Name>RNCID</Name>
                                                <Value>18</Value>
                                            </Parameter>
                                        </ObjectInstance>
                                        <ObjectInstance name="">
                                            <Parameter>
                                                <Name>RAC</Name>
```

```
                                                    <Value>11</Value>
                                                </Parameter>
                                                <Parameter>
                                                    <Name>PLMNID</Name>
                                                    <Value>123452</Value>
                                                </Parameter>
                                                <Parameter>
                                                    <Name>LAC</Name>
                                                    <Value>65533</Value>
                                                </Parameter>
                                                <Parameter>
                                                    <Name>RNCID</Name>
                                                    <Value>21</Value>
                                                </Parameter>
                                            </ObjectInstance>
                                        </ObjectInstance>
                                    </ObjectInstance>
                                </ObjectInstance>
                                </ObjectInstance>
                            </ObjectInstance>
                        </ObjectInstance>
                    </ObjectInstance>
                </ObjectInstance>
            </ObjectInstance>
        </ObjectInstance>
    </ObjectInstance>
</Configuration>
```

The behavior of BAC to apply the above template configuration during configuration sync is explained below:

1. BAC retrieves GetParameterNames RPC data for the object Device.Services.FAPService. with next level option set to true.
2. Since all() option is configured, it uses all of the returned instances for further processing.
3. BAC searches the PLMNID, LAC and RNCID by issuing GetParameterValues RPC for the parameter Device.Services.FAPService.{all()}.CellConfig.UMTS.RAN.NeighborList.IntraFreqCell.{i}.PLMNID, Device.Services.FAPService.{all()}.CellConfig.UMTS.RAN.NeighborList.IntraFreqCell.{i}.LAC and Device.Services.FAPService.{all()}.CellConfig.UMTS.RAN.NeighborList.IntraFreqCell.{i}.RNCID for all the instances.
4. For all the matching instances that have a value of PLMNID equals 123452, LAC equals 65532 and RNCID equals 18, SetParameterValues is performed with the configured RAC value 10.
5. For all the matching instances that have a value of PLMNID equals 123452, LAC equals 65533 and RNCID equals 21, SetParameterValues is performed with the configured RAC value 11.
6. If a matching instance is not found for the configured PLMNID, LAC, RNCID values, a new instance is created using AddObject RPC and the configured values of PLMNID, LAC, RNCID and RAC are applied on the new instance.
7. If there are instances available on the device that does not match the PLMNID, LAC and RNCID, they are deleted using DeleteObject Message.

Example 4-2 Multi-Instance Object Using replace-all-on-change in Configuration Template

The following example illustrates usage of multi-instance object using replace-all-on-change in configuration template:

```
<tc:Template
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tc="urn:com:cisco:bac:common-template"
    xmlns="urn:com:cisco:bac:cwmp-template"
    xsi:schemaLocation="urn:com:cisco:bac:common-template CommonTemplateConstructs.xsd">
    <Configuration templateVersion="3.0">
        <ParameterDictionaries>
            <ParameterDictionary>tr196-cwmp-dictionary-v2.0.xml</ParameterDictionary>
        </ParameterDictionaries>
        <ObjectInstance name="Device">
            <ObjectInstance name="Services">
                <ObjectInstance name="FAPService">
                <ObjectInstance name="{i}" sync-method="discovered" instance="1">
                    <ObjectInstance name="CellConfig">
                        <ObjectInstance name="UMTS">
                            <ObjectInstance name="RAN">
                                    <ObjectInstance name="NeighborList">
                                        <ObjectInstance name="IntraFreqCell"
sync-method="replace-all-on-change" primary-keys="PLMNID,LAC,RNCID">
                                            <ObjectInstance name="{i}">
                                                <Parameter>
                                                    <Name>RAC</Name>
                                                    <Value>10</Value>
                                                </Parameter>
                                                <Parameter>
                                                    <Name>PLMNID</Name>
                                                    <Value>123452</Value>
                                                </Parameter>
                                                <Parameter>
                                                    <Name>LAC</Name>
                                                    <Value>65532</Value>
                                                </Parameter>
                                                <Parameter>
                                                    <Name>RNCID</Name>
                                                    <Value>18</Value>
                                                </Parameter>
                                            </ObjectInstance>
                                            <ObjectInstance name="{i}">
                                                <Parameter>
                                                    <Name>RAC</Name>
                                                    <Value>11</Value>
                                                </Parameter>
                                                <Parameter>
                                                    <Name>PLMNID</Name>
                                                    <Value>123452</Value>
                                                </Parameter>
                                                <Parameter>
                                                    <Name>LAC</Name>
                                                    <Value>65533</Value>
                                                </Parameter>
                                                <Parameter>
                                                    <Name>RNCID</Name>
                                                    <Value>21</Value>
                                                </Parameter>
                                            </ObjectInstance>
                                        </ObjectInstance>
                                    </ObjectInstance>
                                </ObjectInstance>
                            </ObjectInstance>
```

```
                    </ObjectInstance>
                </ObjectInstance>
            </ObjectInstance>
        </ObjectInstance>
    </ObjectInstance>
</Configuration>
</tc:Template>
```

The behavior of BAC to apply the above template configuration during configuration sync is explained below:

1. BAC retrieves GetParameterNames RPC data for the object Device.Services.FAPService. with next level option set to true.
2. Since instance="1" option is configured, it uses instance 1 for further processing.
3. BAC searches the PLMNID, LAC and RNCID by issuing GetParameterValues RPC for the parameter Device.Services.FAPService.1.CellConfig.UMTS.RAN.NeighborList.IntraFreqCell.{i}.PLMNID, Device.Services.FAPService.1.CellConfig.UMTS.RAN.NeighborList.IntraFreqCell.{i}.LAC and Device.Services.FAPService.1.CellConfig.UMTS.RAN.NeighborList.IntraFreqCell.{i}.RNCID for all the instances.
4. For all the matching instances that have a value of PLMNID equals 123452, LAC equals 65532 and RNCID equals 18, SetParameterValues is performed with the configured RAC value 10.
5. For all the matching instances that have a value of PLMNID equals 123452, LAC equals 65533 and RNCID equals 21, SetParameterValues is performed with the configured RAC value 11.
6. The above steps 4 and 5 happen only if all object instances in the template match all the instances in the device.
7. If there are instances available on the device that do not match the PLMNID, LAC and RNCID or if there are object instances defined in the template that do not have matching instance in the device then all the instances in the device are deleted using DeleteObject Message.
8. If all the instances are deleted in the device a new instance is created using AddObject RPC and the configured values of PLMNID, LAC, RNCID and RAC are applied on the new instance. This is done for all the object instances defined in the template.

Example 4-3 Multi-Instance Object Using delete-all in Configuration Template

The following example illustrates usage of multi-instance object using delete-all in configuration template:

```
<Configuration templateVersion="3.0">
    <ParameterDictionaries>
        <ParameterDictionary>tr196-cwmp-dictionary.xml</ParameterDictionary>
    </ParameterDictionaries>
    <ObjectInstance name="Device">
    <ObjectInstance name="Services">
        <ObjectInstance name="FAPService">
            <ObjectInstance sync-method="discovered" instance="last()">
                <ObjectInstance name="REM.UMTS.GSM.Cell" sync-method="delete-all" >
                </ObjectInstance>
            </ObjectInstance>
        </ObjectInstance>
    </ObjectInstance>
</Configuration>
```

The behavior of BAC to push the above template configuration during configuration sync is explained below:

1. BAC discovers the instances in the object Device.Services.FAPService. by issuing GetParameterNames RPC with next level set to true.
2. Since last() method is used, it gets the last instance and discovers the instances in the next level by issuing GetParameterNames RPC for the parameter Device.Services.FAPService.{last()}.REM.UMTS.GSM.Cell.{i}. with next level set to true.
3. BAC performs DeleteObject one by one on all the instances returned.

Example 4-4 Multi-Instance Object Using delete in Pre-requisites of Configuration Template

The following example illustrates usage of multi-instance object using delete in pre-requisites of configuration template:

```
<Configuration templateVersion="3.0">
    <ParameterDictionaries>
        <ParameterDictionary>tr196-cwmp-dictionary.xml</ParameterDictionary>
    </ParameterDictionaries>
    <Prerequisites>
        <MaintenanceWindow>
            <StartTime>00:00:00</StartTime>
            <Duration>15:00</Duration>
        </MaintenanceWindow>
        <Expression>
            <ParameterName>Device.Services.FAPService.{i}.REM.UMTS.GSM.Cell.{i}.BSIC
            </ParameterName>
            <InstanceConfiguration>
                <Instance>
                    <Path>Device.Services.FAPService.{i}.REM.UMTS.GSM.Cell.</Path>
                    <Value>compare(LAC lessThanEquals 65534)</Value>
                </Instance>
                <Instance>
                    <Path>Device.Services.FAPService.</Path>
                    <Value>last()</Value>
                </Instance>
                <MatchCondition>OR</MatchCondition>
            </InstanceConfiguration>
            <Value>18</Value>
            <Operator>matchAllIgnoreCase</Operator>
        </Expression>
    </Prerequisites>
    <ObjectInstance name="Device">
    <ObjectInstance name="Services">
        <ObjectInstance name="FAPService">
            <ObjectInstance sync-method="discovered" instance="last()">
                <ObjectInstance name="REM.UMTS.GSM.Cell" sync-method="delete" instance="2">
                </ObjectInstance>
            </ObjectInstance>
        </ObjectInstance>
    </ObjectInstance>
</Configuration>
```

The behavior of BAC to apply the above template configuration during configuration sync is explained below:

1. During configuration sync execution, BAC validates the time to ensure that the configuration sync falls in the maintenance window
2. Next, BAC validates the expression. Since the expression parameters are configured with multi-instance object parameters, BAC first discovers and identify the instances to evaluate the expression.
3. After the expression is evaluated to true, BAC discovers the instance for Device.Services.FAPService. by using GetParameterNames RPC with next level set to true.
4. To discover the multi-instance object at next level, it issues GetParameterNames RPC for Device.Services.FAPService.{last()}.REM.UMTS.GSM.Cell.
5. Finally, BAC issues DeleteObject Message for the instance in the second index in the returned array.

Example 4-5 Invalid Example-Adding a Template Without Instance Attribute

The following example illustrates adding a template without the instance attribute.

```
<tc:Template
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tc="urn:com:cisco:bac:common-template"
    xmlns="urn:com:cisco:bac:cwmp-template"
    xsi:schemaLocation="urn:com:cisco:bac:common-template CommonTemplateConstructs.xsd">
    <Configuration templateVersion="3.0">
        <ParameterDictionaries>
            <ParameterDictionary>tr196-parameter-dictionary.xml</ParameterDictionary>
        </ParameterDictionaries>
        <ObjectInstance name="Device">
            <ObjectInstance name="Services">
                <ObjectInstance name="FAPService">
                <ObjectInstance name="{i}" sync-method="discovered"
instance="compare(DeviceType equals CWMP1)">
                    <ObjectInstance name="AccessMgmt">
                    <ObjectInstance name="MemberDetail">
                               <ObjectInstance name="{i}" sync-method="discovered">
                         <Parameter>
                             <Name>Enable</Name>
                             <Value>true</Value>
                         </Parameter>
                         <Parameter>
                             <Name>IMSI</Name>
                             <Value>310410268739757</Value>
                         </Parameter>
                        </ObjectInstance>
                    </ObjectInstance>
                    </ObjectInstance>
                </ObjectInstance>
                </ObjectInstance>
            </ObjectInstance>
        </ObjectInstance>
    </Configuration>
</tc:Template>
```

When this template is added through API or admin UI, the following validation error is displayed:

```
Error: Template file validation has failed with the following error.
[Missing instance attribute for the sync-method 'discovered']
```

Firmware Rules Template

Earlier to this release, the expression support in the firmware rules was not flexible enough to handle multi-instance objects in the parameter names. This enhancement is extended to firmware rules template where BAC discovers the multi-instance parameters to evaluate the expression.

You can specify multi-instance parameter name in the expression, and the discover mechanism of these objects can be configured using the newly introduce template element InstanceConfiguration.

Example 4-6 Multi-Instance Object in Expression

The following example illustrates usage of multi-instance object in expression:

```
<Expression>
        <ParameterName>Device.Services.FAPService.{i}.REM.GSM.Cell.{i}.LAC
</ParameterName>
        <InstanceConfiguration>
            <Instance>
                <Path>Device.Services.FAPService</ Path>
                <Value>last()</Value >
            </Instance>
            <Instance>
                <Path>Device.Services.FAPService.{i}.REM.GSM.Cell</ Path>
                <Value>compare(PLMNID equals 123456)</Value >
            </Instance>
            <MatchCondition>OR</MatchCondition>
        </InstanceConfiguration>
        <Value>4660</Value>
        <Operator>match</Operator>
    </Expression>
```

In the above expression the multi-instance object parameter name Device.Services.FAPService.{i}.REM.GSM.Cell.{i}.LAC is used. The InstanceConfiguration discovers the multi-instance object instances Device.Services.FAPService.{i} and Device.Services.FAPService.{i}.REM.GSM.Cell.{i}. The Value tag supports all the available instance discover methods like all(), last(), compare(parameterName operation matchValue) and index. The new tag MatchCondition is available in the instance configuration, which is used to evaluate the multiple expressions only in case there are multiple instances discovered.

Example 4-7 Multi-Instance Object in Firmware Rule Template

The following example illustrates usage of multi-instance object in firmware rule template:

```
<tc:Template
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tc="urn:com:cisco:bac:common-template"
    xmlns="urn:com:cisco:bac:firmware-template"
    xsi:schemaLocation="urn:com:cisco:bac:common-template CommonTemplateConstructs.xsd">

    <FirmwareTemplate templateVersion="3.0">
         <ParameterDictionaries>
             <ParameterDictionary>tr196-cwmp-dictionary-v2.0.xml</ParameterDictionary>
         </ParameterDictionaries>

    <Prerequisites>
        <MaintenanceWindow>
            <StartTime>00:00:00</StartTime>
            <Duration>15:00</Duration>
        </MaintenanceWindow>
```

```
        <Expression>

<ParameterName>Device.Services.FAPService.{i}.REM.UMTS.GSM.Cell.{i}.BSIC</ParameterName>
            <InstanceConfiguration>
                <Instance>
                    <Path>Device.Services.FAPService.{i}.REM.UMTS.GSM.Cell.</Path>
                    <Value>compare(LAC lessThanEquals 65534)</Value>
                </Instance>
                <Instance>
                    <Path>Device.Services.FAPService.</Path>
                    <Value>last()</Value>
                </Instance>
                <MatchCondition>OR</MatchCondition>
            </InstanceConfiguration>
            <Value>18</Value>
            <Operator>matchAllIgnoreCase</Operator>
        </Expression>
        <Expression>

<ParameterName>Device.Services.FAPService.{i}.REM.UMTS.GSM.Cell.{i}.PLMNID</ParameterName>
            <InstanceConfiguration>
                <Instance>
                    <Path>Device.Services.FAPService.</Path>
                    <Value>all()</Value>
                </Instance>
                <Instance>
                    <Path>Device.Services.FAPService.{i}.REM.UMTS.GSM.Cell.</Path>
                    <Value>2</Value>
                </Instance>
                <MatchCondition>OR</MatchCondition>
            </InstanceConfiguration>
            <Value>123451</Value>
            <Value>123452</Value>
            <Operator>matchIgnoreCase</Operator>
        </Expression>
    </Prerequisites>

        <FirmwareRule name="FAPRule">
        <Expression>
            <InformParameterName>Inform.EventCode</InformParameterName>
            <Value>1 BOOT</Value>
            <Value>2 PERIODIC</Value>
            <Operator>match</Operator>
        </Expression>

        <Expression>

<ParameterName>Device.Services.FAPService.{i}.REM.UMTS.GSM.Cell.{i}.BSIC</ParameterName>
            <InstanceConfiguration>
                <Instance>
                    <Path>Device.Services.FAPService.{i}.REM.UMTS.GSM.Cell.</Path>
                    <Value>1</Value>
                </Instance>
                <Instance>
                    <Path>Device.Services.FAPService.</Path>
                    <Value>compare(DeviceType equalsIgnoreCase fap)</Value>
                </Instance>
                <MatchCondition>OR</MatchCondition>
            </InstanceConfiguration>
            <Value>18</Value>
            <Operator>matchAllIgnoreCase</Operator>
        </Expression>

        <InternalFirmwareFile>
```

```
            <FileName>sample-firmware-image.bin</FileName>
            <DeliveryTransport>service http 1</DeliveryTransport>
        </InternalFirmwareFile>
         </FirmwareRule>
    </FirmwareTemplate>
</tc:Template>
```

Parameter Discovery

Multi-instance object parameters can now be specified in the data discovery. This support is extended for preregistered and unregistered devices.

Preregistered Devices

For preregistered devices, the multi-instance object parameters can be configured in the parameter `IPDeviceKeys.CWMP_CUSTOM_DISCOVER_PARAMETERS` as in the example below:

```
/IPDevice/custom/discover/parameters= Device.Services.FAPService.{i}.REM.GSM.Cell.{i}.LAC,
Device.Services.FAPService.{i}.REM.GSM.Cell.1.PLMNID
```

Unregistered Devices

For unregistered devices, the multi-instance object parameters can be configured in the parameter `ServerDefaultsKeys.CWMP_UNKNOWN_CPE_PARAMETERS` as in the example below:

```
/server/acs/unknown/parametersToRetrieve=
Device.Services.FAPService.{i}.REM.GSM.Cell.{i}.LAC,
Device.Services.FAPService.{i}.REM.GSM.Cell.1.PLMNID
```

Display Live Data Operation

The device operation enables you to view live device parameter values. The parameters retrieved from the device are selected from a Parameter List file, which is an XML file that details all the parameters.

If the multi-instance object parameters are configured in the Parameter List XML file, the device operation retrieves the parameter values from all the available instances in the device.

The following example illustrates the usage of multi-instance object support to retrieve live device data.

```
<ParameterList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ParameterListSchema.xsd">
       <ParameterDictionaries>
            <ParameterDictionary>tr196-cwmp-dictionary-v2.0.xml</ParameterDictionary>
       </ParameterDictionaries>
       <Parameter>
            <Name> Device.Services.FAPService.{i}.REM.GSM.Cell.{i}.LAC</Name>
       </Parameter>
   <Parameter>
            <Name> Device.Services.FAPService.{i}.REM.GSM.Cell.1.PLMNID </Name>
       </Parameter>
    <Parameter>
            <Name>Device.Services.FAPService.{i}.REM.UMTS.GSM.CellNumberOfEntries</Name>
       </Parameter>
</ParameterList>
```

Instance Sorting

In all the operations, when selecting the instances based on the instance configuration, you sort the list of instances returned in GetParameterNames RPC call. The discovered object instances can be sorted when the property `ProvGroupKeys.ENABLE_INSTANCE_SORTING` (/provGroup/enable/instanceSorting) is enabled.

This property can be configured at the Provisioning Group level in the object hierarchy. For details, see Provisioning Group Configuration Workflow, page 3-8.

Instruction Generation and Processing

Instruction generation is the process of generating specific instruction sets for CWMP devices. By using technology extensions, through which device technologies are incorporated into Cisco BAC, device details are combined with provisioning rules to produce instruction sets appropriate to the CPE. These instructions are then forwarded to the DPEs in the device's provisioning group and cached there.

The scriptable extensions are also supported in Cisco BAC. The extension points are defined in DPE for running the java based extension scripts. These scripts are executed in DPE, whenever the device establishes connection with the DPE. The extension scripts are added in the RDU database and forwarded to the DPE cache. For more information on scriptable extension service, see Scripting Framework, page 18-1.

When a device is activated in a Cisco BAC deployment, it initiates contact with the Cisco BAC server. After contact is established, the device's preconfigured policy, based on configuration templates or firmware rules templates associated with the device, determine the management actions undertaken by the DPE.

This preconfigured policy determines the device's level of service, also known as Class of Service. Device configurations can include customer-required provisioning information, such as authentication information, periodic inform rate, and Class of Service. This authoritative provisioning information for the device is forwarded to DPEs from the RDU as device configuration instructions.

Instructions are logical operations which the DPE autoconfiguration server (ACS) performs for a specific device. The instructions may map directly into a CWMP remote procedure call (RPC), for example, GetParameterValues; or, they may combine additional logic with multiple CWMP RPCs, such as firmware rules.

IGS could be controlled using the new API property, *ApiCommandKeys.IGS_ENABLE*. This property determines whether IGS should be triggered or not. The API command uses this property to control the regeneration of device configuration being affected by other API commands.

When this property is set to False in the map of associated properties of an API command, the command skips the regeneration of the configuration process and it returns immediately with a warning message. The API commands affected by this property are:

- *replaceFile*
- *changeClassOfServiceProperties*
- *changeProvGroupProperties*
- *changeNodeProperties*
- *changeDefaults*

For the list of other API commands see Non-concurrent Commands, page 19-2.

The RDU requests that instructions be processed, by passing “InstructionRecords” to the DPE. The DPE server then converts these “InstructionRecords” to “Instructions”, and returns the results to the RDU as “InstructionResponseRecords”.

Cisco BAC generates instructions through:

- The Instruction Generation Extension—Generates “InstructionRecords” for a single device.
- The Instruction Generation Service—Generates “InstructionRecords” for more than one device.

  You can access statistics from the Instruction Generation Service from the administrator user interface at **Servers > RDU > View Regional Distribution Unit Details**.

Among the various instructions that the RDU generates are:

- Data Synchronization Instruction (DataSyncRecord)—Keeps the RDU up-to-date on various CPE parameters, such as the software version and the model name. These parameters may, in turn, be used in generating other instructions, such as configuration instructions that are specific to a given device type.

  Some parameters are checked on every connection, while others are checked only when a change in the firmware version occurs. For details, see Discovering Data from Devices, page 12-12.
- Routable IP Address Instruction (RoutableIPAddressRecord)—Discovers if a particular device is reachable, enabling the DPE to create a TCP connection with a device in order to service a connection request. The instruction retrieves the WAN IP address for the device, PPP or DHCP, and compares it with the source IP address. If the IP addresses are different, the instruction updates the RDU.
- Firmware Rules Instruction (FirmwareRulesRecord)—Determines the firmware image to be downloaded to a device. The firmware image files are associated to groups of devices by a firmware rules template.

  Cisco BAC uses the rules in the associated template to evaluate the firmware to be downloaded to the CPE. This instruction takes effect only if the device has been associated with a firmware rules template.
- Configuration Synchronization Instruction (ConfigSyncRecord)—Triggers a synchronization of the CPE configuration that is stored in the DPE cache. This instruction comes into effect only if the device has been associated with a configuration template. The process of configuration synchronization is explained in the subsequent section.

When the Signed Configuration feature is enabled, the RDU parses the ToBeSigned tag specified in the configuration template and sets the corresponding value on the CWMP parameter object.

By default, the ToBeSigned flag on the CWMP parameter object is set to False. For more information on Signed Configuration, see Signed Configuration for Devices, page 13-20.

Device Configuration Synchronization

During the process of CPE configuration synchronization, a device’s configuration is automatically synchronized based on the configuration template associated with the device’s Class of Service object. The process of synchronizing the CPE configuration according to the ConfigSync instruction stored in the DPE for this device is called Configuration Synchronization.

As part of this process, the DPE configures all parameters values and attributes found in the configuration template associated with a device through its Class of Service, so that:

- Notifications reflect those configured in the configuration template. For more information on Notifications, see Notification, page 5-8.

- Access control for all parameters reflects that configured in the configuration template. For more information on access control, see Access Control, page 5-9.

CPE configuration is associated with a unique configuration key. This configuration key is saved in the DPE database, and is used as the *ParameterKey* parameter in RPCs that are forwarded to the CPE.

Every time the CPE establishes a connection with the DPE, the device reports the value for the *ParameterKey*—in the form of a configuration revision number—by using the Inform message that the device forwards to the DPE. The DPE compares this value with the one in its cache for the particular device. A mismatch of the values, triggers the DPE and the device synchronization process.

During the process of configuration synchronization:

1. The DPE receives a *ParameterKey* from the device. If the value of this *ParameterKey* matches the one stored in the DPE, no synchronization is initiated. If the *ParameterKey* values differs, the synchronization process continues.
2. If access control is set in its configuration, the DPE sets the *AccessList* parameter to ACS-only. The access control feature is, by default, enabled. For more information on access control, see Access Control, page 5-9.
3. If you enable the notification feature, the DPE sets notification attributes as specified in the device configuration. Notifications are, by default, enabled. For more information on notifications, see Notification, page 5-8.
4. After the DPE configures the parameter values on the device according to the template, it sets a new configuration revision number in the *ParameterKey* argument. This revision number is used to determine if the device configuration is synchronized the next time the device and the DPE establish a connection.

   If the device connection with the DPE times out during the synchronization process, the CPE attempts to reconnect to the DPE.

   In this scenario, the value of the *ParameterKey* in the Inform message remains the same, because only a successful synchronization process changes the *ParameterKey* value. When the CPE reconnects to the DPE, the DPE initiates another round of synchronization with the original *ParameterKey* value.
5. The synchronization process ends with the DPE forwarding the new value for the *ParameterKey* attribute in its last update to the CPE.

In some situations, you must update the device even if the *ParameterKey* on the DPE matches the one on the device.

To force a configuration synchronization:

Step 1 From the **Devices** page, locate the device whose configuration you want to synchronize.

Step 2 Click the **Operations** icon () corresponding to the device.

The Device Operations page appears.

Step 3 From the drop-down list under Perform Device Operation, select Force Configuration Synchronization.

Step 4 Click **Submit**.

The device configuration is synchronized with the DPE.

Device Deployment in Cisco BAC

A Cisco BAC deployment is divided into provisioning groups, with each provisioning group responsible only for a subset of the devices. All services provided by the provisioning group are implemented to provide fault tolerance.

Note A key principle of device management is that the RDU does not directly communicate with devices. All device interactions are delegated to DPEs in the provisioning group to which the device belongs.

Cisco BAC provides two device deployment options, which can also be used in combination:

- Preregistered—The device record is added to the RDU before the device makes initial contact with the DPE, also known as the ACS.
- Unregistered—The device makes contact with the DPE, before the device record is added to the RDU.

Preregistered Devices

In this scenario, device data is preprovisioned into Cisco BAC, and the device is associated with a specific Class of Service. The Class of Service can correspond to a service that the subscriber registered for or a default configuration.

A preregistered device is preconfigured with certain parameters specific to the service provider. These parameters are typically "burnt-in" as factory defaults.

Note If you reset the device to factory defaults, the settings on the device revert to the preburnt configuration, and the device may go through the reconfiguration process.

Device data is preregistered in Cisco BAC. This is typically done through the API; alternatively, it can be done using the administrator user interface.

Preconfiguration involves three important issues:

- The device must be able to establish network connectivity. For DSL devices, this typically involves using auto-detection of ATM PVC and using PPP for authentication. The IP address is obtained using PPP or using DHCP. Other devices typically use an existing internet connection and local DHCP for address assignment.
- CPE must contact the configuration servers of the appropriate service provider; in other words, the CPE must know the ACS URL. The ACS URL can be preburnt into the device (assigned) or discovered using DHCP from the WAN side.
- The service provider must be able to associate the CPE with a specific subscriber. This process is typically accomplished by the Operations Support Systems (OSS) application responsible for subscriber registration. Cisco BAC is updated with appropriate data to provision device configuration.

Unregistered Devices

In this scenario, no device data is prepopulated into Cisco BAC. Device data is added to Cisco BAC only when the device first contacts a Cisco BAC server.

Cisco BAC allows unregistered devices (with no preconfigured parameters) to appear on a network and gain default access. However, the lack of support for preregistering device data into Cisco BAC restricts authentication options for unregistered devices, to using mechanisms based on certificates as opposed to shared secrets.

The lack of preregistered data also means that Cisco BAC has to dynamically classify the devices and determine the default configuration of a device.

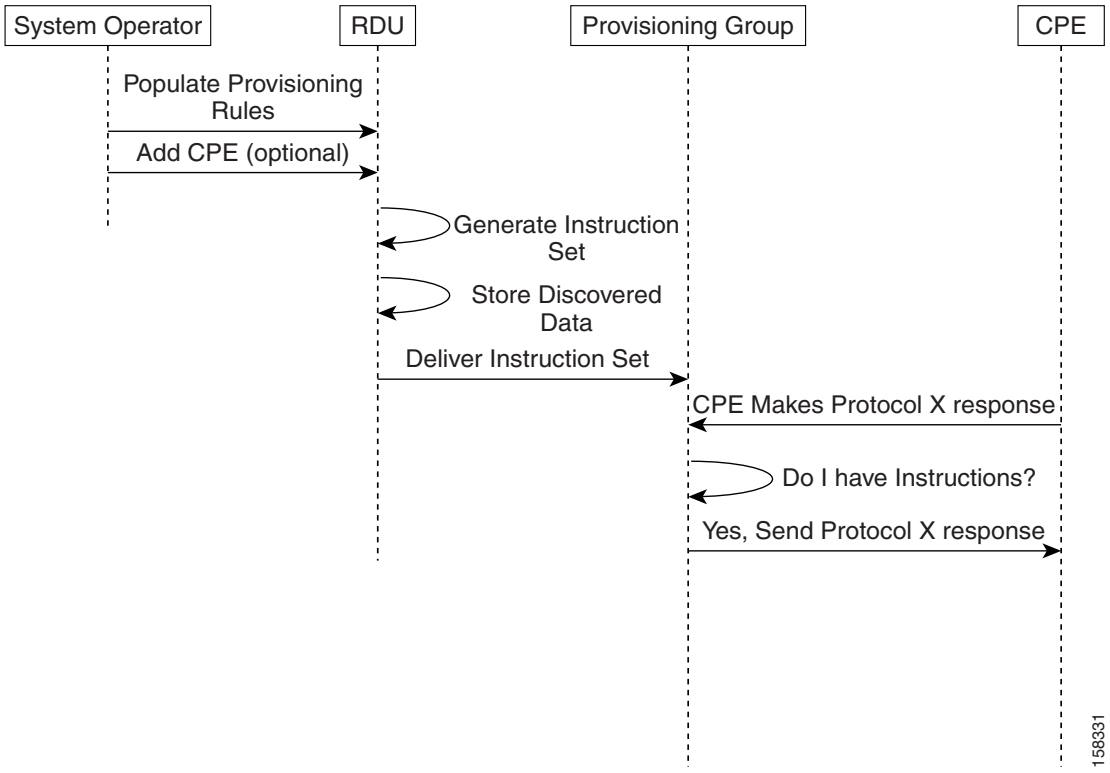
Note With no preregistered device data available, the chances of a Denial of Service attack increase, as unknown devices are not authenticated.

Initial Provisioning Flows

This section describes the configuration workflow for a device, which differs based on whether a device is preregistered or unregistered.

Figure 4-2 shows a common initial configuration flow.

Figure 4-2 Initial CPE Configuration Workflow



For Preregistered Devices

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

b. The preregistered device finds its provisioning group by contacting the Cisco BAC server at a preconfigured URL, and initiates autoprovisioning with provisioning group servers (DPEs).

c. The RDU generates instructions appropriate for the device. The resulting device instructions direct DPE responses to various CPE protocol events, such as a TR-069 Inform and an HTTP file request.

d. The device instruction set is forwarded to the DPE and cached there. Now, the DPE is programmed to handle subsequent CPE protocol interactions for this device autonomously from the RDU. After the device is added to the network and a configuration is generated for the device, the device boots to allow the DPE to begin its interactions with the preregistered device.

e. During interactions with the device, additional information can be discovered and forwarded to the RDU. In this case, the RDU may decide to generate new instructions and forward them to all DPEs.

For Unregistered Devices

a. From the Cisco BAC API, the RDU is populated with specifically defined configurations and rules for various device types.

b. During bootup, when the DPE receives a device request, it performs a local search for instructions cached for the specific CPE. Because the CPE has never previously contacted the DPE and because device data was not preregistered into Cisco BAC, no instructions are found.

   The DPE then packs all relevant CPE information into an “instruction set” generation request, and forwards the request to the RDU. At the same time, the device request is rejected, forcing the device to retry. Also, the device receives the configured SOAP request/response extensions from DPE cache and the extensions are executed for the device. Unknown device extensions are configured in CWMP defaults and are sent to DPE cache. For more information on configuring extensions for unknown devices, see Configuring Extensions for Unknown Devices, page 18-4.

c. The RDU generates instructions appropriate for the CPE and distributes it to all DPEs within the device’s provisioning group. The resulting CPE instructions direct DPE responses to various CPE protocol events, such as a TR-069 Inform, and an HTTP file request.

d. The CPE is now registered and the details are stored in the RDU. The RDU captures the CPE registration time. The property to retrieve the registered time is /IPDevice/registeredTime. This property value can be obtained through the API IPDevice.getDetails() call.

e. The device instruction set is delivered to the DPE and cached there. Now, the DPE is programmed to handle all subsequent CPE protocol interactions for this device autonomously from the RDU.

   The following parameters are discovered by Cisco BAC for unknown devices:

   - Whether the device IP address is routable or NAT’ed.
   - Inform.DeviceId.Manufacturer.
   - Inform.DeviceId.ManufacturerOUI.
   - Inform.DeviceId.ProductClass.
   - InternetGatewayDevice.DeviceInfo.HardwareVersion.
   - InternetGatewayDevice.DeviceInfo.SoftwareVersion.
   - InternetGatewayDevice.DeviceInfo.ModelName.

- InternetGatewayDevice.ManagementServer.ParameterKey.

Note You can change the default list of discovered parameters. See Discovering Data from Devices, page 12-12.

f. The device then reconnects, and receives the configuration instructions generated for it from the RDU and cached at the DPE.

Assigning Devices to Provisioning Groups

Devices can be assigned to a provisioning group in three ways: explicitly, automatically, or using a combination of both.

Explicit Assignment

You can explicitly assign a device to a provisioning group. After devices appear in the default provisioning group, the provisioning system may, using the API, assign the device to a new provisioning group. On next contact with the device, Cisco BAC redirects the device.

To set the device to contact the assigned provisioning group, change the URL of the Cisco BAC server to the URL of the provisioning group URL. The Cisco BAC server URL is stored, and from then on, the device contacts Cisco BAC at the new address.

To move the device from one provisioning group to another, change the home provisioning group of the device using the API or the administrator user interface. Each provisioning group has a URL associated with it. To facilitate the move, on next contact, the ACS URL on the device is changed to the new provisioning group URL.

Automatic Membership

If a device has not been explicitly assigned to a provisioning group, the device stays in the provisioning group in which it is brought up. This allows a network-directed assignment of CPE to the provisioning groups. You can use the automatic membership feature for roaming devices, enabling the local provisioning group to service these devices when they are moved.

When a device appears in a new provisioning group, it is automatically assigned to the new provisioning group, and the device data is purged from the old provisioning group. This process involves communicating with the RDU, which, in turn, updates the DPEs in the old and new provisioning groups. This is an expensive process; therefore, take care to prevent a large number of device migrations.

Automatic assignment of a device to a provisioning group works only if the DPEs are configured to allow unknown (unregistered) devices that do not show up in any provisioning group.

If the devices do show up in another provisioning group and the provisioning group is configured to allow access for unknown devices, Cisco BAC automatically assigns the device to the provisioning group.

For details on how to configure access for unknown devices, see the *Cisco Broadband Access Center 3.8 DPE CLI Reference*.

Combined Approach

You can explicitly assign devices and allow automatic membership of devices to a provisioning group. For example, a generic unregistered device that appears on the network in a provisioning group is automatically assigned to it. Subsequently, the OSS explicitly assigns the device to another provisioning group by using the API.

Device Diagnostics

CWMP supports device troubleshooting and diagnostics features that you use to focus on a single device and collect diagnostics information for further analysis. This feature enables you to query devices for any data, including:

- Configuration
- Live statistics
- Fault indications
- Log file
- Diagnostics results

Device diagnostics is made possible in Cisco BAC through a set of operations that can be run on the device. These include:

- Reboot—Reboots the device. This reboot is primarily intended for diagnostic purposes.
- Request Connection—Initiates a connection request with Cisco BAC.
- Factory Reset—Resets a preregistered device settings to its original factory settings, to before a subscriber-specific configuration was burnt in.
- Display Live Data—Views device parameters directly from a device. You can define the parameters you want to appear.
- Ping Diagnostic—Enables you to perform an IP ping diagnostics test from the device to any host.
- Force Firmware Upgrade—Forces a CPE to update its firmware.
- Force Configuration Synchronization—Enables you to force an individual CPE to synchronize its configuration.

For details on performing these device operations, see Performing Operations on Devices, page 16-14.

Cisco BAC also provides the following features to help troubleshooting:

- Device History—Provides a detailed history of significant events that occur in a device provisioning lifecycle. See Device History, page 8-1.
- Device Faults—Detects devices with recurring faults, which can cause bottlenecks and affect network performance. See Device Faults, page 8-6.
- Device Troubleshooting—Provides detailed records of device interactions with Cisco BAC servers for a set of devices that are designated for such troubleshooting. See Device Troubleshooting, page 8-9.
- Performance Statistics—Provides detailed performance statistics that are related to system performance across major components. It also provides an analysis of the statistical data to aid troubleshooting. See Monitoring Performance Statistics, page 11-14.

Configuring SNMP Trap for CPEs

The fault management module of Cisco BAC facilitates CPEs to raise alarms and send events when any fault occurs. The fault management module can be enhanced to support the SNMP trap facility. With the SNMP trap facility, DPE receives the events from the CPEs and converts them into SNMP traps. DPE further sends these SNMP traps to the trap receiver.

Cisco BAC also supports integration with Cisco Prime Central application to receive EPM MIB OIDs traps in the Prime Central alarm browser. Each RDU and DPE component is registered as individual domain manager with Prime Central.

The following components are configured in Cisco BAC for SNMP Trap facility:

- RDU, see Configuration at RDU Level, page 4-24
- DPE and Trap Receiver, see Configuration at DPE Level, page 4-25
- Device, see Configuration at Device Level, page 4-26

Configuration at RDU Level

To configure RDU for SNMP trap facility:

Step 1 Create or update the BAC configuration template to support expedited and queued alarms.

The following parameters must be set in the configuration template:

- For expedited alarms:
  - FAPService.{i}.FaultMgmt.SupportedAlarm.{i}.ReportingMechanism=0 {Expedited}
  - FAPService.{i}.FaultMgmt.ExpeditedEvent=Active
- For Queued alarms
  - FAPService.{i}.FaultMgmt.SupportedAlarm.{i}.ReportingMechanism=1 {Queued}
  - FAPService.{i}.FaultMgmt.QueuedEvent=Active

For information on how to author configuration template, see Authoring Configuration Templates, page 5-14.

Step 2 Add this configuration template to the RDU database:

a. Log into the Cisco BAC admin UI.

b. Choose **Configuration** on the primary navigation bar.

c. Choose **Files** on the secondary navigation bar. The View Files page appears.

d. Click **Add**.

The Add Files page appears.

e. Choose **Configuration Template** from the File Type drop-down list.

f. Browse for the Source File Name.

g. Add the configuration template name in the File Name field.

h. Click **Submit**.

Step 3 Associate this configuration template with the required Class of Service. For details, see Configuring the Class of Service, page 17-1.

Configuration at DPE Level

To configure DPE for SNMP trap facility:

Step 1 Configure the SNMP alarm property values in the DPE. The following alarm property values must be configured in the DPE to support SNMP trap:

- AlarmIdentifier
- AlarmRaisedTime
- AlarmChangedTime
- ManagedObjectInstance
- EventType
- ProbableCause
- SpecificProblem
- PerceivedSeverity
- AdditionalText
- AdditionalInformation

These SNMP alarm property values can be configured in dpe.properties file through DPE CLI. For details on DPE CLI, see *Cisco Broadband Access Center 3.8 DPE CLI Reference*.

Step 2 Customize the **/cpeAlarm/alarmParams** property in dpe.properties file to include SNMP alarm property values. The SNMP alarm property values configured for this property are seen in the SNMP traps.

The default setting for **/cpeAlarm/alarmParams** property is:

/cpeAlarm/alarmParams=EventTime,AlarmIdentifier,EventType,ProbableCause,PerceivedSeverity

To integrate BAC with Prime Central (to receive the traps in Prime Central Alarm Browser), the setting should be:

/cpeAlarm/alarmParams=EventTime,AlarmIdentifier,NotificationType,ManagedObjectInstance, EventType,ProbableCause,SpecificProblem,PerceivedSeverity,AdditionalText,AdditionalInformation

To add parameters that will be part of SNMP trap for 'value change' and 'boot notification' of FAP, the setting should be:

/cpeAlarm/alarmParams=AlarmIdentifier,EventTime,ManagedObjectInstance,ManagedObjectAddress, ManagedObjectAddressType,EventType,ProbableCause,SpecificProblem,PerceivedSeverity,Perceived SeverityIndex,AdditionalText,AdditionalInformation,NotificationType,AlarmType

Step 3 Configure the trap receiver to receive the SNMP traps from the DPE. The trap receiver is added as SNMP host with the required community and listening port. For information on how to add SNMP host and community, see Using the snmpAgentCfgUtil.sh Tool, page 11-6.

Note You can configure multiple trap receivers to receive SNMP traps from the CPEs.

Prime Central also needs to be configured as one of the trap receiver for Cisco BAC. This is done by running the Prime Integrator script with the DPE component installed, or by manually adding Prime Central as a trap listener in DPE. For more details, see the Cisco Broadband Access Center Installation Guide 3.9.

Configuration at Device Level

To configure the device for SNMP trap facility:

Step 1 Modify the device property hierarchy to include the following properties:

- /IPDevice/cpeAlarm/prefixEID
- /IPDevice/cpeAlarm/prefixFaultMOI
- /IPDevice/cpeAlarm/prefixDeviceProp

Table 4-3 provides the significance of these properties along with the possible and default values.

Table 4-3 Properties - SNMP Trap Facility

Property	Possible values	Default Values	Description
/IPDevice/cpeAlarm/prefixEID	True/False	False	If set to true, then EID will be prefixed with ManagedObject Instance and AlarmIdentifier.
/IPDevice/cpeAlarm/ prefixFaultMOI	True/False	False	If not set, the default value will be false. If set to false, then ManagedObjectInstance will not be prefixed with any property. If set to true, ManagedObjectInstance will be prefixed with the value defined for /IPDevice/cpeAlarm/ prefixDeviceProp.
/IPDevice/cpeAlarm/ prefixDeviceProp	Any property in the device hierarchy, which may be custom or built-in	FC-DN-PREFIX	The value of property defined in */IPDevice/cpeAlarm/prefixDeviceProp* will be prefixed with ManagedObjectInstance

For details on how to add these properties in the device property hierarchy, see Modifying Device Records, page 16-11.

Step 2 For the DPE to process both queued and expedited events in “4 Value Change” Inform from FAP and convert them into traps, set the value of the custom property FC-ENABLE-CPE-ALARM to *true*.

Step 3 For Cisco BAC to send a Boot Notification alarm to the trap receiver whenever it receives Boot/Bootstrap ("1 BOOT" / "0 BOOTSTRAP") event from the FAP, set the value of the custom property FC-ENABLE-BOOT-NOTIFY to *true*. This helps the trap receiver to separate the event and automatically clear all outstanding alarms of an FAP whenever the FAP reboots.

Step 4 Configure these properties (explained in Step 1, Step 2, and Step 3) to be available in **/IPDevice/properties/available/pg.**

Step 5 Perform a force synchronization of the device with the DPE to update the device configuration. For details, see Forcing a Configuration Synchronization, page 16-17.

Step 6 Reboot the device and verify if the trap receiver receives the SNMP traps from the DPE.

The SNMP trap consists of the alarm property values. Table 4-4 describes the OIDs (Object Identifiers) that are used in device MIB for these SNMP alarm property values.

Table 4-4 OID mapping

OID	.FaultMgmt.ExpeditedEvent.{i}.	.FaultMgmt.QueuedEvent.{i}.
cenAlarmInstanceID	AlarmIdentifier For Boot Notification of FAP, this is the Device-Id of the FAP.	AlarmIdentifier For Boot Notification of FAP, this is the Device-Id of the FAP.
cenAlarmTimestamp	EventTime For Boot Notification of FAP, this is the time when DPE receives the "1 BOOT" information.	EventTime For Boot Notification of FAP, this is the time when DPE receives the "1 BOOT" information.
cenAlarmManagedObject-Class	ManagedObjectInstance This value also depends on the value set for the property, /IPDevice/cpeAlarm/prefixFaultMOI, wherein it will be prefixed with another value. For Boot Notification and Value Change of FAP, the ManagedObjectInstance contains the DN Prefix, which includes properties like Device ID, Owner ID, Enterprise ID, and so on (which is configurable using the custom property, FC-DN-PREFIX).	ManagedObjectInstance This value also depends on the value set for the property, /IPDevice/cpeAlarm/prefixFaultMOI, wherein it will be prefixed with another value. For Boot Notification and Value Change of FAP, the ManagedObjectInstance contains the DN Prefix, which includes properties like Device ID, Owner ID, Enterprise ID, and so on (which is configurable using the custom property, FC-DN-PREFIX).
cenAlarmManagedObject-Address	For Boot Notification and Value Change of FAP, this is the Device-Id of the FAP.	For Boot Notification and Value Change of FAP, this is the Device-Id of the FAP.
cenAlarmManagedObject-AddressType	For Boot Notification and Value Change of FAP, this is “DNS”.	For Boot Notification and Value Change of FAP, this is “DNS”.
cenAlarmDescription	EventType For Boot Notification of FAP, this is “BootNotification”.	EventType For Boot Notification of FAP, this is “BootNotification”.
cenUserMessage1	ProbableCause	ProbableCause
cenUserMessage2	DMType=bac,<SpecificProblem>	DMType=bac,<SpecificProblem>
cenAlarmSeverity/cenAlarmSeverityDefinition	PerceivedSeverity[1] For Boot Notification and Value Change of FAP, this is “5”.	PerceivedSeverity[1] For Boot Notification and Value Change of FAP, this is “5”.
cenUserMessage3	AdditionalInformation	AdditionalInformation
cenAlarmStatusDefinition	NotificationType	NotificationType
cenAlarmType	unknown(1), direct(2), indirect(3), mixed(4)	unknown(1), direct(2), indirect(3), mixed(4)
cenAlarmServerAddress	DPE address	DPE address

1. The value for PerceivedSeverity is the index between 0-5, and the mapping for severity definition is "Critical", "Major", "Minor","Warning", "Indeterminate",or "Cleared" as per the index value 0-5.