Datacom Network Open Programmability V100R020C10

Development Guide

Issue	01
Date	2021-02-05

HUAWEI

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: https://www.huawei.com

Email: support@huawei.com

About This Document

Overview

This document describes how to perform secondary development, such as device interconnection and fast service construction, based on the open programmability function provided by NCE.

Intended Audience

This document is intended for:

- System administrators
- Maintenance engineers
- Technical support engineers

Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
DANGER	Indicates a hazard with a high level of risk which, if not avoided, will result in death or serious injury.
WARNING	Indicates a hazard with a medium level of risk which, if not avoided, could result in death or serious injury.
CAUTION	Indicates a hazard with a low level of risk which, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a potentially hazardous situation which, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated results. **NOTICE** is used to address practices not related to personal injury.

Symbol	Description
NOTE	Supplements the important information in the main text. Note is used to address information not related to personal injury, equipment damage, or environment deterioration.

GUI Element Reference Conventions

The GUI element reference formatting that may be found in this document are defined as follows.

Format	Description
""	Buttons, menus, parameters, tabs, windows, and dialog titles are in **boldface**. For example, click **OK**.
>	Multi-level menus are in **boldface** and separated by the ">" signs. For example, choose **File** > **Create** > **Folder**.

Command Formatting Conventions

The command formatting that may be found in this document are defined as follows.

Format	Description
Boldface	Command keywords which must be reserved exactly are in **boldface**.
Italic	Command parameters which must be replaced by specific values in an actual command, are in *italics*.
[ ]	Items (keywords or arguments) in brackets [ ] are optional.
{x | y |...}	Indicates that one option is selected from two or more options.
[x | y | ...]	Indicates that one or no option is selected from two or more options.
{ x | y | ... } *	Indicates that multiple options are selected from two or more options. At least one option must be selected, and at most all options can be selected.
[x | y | ...] *	Indicates that multiple options are selected or none is selected from two or more options.

Change History

Issue	Date	Description
02	2021-07-30	This issue is the second official release.
01	2021-02-28	This issue is the first official release.

Contents

1 Basic Knowledge

This chapter describes the basic knowledge required for open programming, including NETCONF, YANG, Jinja2, Python, and the software package structure required for programming.

1.1 NETCONF

Network Configuration Protocol (NETCONF) is an IETF-defined network device management protocol, similar to SNMP. It provides mechanisms to install, manipulate, and delete the configurations of network devices, and query configurations, status, and statistics. NETCONF operations are realized on the Remote Procedure Call (RPC) layer based on Extensible Markup Language (XML) data encoding. The OPS uses NETCONF to communicate with NEs. This section briefly introduces NETCONF based on RFC 6241.

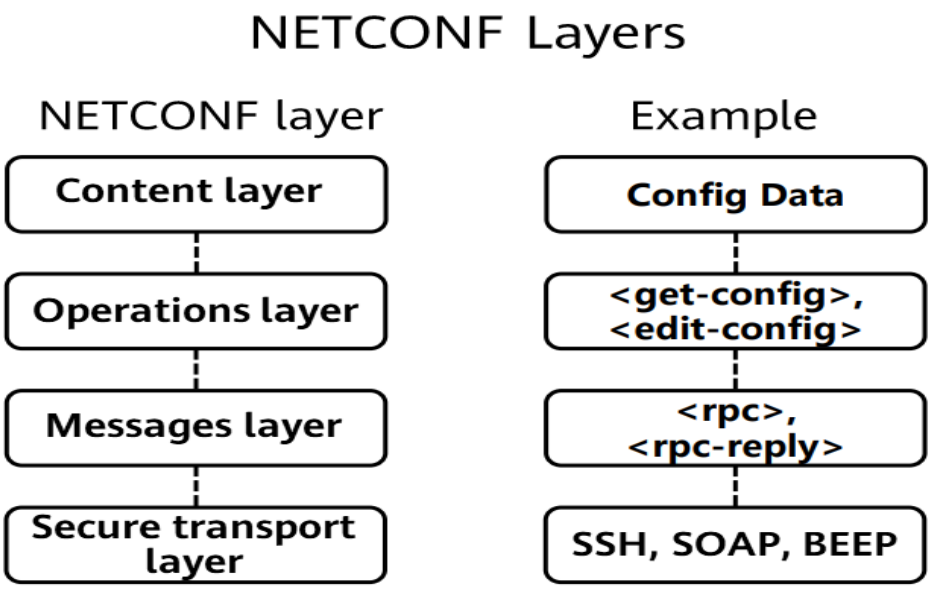
Basic Concepts

- Configuration data: a set of writable data required to transform a system from its initial default state to its current state.
- State data: additional data on a system that is not configuration data, such as read-only status information and collected statistics.
- Protocol operation: a specific RPC used in NETCONF.
- RPC: realized by exchanging <rpc> and <rpc-reply> messages.
- Server: performs protocol operations invoked by a client. In addition, a server can send notifications to a client.
- Client: invokes protocol operations on a server. In addition, a client can subscribe to receive notifications from a server.

For more NETCONF-related concepts, see **Terminology** in RFC 6241.

Protocol Layers

Figure 1-1 NETCONF protocol layers



- Content layer: consists of configuration data and notification data.
- Operations layer: defines a set of XML-encoded operations for realizing RPC operations.
- Messages layer: provides a simple framing mechanism based on transport protocol requirements for encoding RPC messages and notifications.
- Secure transport layer: provides a communication path between a client and a server. Hierarchical NETCONF messages are implemented based on communication paths that meet specific conditions.

For more information about the requirements for data exchange and transport protocols, see "Protocol Overview", "Capabilities", "Separation of Configuration and State Data", and "Transport Protocol Requirements" in RFC 6241.

RPC Model

NETCONF uses an RPC-based communication model. The <rpc> and <rpc-reply> elements are used to transmit the data independent of transport protocols in NETCONF requests and responses sent between NETCONF peers.

- <rpc>

  Encapsulates RPC requests sent from clients to servers. The following is an example.

```
<rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <some-method>
       <!-- method parameters here... -->
   </some-method>
</rpc>
```

  The <rpc> element has a mandatory attribute **message-id**, which is a character string chosen by the sender of the RPC request. The request receiver

does not decode this string but simply uses it as the **message-id** attribute in the resulting <rpc-reply> message.

- <rpc-reply>

  Encapsulates RPC responses returned by servers to clients. The following is an example.

  ```
  <rpc-reply message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <data>
      <!-- contents here... -->
    </data>
  </rpc-reply>
  ```

  The <rpc-reply> element has a mandatory attribute **message-id**. The value of this attribute must be the same as that in the <rpc> element carried in the corresponding request. For other attributes carried in the RPC request, the receiver must return them unchanged in the <rpc-reply> element.

- <rpc-error>

  An <rpc-error> element is sent in an <rpc-reply> element if an error occurs during the processing of an <rpc> element. The following is an example.

  ```
  <rpc-reply
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <rpc-error>
      <error-type>rpc</error-type>
      <error-tag>missing-attribute</error-tag>
      <error-severity>error</error-severity>
      <error-info>
        <bad-attribute>message-id</bad-attribute>
        <bad-element>rpc</bad-element>
      </error-info>
    </rpc-error>
  </rpc-reply>
  ```

  In the preceding example, the <rpc-reply> element does not carry the **message-id** attribute. This is because the RPC request received by the local end does not carry the **message-id** attribute. In this case, the local end directly returns an <rpc-reply> element without this attribute. Note that only in this case is it acceptable for the NETCONF peer to omit the **message-id** attribute in the <rpc-reply> element.

- <ok>

  An <ok> element is sent in an <rpc-reply> element if no errors or warnings occurred during the processing of an <rpc> element. The following is an example.

  ```
  <rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/>
  </rpc-reply>
  ```

  For more information about RPC models, see "RPC Model" in RFC 6241.

RPC Operations

NETCONF provides a set of underlying operations to manage device configurations and retrieve device status information. The base protocol provides operations to retrieve, configure, copy, and delete configuration datastores. Additional operations are provided based on the capabilities advertised by devices.

The base protocol provides the following protocol operations:

- get: obtains the state data and configuration data of the running datastore.
- get-config: obtains the configuration data of a datastore.
- edit-config: modifies, adds, or deletes configuration data.
- copy-config: replaces all data in a datastore.
- delete-config: deletes all data from a datastore.
- lock: locks a datastore.
- unlock: unlocks a datastore.
- close-session: terminates the local NETCONF session.
- kill-session: forces the termination of a NETCONF session.

Here is an example.

In this RPC request, the <get-config> operation is used to request the content of the <users> node in the running configuration from the server.

```
<rpc message-id="101"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-config>
      <source>
         <running/>
      </source>
      <filter type="subtree">
         <top xmlns="http://example.com/schema/1.2/config">
            <users/>
         </top>
      </filter>
   </get-config>
</rpc>
```

For more information about RPC operations, see "Protocol Operations" in RFC 6241.

1.2 YANG

Yet Another Next Generation (YANG) is a data modeling language used to model NE configurations and state data in standard methods. It is used to model services and NEs and provides a mechanism for mapping service models to NE models. This section describes YANG models based on RFC 7950.

Basic Concepts

YANG models the hierarchical organization of data as a tree in which each node has a name, a value, or a set of child nodes. YANG provides clear and concise descriptions of the nodes, as well as the interaction between these nodes. The following describes basic concepts in YANG:

- Module and submodule

  YANG data models are defined by modules. A module contains a collection of related definitions. A YANG-supporting NE can implement multiple modules, allowing different views of the same data. Alternatively, the NE can implement only one module to define all data.

  Definitions in a module can be separated to submodules based on the module design. The import statement allows a module or submodule to reference definitions in other modules or submodules, and the include statement is used in a module to identify the submodule that belongs to it.

Each file or module requires a dedicated namespace, which is a globally unique URI. A standard YANG-defined namespace is in the format urn:ietf:params:xml:ns:yang: *module name*.

When used inside the module statement, the prefix statement defines the prefix to be used if this module is imported. When used inside the import statement, the prefix statement defines the prefix to be used when accessing definitions inside the imported module.

The following example defines a module named HVPNService.

```
module HVPNService {
   namespace "http://example.com/HVPNService";
   prefix CreateInterfaceService;

   import huawei-ac-applications {
       prefix app;
   }

   import huawei-ac-nes-device {
       prefix device;
   }

   description
       "The module for HVPNService example.";

   revision 2018-12-09 {
       description "Initial revision.";
   }
   augment "/app:applications" { ... }
}
```

- Leaf

  A leaf node contains simple data, such as an integer or a character string. It only has one value of a particular type and no child nodes.

  The following is an example.

  ```
  leaf host-name {
    type string;
    description "Hostname for this system";
  }
  ```

- Leaf-list

  A leaf-list contains only one leaf node and has no keys. It can be only created or deleted.

  The following is an example.

  ```
  leaf-list domain-search {
    type string;
    description "List of domain names to search";
  }
  ```

- Container

  A container node is used to group nodes. It has only child nodes and no value. A container node can contain child nodes of the container, leaf, leaf-list, and list types.

  The following is an example.

  ```
  container system {
    container login {
      leaf message {
      type string;
      description "Message given at start of login session";
      }
    }
  }
  ```


- List

  A list is a collection of data nodes identified by keys. The unique constraint determines the data nodes that must be unique. A list can contain child nodes of the container, leaf, and leaf-list types.

  The following is an example.

```
list user {
  key "name";
  config true;
  description "This is a list of users in the system.";
  leaf name {
    type string;
  }
  leaf type {
    type string;
  }
  leaf full-name {
    type string;
  }
}
```

- Configuration data and state data

  YANG distinguishes configuration data from state data using the config statement. When a node is flagged with **config false**, its sub-hierarchy is flagged as state data, whereas when a node is flagged with **config true**, its sub-hierarchy is flagged as configuration data.

- Built-in and derived types

  When defining leaf and leaf-list nodes, YANG specifies the value type. YANG has a set of built-in types: binary, bits, boolean, decimal64, empty, enumeration, identityref, int8, int16, int32, int64, leafref, string, unit8, unit16, unit32, and unit64. In addition, YANG can define new derived types from built-in types and other derived types using the typedef statement.

  The following is an example.

```
typedef percent {
  type unit8 {
    range "0 .. 100";
  }
}
```

  The new type named **percent** can be used in the current YANG model.

```
container group-statistics {
  list group {
    key "name";
    leaf name {
      type string;
    }
    leaf proportion {
      type percent;
    }
  }
}
```

- Extending data model

  The augment statement defines an extending data model in the current module for another module, for example, adds some attributes in a container or list node. It is defined with a dedicated packet format as follows.

```
augment "/if:interfaces/if:ifEntry" {
    when "if:ifType='ds0'";
    leaf ds0ChannelNumber {
        type ChannelNumber;
    }
}
```


- Grouping and choices

  A grouping defines a set of nodes that are repeatedly used in multiple modules or nodes, and can be referenced in the uses statement in the target module or node. YANG allows models to segregate incompatible nodes using the choice and case statements. The choice statement defines a set of case statements which define nodes that cannot appear together. A choice node consists of a number of branches, each defined with a case statement. The case statement in YANG is similar to that in the C programming language.

  In the following example, a grouping named **ip-port** is defined, which contains two leaf nodes named **ip** and **port**, respectively.

```
grouping ip-port {
  leaf ip {
    type inet:ip-address;
    description "ip address";
  }
  leaf port {
    type inet:port-number;
    description "port number";
  }
}
```

  The grouping **ip-port** can be used in the current data model. A choice statement is used in this example.

```
choice terminal {
  case source {
    container source {
      uses ip-port;
    }
  }
  case destination {
    container destination {
      uses ip-port;
    }
  }
}
```

- Operation and action

  In YANG models, operations define functions on the top level in modules with the RPC statement whereas actions define specific actions to take on the current container or list node.

```
module server-farm {
  namespace "urn:example:server-farm";
  prefix "sfarm";
  import ietf-yang-types {
    prefix yang;
  }
  rpc reset-specified-servers {
    input {
      leaf-list servers {
        type "inet:ip-address";
      }
    }
  }
  list servers {
    key "name";
    leaf name {
      type string;
    }
    leaf address {
      type inet;
    }
    leaf location {
      type string;
    }
```



```
 action reset {
   input {
     leaf reset-at {
       type yang:date-and-time;
     }
   }
   output {
     leaf complete-at {
       type yang:date-and-time;
     }
   }
 }
}
```

In this example, a module named **server-farm** is defined. An RPC is defined in this module. The input parameters of this RPC are a set of IP addresses used to specify the servers to be restarted. The action **reset** is defined in the list **servers**. The input of this action is the time when servers start to be restarted, and the output is the time when servers are restarted.

1.3 Jinja2

Jinja2 is a modern and designer-friendly template engine for Python. The open programming system can process service packets quickly based on Jinja2 templates. This section provides a brief introduction to Jinjia2 by using an example in combination with Jinja2 template design documentation at the Jinja official website. Simply put, a Jinja2 template is a text file that contains tags, variables, and expressions. The tags in the file are used to control the internal logic of the template, and variables and expressions are replaced with specific values when the template is rendered.

The following is an example.

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
   <nes>
      <ne>
          <neid>{{neName.deviceName| to_ne_id}}</neid>
         <ifm xmlns="urn:huawei:yang:huawei-ac-ne-ce-ifm">
            <interfaces>
               {% for info in ifm.interfaces.interface %}
               <interface>
                  <ifName>{{ info.ifName }}</ifName>
                  <ifDescr>{{ info.ifDescription }}</ifDescr>
                  <ifmAm4>
                     <am4CfgAddrs>
                        <am4CfgAddr>
                           <ifIpAddr>{{ info.ifIP.ipv4Addr }}</ifIpAddr>
                           <subnetMask>255.255.255.0</subnetMask>
                        </am4CfgAddr>
                     </am4CfgAddrs>
                  </ifmAm4>
               </interface>
               {% endfor %}
            </interfaces>
         </ifm>
      </ne>
   </nes>
</inventory-cfg>
```

The preceding template contains the following two delimiters, which are used to identify control structures and variables, respectively:

- {% ... %}

  It is the tag of a control structure used to control Python scripts, and process if, elif, else conditionals, for-loops, as well as marcos and blocks. In this example, a for-loop is processed. For details about control structures, see "List of Control Structures" in the *Jinja Template Design Documentation*.

- {{ ... | ... }}

  It is the tag of an expression used to control Python variables and expressions. In the for-loop at the innermost layer of this example, it is used to control the attributes of the literal **info**. For details about expressions, see "Expressions" in the *Jinja Template Design Documentation*.

In addition, the filter **to_ne_id** is used in the <neid> tag in this example. In a Jinja2 template, a filter is used to specify the method for modifying the target variable. An expression that uses a filter contains a vertical bar (|). The part before the symbol is the target variable, and the part following the symbol is the method for modifying the target variable. Jinja2 provides some common filters. You can also configure filters as required.

For more information about Jinja2, see the *Jinja Template Design Documentation* at the Jinja official website.

1.4 Python Programming Basics

Python is an object-oriented high-level programming language. It has simple syntax, high readability, wide application scope, and smooth learning curve. Considering these, Python is used as the main software package development language for open programming. If you want to learn more about Python, see tutorials on the Internet.

1.5 Software Package Structure

The software packages exported from the OPS include:

- Python software packages

  When a Python software package is activated or uninstalled, the package manager instructs the Python-running container to install or uninstall the software package, respectively.

Python language

package.zip

|---package ------- Folder name, which is the same as the name of the compressed package.

| |---pkg.json [Mandatory] Package description file.

| |---bin Compilation command directory.

| |---python Python code directory. This directory exists only when the mapping type contains Python.

| |--- Code developed using Python

| |---resources Directory for storing resources. This directory exists only when the mapping type contains Python.

| |---template Directory for storing templates. This directory is available only to SSP packages.

| |---test Directory for storing unit test files. This directory is available only to SSP packages.

| |---yang Directory for storing the YANG model.

1.6 Logging

1.6.1 Viewing Logs

- Path of microservice logs: **/opt/oss/log/NCE/AOCService/AOCService-0-0/log/aocservice.log**

  Log format: *Time Log level* [*Thread ID*] [*User*] [*Class Line number*] *Log content*
- Path of Python run logs: **/opt/oss/log/NCE/ExtendedPkgRTService**

  Log format: *Time* (*Thread ID*) [*Log level*] [*Class Line number*] *Log content*

1.6.2 Setting the Log Level

Python run log levels can be customized as needed.

Log Configuration File logger.conf

The content of the **logger.conf** file is as follows:

[logger_user]

level=INFO

NOTE

The log level can be any of the following: DEBUG, INFO, WARN, or ERROR, which is case-insensitive Log levels can be set dynamically without the need of microservice restart.

Log Configuration File Path

/opt/oss/envs/Product-ExtendedPkgRTService/{$version}/venv-{$group_name}/Python/{$package_name}/resources

NOTE

{$version} indicates the version of the microservice ExtendedPkgRTService running in the current environment.

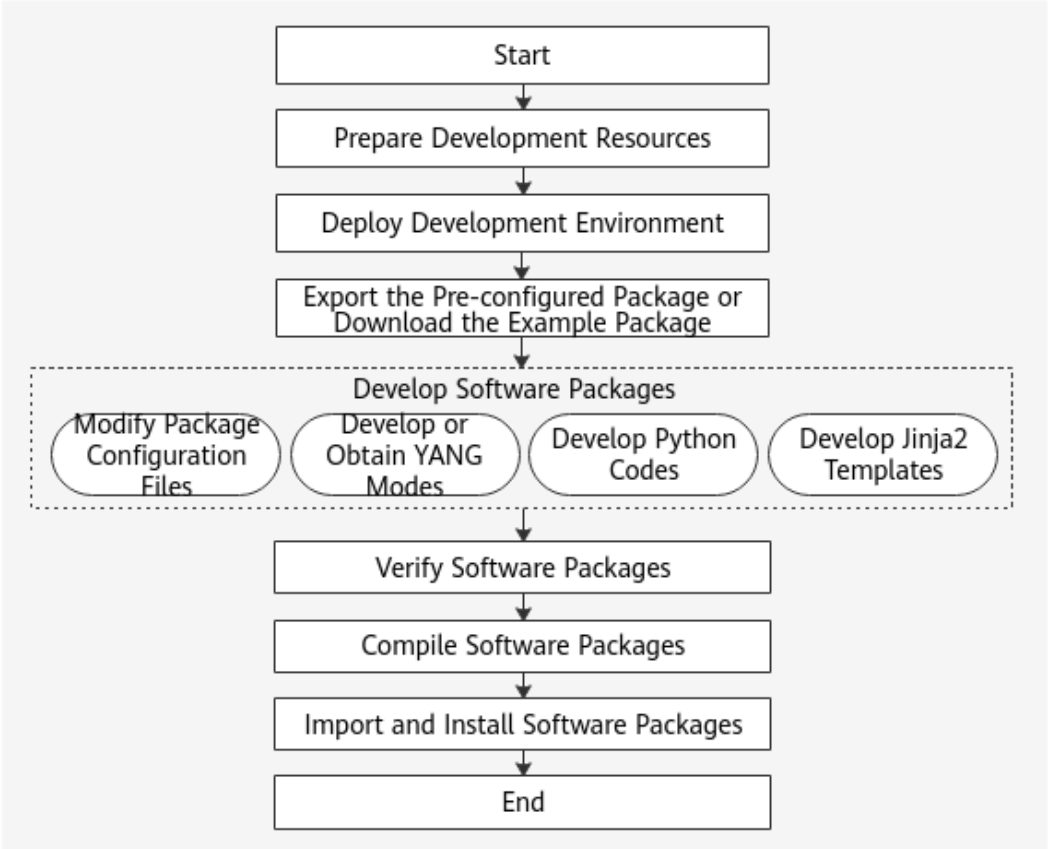
{$group_name} indicates the name of the group to which the user-developed third-party package belongs.

{$package_name} indicates the name of the user-developed third-party package.

2 Development Process

The following figure shows the software package development process.

Figure 2-1 Software package development process



3 Development Preparation

This section describes the resources required for developing software packages.

No.	Name	Description	How to Obtain
1	aoc_api-2.0.0-py3-noneany.whl	The open and programmable SDK is a set of Python-based programming interfaces provided by Huawei. It contains the base class NcsService to be inherited by derived classes and all interfaces that may be called.	Log in to **https://intl.devzone.huawei.com/community/en/aoc/resDownload.html** and download the SDK file.
2	yang-offline-util.zip	Offline verification tool for YANG models.	Log in to **https://intl.devzone.huawei.com/community/en/aoc/resDownload.html** and download the verification tool.
3	Sample code packages for open programming in typical scenarios	Sample code packages for open programming in typical scenarios, which facilitate user customization.	Log in to **https://intl.devzone.huawei.com/community/en/aoc/resDownload.html** and obtain the required sample code packages.

4 Development Environment Deployment

This chapter uses Windows 10 as an example to describe how to deploy a development environment.

4.1 Installing Python

Downloading and Installation

Visit the **Python official website** and click **Download Python** *3.x.x* to download and install the Python parser. During the installation, select **Add Python** *3.x* **to PATH** to automatically add the Python installation path to environment variables. You are advised to install Python 3.8.2. Other versions are not fully tested by Huawei, and software packages developed using Python of other versions may fail to be activated.

Verification

After the installation is successful, you can verify the installation.

Step 1 Choose **Start**. Then, choose **Windows System** > **Command Prompt**.

Step 2 Enter **python** at the current prompt. The command output is as follows.

```
C:\Users\demo>python
Python 3.8.2 (tags/v3.8.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

----End

Manually Configuring Environment Variables

If **Add Python 3.x to PATH** is not selected during the installation, the installation path of Python is not automatically added to the environment variables. In this

case, perform the following steps to manually configure the environment variables:

Step 1 Right-click **This PC** and choose **Properties** from the shortcut menu. The **System** page is displayed.

Step 2 In the navigation pane, choose **Advanced system settings**. The **System Propertie** window is displayed.

Step 3 On the **Advanced** tab page, click **Environment Variables**. The **Environment Variables** window is displayed.

Step 4 In the user variables area, double-click **Path**. The **Edit environment variable** dialog box is displayed.

Step 5 Click **New** and enter the Python installation path or click **Browse** to select the Python installation path.

Step 6 Click **New** and enter Python installation path **\Scripts** or click **Browse** to select the **Scripts** directory in the Python installation path.

Step 7 Click **OK**.

Step 8 If the command output is normal, the environment variables are configured successfully.

----End

4.2 Installing the Open and Programmable SDK

To check whether the compiled Python logic meets expected requirements in offline mode, you need to install the **open programmability SDK** and required third-party dependency packages. The open and programmable SDK is a set of Python-based programming interfaces provided by Huawei. It contains the base class NcsService to be inherited by derived classes and all interfaces that may be called. During the SDK installation, required third-party dependency packages are automatically downloaded and installed. You need to configure the pip image source on the local host in advance. After the SDK is installed, all third-party packages on which SDK depends are also installed on the local host.

NOTE

1. There is no open-source plan for the open and programmable SDK, which must be installed by running the .whl installation package on the local host. You can obtain the installation package from the matching software resource package.
2. The EasyMap algorithm automatically calculates differences. For key old configurations, you can use the nodelete label to prevent automatic deletion.

Step 1 Copy the obtained SDK installation package **aoc_api-x.y.z-py3-none-any.whl** to the current user directory.

Step 2 Choose **Start**. Then, choose **Windows System** > **Command Prompt**.

Step 3 Run **pip install aoc_api-x.y.z-py3-none-any.whl** at the current prompt. If information similar to the following is displayed, the open and programmable SDK is successfully installed.

```
C:\Users\demo>pip install aoc_api-2.0.0-py3-none-any.whl
...
```



```
...
Successfully installed aoc-api-2.0.0
You are using pip version 18.1, however version 20.2.2 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

----End

4.3 Installing an IDE

Downloading and Installation

Visit **PyCharm website** or **IntelliJ IDEA website** and download and install the integrated development environment (IDE) with a version applicable to your operating system.

Configuring the Python Parser

The Python parser must be configured after PyCharm or IntelliJ IDEA installation so that PyCharm or IntelliJ IDEA can work properly. The following uses PyCharm as an example to describe how to configure the Python parser.

Step 1 In the **Start** menu, click **JetBrains PyCharm Community Edition** to run PyCharm Community Edition.

Step 2 Use the default settings in the initial settings, and click **Skip Remaining and Set Defaults** in the **Customize PyCharm** dialog box.

Step 3 In the **Welcome to PyCharm** window, select **Settings** from the **Configure** drop-down list in the lower right corner.

Step 4 In **Settings for New Projects**, click **Project Interpreter**. Click the setting button on the right of the **Project Interpreter** drop-down list box and select **Add...**. Select **System Interpreter** and select the Python 3.x installation path in **Select Python Interpreter**.

Step 5 Click **OK**. The **Project Interpreter** list in the **Settings for New Projects** dialog box displays the version and path of the installed Python parser and the version information about the pip and setuptools extension packages.

After the configuration is complete, PyCharm can be used properly.

----End

4.4 Installing Gpg4Win and Generating Key Files for Signature

GNU Privacy Guard for Windows (Gpg4Win) is a free and open-source tool that can verify OpenPGP signatures in Windows. The OPS uses this software to sign software packages.

Downloading and Installation

Visit the **Gpg4Win official website** and download and install Gpg4Win of the latest version.

Generating a Signature File

After the installation is complete, perform the following operations to use the Kleopatra tool to generate an OpenPGP signature file.

Step 1 In the **Start** menu, click **Kleopatra** to run the signature tool.

Step 2 Choose **File** > **New Key Pair...** from the main menu to run **Key Pair Creation Wizard**. Then, click **Create a personal OpenPGP key pair**.

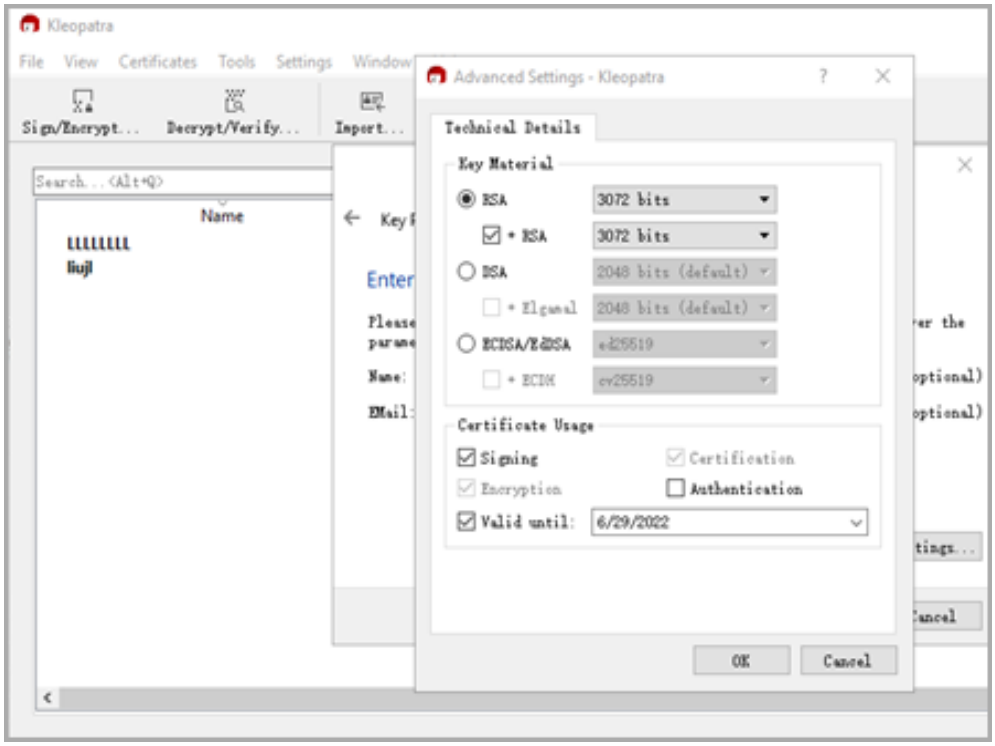
Step 3 Set **Name** and **Email** in **Enter Details**.

Step 4 Click **Advanced Settings**. In **Key Material**, select **RSA** and set the key length to 3072 bits. Click **OK**. Then, click **Next**.

NOTE

Do not select **Authentication** in the **Certificate Usage** area.

Figure 4-1 Advanced settings



Step 5 In the **Review Parameters** window, select **Show all details**. Confirm the parameter settings and click **Create**.

Step 6 When configuring a key pair, set and record the password in the dialog box that is displayed. Then, click **OK**. This password will be used when you develop a software package. Keep it safe.

Step 7 In the **Key Pair Successfully Created** window, click **Make a Backup Of Your Key Pair** to save the private key file to a secure location.

You are advised to name the private key file **privkey.asc**. When saving the private key, you need to enter the password set in the previous step.

Step 8 Click **Finish** to close the **Key Pair Creation Wizard** dialog box and find the created key pair in the certificate list. Right-click and choose **Export** from the shortcut menu to export the public key file. A public key file needs to be uploaded when a software package is imported to the OPS.

----End

5 Developing an SND Package

You can develop SND packages of protocols such as NETCONF, CLI, and RESTCONF to quickly customize drivers for devices.

The following types of SND packages can be developed:

- NETCONF SND package: provides a general-purpose driver for YANG-to-NETCONF conversion. You can develop NETCONF SND packages to quickly customize drivers.
- CLI SND package: provides a general-purpose driver for YANG-to-CLI conversion. You can develop CLI SND packages to quickly customize drivers.
- NETCONF and CLI SND package: provides a dual-protocol (NETCONF and CLI) driver. You can develop dual-protocol SND packages to quickly customize dual-protocol drivers. The CLI driver in a dual-protocol SND package is used for transparent transmission of northbound CLIs. CLI is generated by a third-party app and transparently transmitted to a device through a northbound API over the CLI protocol channel.
- RESTCONF SND package: provides a general-purpose driver for YANG-to-RESTCONF conversion. You can develop RESTCONF SND packages to quickly customize drivers.

This section uses the NETCONF SND package of Huawei NE8000 M8 as an example to describe the development process.

When developing an SND package, you need to protect the privacy of key information, such as the user password, login token, session, and other personal data. Do not display this information in logs.

Preparing a Development Environment

The product supports the development of device atomic driver packages using Python. If Python is used, prepare the environment by referring to the section "Development Environment Deployment".

5.1 Creating an SND Package Template

Step 1 Choose **Package Repo** from the main menu. Then, choose **Package Management** from the navigation pane, and click **Create Template** on the displayed page.

Step 2 In the **Add** dialog box that is displayed, set the attributes of the software package.

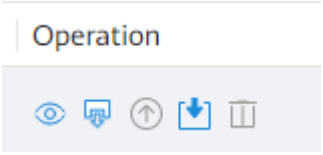
Table 5-1 lists the software package attributes.

Table 5-1 Attributes of the NE8000 M8 software package

Attribute	Value
Name	NE8000M8_SND
Version	1.0.0
Provider	HUAWEI
Package type	Specific NE driver
Mapping type	python
Device type	NetEngine 8000 M8
Protocol	NETCONF

Step 3 Click **OK**. In the dialog box that is displayed indicating that the software package is added successfully, click **OK**. The new software package is displayed in the software package list.

Step 4 Click in the **Operation** column of the software package record to download the software package to the local PC.



----End

5.2 Developing an SND Package

This section describes how to develop SND packages.

5.2.1 Editing the SND Package Configuration File

Step 1 Open PyCharm (the community edition is used as an example) and click **Create New Project**. The **New Project** dialog box is displayed. Expand the **Project Interpreter** area, confirm the configuration, and click **Create**. If a project has been created, you can directly open it in PyCharm.

Step 2 Decompress the downloaded software package to the directory where the project is located.

Step 3 In the navigation pane of the IDE, double-click the **pkg.json** file.

Step 4 Change the parameter values according to the table below.

Table 5-2 Parameter values in the **pkg.json** file

Parameter	Description	Mandatory/ Optional
name	Software package name.	Mandatory
version	Software package version number. The version number must be in the xxxx.xxxx.xxxx format. The value of each *x* ranges from 0 to 9, and each segment supports a maximum of four *x*s.	Mandatory
description	Package description.	Optional
package-type	Package type.	Mandatory
group	Package group information, which is used for sandbox process isolation.	Optional
producer	Provider of the software package. For SND and GND packages, only the packages with this field being **huawei** are supported currently.	Mandatory
augment-isolation	Model range ID, which is a Boolean value and can only be **true** or **false**. In the DCN single-NE scenario, set this parameter to **true**. In other scenarios, set this parameter to **false**.	Optional
nce-min-versions	Minimum version of the OPS to which the package is compatible with. If this parameter is left empty, the package is compatible with all versions.	Optional
package-dependencies	Dependency on third-party packages. If this parameter is set, the **name** and **version** fields must be specified.	Optional
snd-type	Type of the SND package. The value **system** indicates a built-in SND package. You are advised not to change the parameter value for built-in SND packages. The default value is **specific**.	Optional
snd-id	SND package ID. This parameter is valid only when **package-type** is set to **snd**, and must be unique for each SND package.	Optional

Parameter	Description	Mandatory/ Optional
devices	Information about the device matching the driver, which is specific to SND and GND packages. This parameter describes the software package and is not used for device management. If this parameter is set, the **vendor**, **device-type**, and **device-version** fields must be specified.	Optional
service-name	Service name, which is specific for SSP packages. Packages with different names cannot use the same service name.	Optional
hooks	Callback mapping information. The hooks combination must be unique in a single package, but can be the same in different packages. The snd-id combination must be unique. Packages with different names cannot use the same snd-id. If this parameter is set, the **type** and **key** fields must be specified, and the **java-class-name**, **Python-class-name**, **groovy-class-name**, and **template** fields are optional. The parameter **type** needs to be set to **mapping**, **service-rpc**, and **discover** when you configure the EasyMap, RPC, and discover functions, respectively. Multiple hooks can be set when multiple functions are developed together.	Mandatory

Check the file content and add attributes such as **device-version** and **nce-min-versions**. An example of the file is as follows:

```
{
    "name": "NE8000M8_SND",
    "version": "1.0.0",
    "description": "NE8000M8_SND",
    "package-type": "snd",
    "producer": "HUAWEI",
    "nce-min-versions": [
        "1.0.0"
    ],
    "snd-id": "NE8000M8_SND",
    "devices": [
      {
        "vendor": "HUAWEI",
        "device-type": "NetEngine 8000 M8",
        "device-version": "V800R012C00SPC300"
      }
    ],
    "hooks": [
      {
        "type": "snd",
        "key": "ecs-driver",
        "python-class-name": "com.huawei.controller.devicetype.ne8000m8snd.NE8000M8NETCONF"
      }
    ]
}
```


NOTE

If you need to configure **netconf-keepalive-info**, add the following hook to the **pkg.info** file:

```
{
  "type": "snd",
  "key": "ecs-connect-agent",
  "java-class-name": "com.huawei.enterprise.naas.neagent.service.NeAgentImpl"
}
```

----End

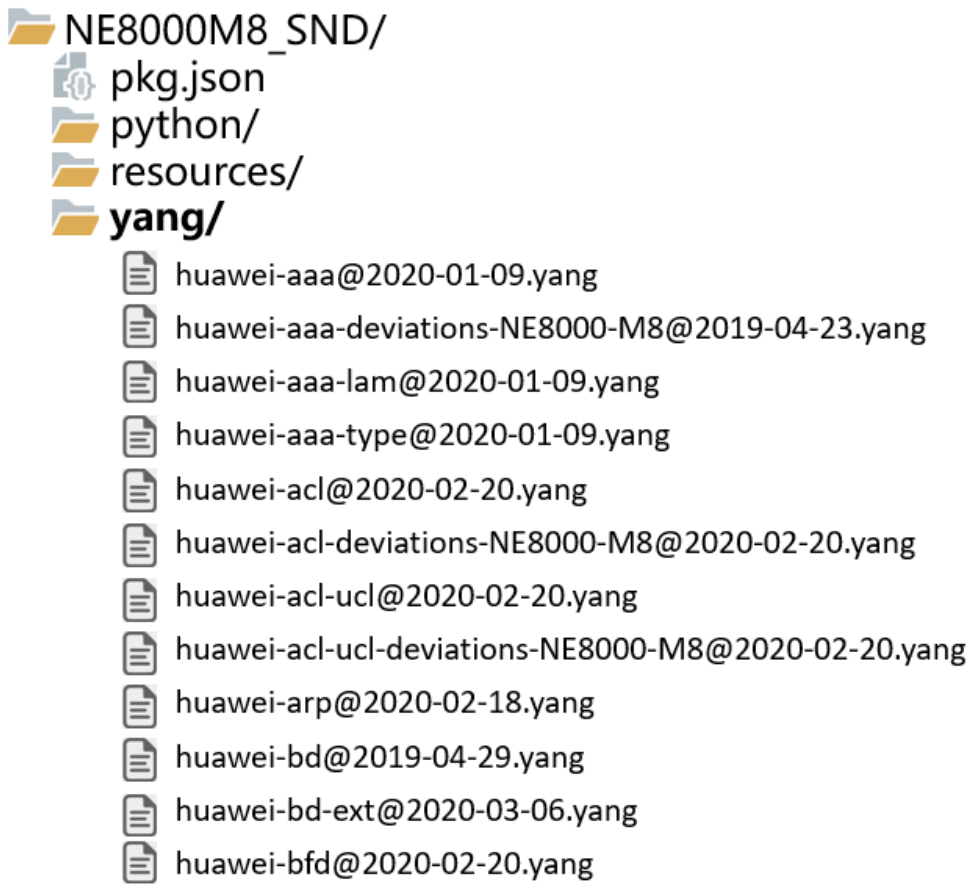
5.2.2 Writing a Device YANG Model

The YANG model in an SND package is not involved in compilation. It is used to describe the device configuration capability and northbound APIs of devices, and generate southbound packets. You can obtain or compile a YANG model based on the SND package type to replace the default YANG model file in the software package.

- NETCONF SND package: Delete the default YANG model files from an SND package, obtain the YANG model files of the NETCONF protocol from the official website of the device vendor, and add the model files to the directory where the default model files are placed.
- CLI SND package: Obtain *Openness and Programmability Framework CLI Driver User Guide* and write the YANG model of the CLI protocol based on the document.
- NETCONF and CLI SND package: You only need to obtain the YANG model of the NETCONF protocol from the official website of the device vendor.
- RESTCONF SND package: Obtain the YANG model of the RESTCONF protocol from the official website of the vendor.

In this example, the NE8000 M8 uses a NETCONF SND package. Delete the default YANG file from the SND package template and save the YANG model file obtained from the vendor for the NE8000 M8 device to the YANG file directory. **Figure 5-1** shows the YANG file directory.

Figure 5-1 YANG file directory



If you need to use functions such as user-defined data encryption, high-risk operation definition, YANG model blacklist masking, YANG model verification exemption, and user-defined model permission, refer to the following information. The sample software package does not involve these configurations.

Data Encryption

You can edit the **security.xml** file in the **yang** directory to determine whether to encrypt the data corresponding to the nodes in the YANG model.

If data encryption is enabled, data is encrypted when being saved to the database and is decrypted when being read from the database; when the configuration is delivered, the encrypted data fields are displayed as ******; when the northbound RESTCONF v1 API queries the encrypted data field, the value in ciphertext is returned; during synchronization, encrypted data fields are not synchronized; during data consistency verification, encrypted data fields are not delivered.

Note: When data is saved to the database, the key field of the list cannot be encrypted. In addition, only data of the string type can be encrypted.

Table 5-3 Parameters in the **security.xml** file

Node	Parent Node	Attribute	Description
security-node	/	N/A	Root node.
path	security-node	-	Security node path. Example: /a:b/c/d
-	-	type	Node type: leaf leaf-list typedef: customized type By default, the node is of the leaf type.

If the node type is leaf, only the <path> label needs to be defined. Generally, you do not need to set the attribute. The data corresponding to each <path> is encrypted. The value of **path** must start with the top-level node **<moduleName>:<nodeName>**, where *<moduleName>* is the module name of the YANG file and *<nodeName>* is the name of the top-level node in the module. The top-level node of the path can be followed by child nodes, which are separated by a slash (/). An example is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<security-node>
   <path>/db-security-test:security-top/security-sec/security-sec-leaf-list</path>
   <path type="typedef">/db-security-test:security-len-typedef-leaf</path>
   <path>/db-security-test:security-top/security-list/security-list-leaf</path>
   <path>/dbtest:nodes/dbtest:node/db-security-test:augment-cont/augment-cont-security-leaf</path>
   <path>/dbtest:nodes/dbtest:node/db-security-test:augment-security-leaf</path>
</security-node>
```

High-Risk Operations

You can edit the **risk_operation.xml** file in the **yang** directory to define whether the RPC operation in the YANG model is a high-risk operation. When you perform a high-risk operation on the GUI, a confirmation dialog box is displayed. No confirmation dialog box will be displayed for non-high-risk operations.

All RPC operations are defined in the <rpcs> label. Each RPC high-risk operation corresponds to one <rpc> label. The value of the <rpc> label is in the format moduleName:rpcName. An example is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<risk-operation>
   <rpcs>
      <rpc>huawei-aaa:reactivateUser</rpc>
      <rpc>huawei-aaa:changeMyPassword</rpc>
      <rpc>huawei-aaa:changeMyIdleTimeout</rpc>
   </rpcs>
 </risk-operation>
```


Blacklist

You can edit the **blacklist.xml** file in the **yang** directory to define whether modules in the YANG model are shielded on the GUI.

The modules to be shielded are defined in <module>, in which the module name and module revision need to be set. An example is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<blacklist>
  <module>
     <name>huawei-cfg</name>
     <revision>2020-03-18</revision>
  </module>
  <module>
     <name>huawei-dhcp</name>
     <revision>2020-04-29</revision>
  </module>
</blacklist>
```

YANG Model Verification Exemption

You can edit the **multiRevisionModule.xml** file in the **yang** directory to define whether modules in the YANG model need to be verified during the upgrade.

Verification-free modules are defined in <modules>. Each module corresponds to one <module> label, which is set to the module name. An example is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<modules>
   <module>ietf-yang-types</module>
   <module>ietf-inet-types</module>
 </modules>
```

User-Defined Model Permission

By writing a permission control file, you can customize operation permissions on nodes in YANG models. After a software package is successfully installed, the permission information is dynamically injected to the OPS. The user management function allows users with different responsibilities to be granted with appropriate permissions, preventing unauthorized and insecure operations. Permissions on software package models are automatically assigned to the default role whereas permissions assigned to user-defined roles need to be configured manually.

You can add the **permission.json** file to the **resources** directory and customize permissions on nodes in YANG models. If you have not written the **permission.json** file, the OPS automatically generates a level-0 permission control file for each module based on the YANG file during software package loading. The permission control file includes the create, delete, read, update, and execute permissions.

The OPS supports permission-based operations on container and list nodes in YANG models. You can define permissions on these nodes as needed. Customization of permissions at levels 0, 1, and 2 is supported. Level 0 indicates the permission control at the module level. If a user does not match permissions, the OPS executes permission control at level 0 by default. At level 1, a container or list node has only one level and does not have subnodes. At level 2, a container or list node has two levels and subnodes. During permission control, the OPS matches permissions based on the longest match rule. That is, permissions at a deeper level are matched first.


```
{
   "modules": [
   {
      "module-name":"hbng",
      "operations": [
      {
         "uri-pattern":"containerA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":"listA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":" listA/listC",
         "method":" create/delete/read/update"
      },
      {
         "uri-pattern":"resetStatictic",
         "method":"exec"
      }
      ]
   }
   ]
}
```

5.2.3 Developing Code

You can develop Python code to implement device management, configuration management, and customization of inconsistency discovery, synchronization, and consistency verification.

The NE8000 M8 and controller are provided by the same vendor, and they do not have differences in data format and packet format, eliminating the need of customization. You only need to develop the device management code and device driver data code for the software package. In addition, no customization is required for the device capabilities. By default, full inconsistency discovery is implemented.

5.2.3.1 Importing Header Files

In the navigation pane of the IDE, double-click the .py file in the Python directory to write Python code.

At the beginning of the code, you need to import the related classes in the necessary header files. Refer to the sample code to import required classes based on the SND package type and service logic.

In this example, the following classes need to be imported to the NE8000 M8 software package. For details about the classes, see the description in the following sections.

```
from aoc.snd.netconfsnd import NetconfSND
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity, DEFAULT_CONNECT,
PRIMARY_CONNECTION, HelloEntity
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL, PROTECTED_MODE
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import NetconfDriverInfo
```


Public Classes

The classes are irrelevant to the SND package type. The following is the reference code of public classes:

```
# Optional. Import the SysoidInfo class when overriding the sysoid API used for device registration, and configure the device type, vendor, and model corresponding to the SND package.
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo

# Mandatory. Import the ConnectInfos, ProtocolEntity, DEFAULT_CONNECT, PRIMARY_CONNECTION, and HelloEntity classes to configure connection information, such as the connection protocol, number of established channels, and handshake hello packets.
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity, DEFAULT_CONNECT, PRIMARY_CONNECTION, HelloEntity

# Optional. Import the SINGLE_CHANNEL class, which is used for configuring device connection capabilities.
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL

# Optional. Import the CommonDriverInfo class. This class is used to customize operation types in device configuration packets based on common capabilities. For example, if a device does not support the create operation, you can use the merge operation. You can also customize the device synchronization mode, for example, deleting devices first and then adding the devices.
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo

# Optional. Import the FeatureCfgsMsg, Feature, and Function classes for configuring device feature capabilities.
from aoc.snd.snd_model_pb2.FeatureCfgs_pb2 import FeatureCfgsMsg, Feature, Function

# Optional. Import the EcsDbConfigOut and EcsDbConfig classes for configuring data association.
from aoc.snd.snd_model_pb2.ecsDbProcess_pb2 import EcsDbConfigOut, EcsDbConfig
```

Classes to Be Imported to a NETCONF SND Package

The following code is a reference for the classes to be imported during NETCONF SND development:

```
# Mandatory. When developing a NETCONF SND, you need to introduce the NetconfSND class, that is the parent class of the SND. The parent class has been configured by default. For customized configuration, you need to inherit the parent class NetconfSND in the SND and override the methods in the parent class.
from aoc.snd.netconfsnd import NetconfSND

# Optional. Import the NetconfDriverInfo class, which is used for configuring device types. This class is customized based on NETCONF capabilities and is used to configure NETCONF parameters, such as whether two phases are supported and whether the set operation is supported.
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import NetconfDriverInfo
```

Classes to Be Imported to a CLI SND Package

The following code is a reference for the classes to be imported during CLI SND package development:

```
# Mandatory. The CliSND class needs to be imported when a CLI SND package is developed.
from aoc.snd.clisnd import CliSND

# Optional. Import the CliDriverInfo class, which is used for configuring device types and is customized based on the CLI capability.
from aoc.snd.snd_model_pb2.cliDriver_pb2 import CliDriverInfo

# Mandatory. The CliCustomTransformOutput class defines the CLI-to-YANG conversion output, the OperType class defines the operation type, and the CliToYangCustom class defines the CLI-to-YANG conversion logic.
from aoc.snd.snd_model_pb2.cliTransform_pb2 import CliCustomTransformOutput
from aoc.snd.snd_model_pb2.cliTransform_pb2 import OperType
from .clitoyang import CliToYangCustom
```


Classes to Be Imported to a NETCONF and CLI SND Package

The following code is a reference for the classes to be imported during dual-protocol (NETCONF and CLI) SND package development:

```
# Mandatory. The netconfClisnd class needs to be imported during NETCONF and CLI SND package development.
from aoc.snd.netconfClisnd import netconfClisnd

# Optional. Import the NetconfDriverInfo class.
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import NetconfDriverInfo
```

NOTE

In an SND package, the combination of a user-defined package name and class name must be globally unique. Otherwise, the package will fail to be activated.

5.2.3.2 Writing the Code for Device Management Customization

Write the code for device management customization, including setting device sysoid information and device connection capability information. The code is public code. You can refer to the comments in the code to customize the code.

Setting Device sysoid Information

This API is used to set the sysoid information for a device to manage this device model. If the sysoid information of the device needs to be verified during device management, the sysoid field must be correctly set. If the sysoid field is not specified when a device is added, the system matches the device based on the triplet in the **pkg.json** file.

In this example, the code for registering device information in the NE8000 M8 software package is as follows:

```
# Register the sysoid value, type, model, and vendor of the device.
def getSysoidInfo(self, aoccontext, request=None):
    sysoidInfo = SysoidInfo()
    sysoidEntity = sysoidInfo.sysoidEntity.add()

    # Set sysoid for a device.
    sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.360.1.10"

    # Set the device type.
    sysoidEntity.deviceType = "ROUTER"

    # Set the device model.
    sysoidEntity.deviceModel = "NetEngine 8000 M8"

    # Set the device vendor.
    sysoidEntity.deviceVendor = "HUAWEI"
    return sysoidInfo
```

Setting Connection Capability Information for a Device

NETCONF and CLI SND packages support dual protocols. Therefore, you need to set both primary and auxiliary protocol connection capability information. For other types of SND packages, you only need to set primary protocol connection capability information.

In this example, you only need to configure the primary protocol connection capability information in the NE8000 M8 software package. The code is as follows:


```
# Configure the NETCONF protocol type and connection capability information. In this example, a single
channel is shared for read and write.
def getConnectInfo(self, aoccontext, request=None):
    self.logger.info('getConnectInfo start.')
    connectInfos = ConnectInfos()

    # Configure primary protocol capability information for all types of SND packages.
    primaryConnectInfo = connectInfos.connectInfo.add()

    # Set a connection policy.
    primaryConnectInfo.connectPolicy = DEFAULT_CONNECT

    """
    Set channel parameters. NETCONF requires different channel types.
    SINGLE_CHANNEL indicates that a single link is used, with no protection implemented.
    If is_read_share_write is set to true, a single channel is shared for read and write. In this case,
readChannel does not need to be set.
    """
    primaryConnectInfo.channelInfo.writeChannel = SINGLE_CHANNEL
    primaryConnectInfo.channelInfo.is_read_share_write = True

    # Set the primary protocol type.
    primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.netconf

    """
    Set the packet type. defaultType: Establish a NETCONF channel for schema interconnection and leave the
capability set empty.
    standardType: Establish a NETCONF channel for the standard YANG capability set and leave the
capability set empty.
    extendType: Establish a NETCONF channel for the extended YANG capability set. A capability set list
needs to be added.
    """
    primaryConnectInfo.protocolEntity.helloEntity.helloType = HelloEntity.extendType

    # Set the connection invoking priority.
    primaryConnectInfo.connectionPriority = PRIMARY_CONNECTION

    # For NETCONF and CLI SND packages, you also need to set the auxiliary protocol connection capability
information. For details, see the comments in the code for configuring the primary protocol connection
capability information.
    """
        auxiliaryConnectInfo = connectInfos.connectInfo.add()
    ...
    ...
    """
    self.logger.info('getConnectInfo end.')
    return connectInfos
```

5.2.3.3 Writing the Code for Device Type Capability Customization

Write the code for device type capability customization, including device driver data configuration, service customization processing, data pre-processing and post-processing, and data association processing.

5.2.3.3.1 Configuring Device Driver Data

You can write Python code or write an XML file to configure the device driver data. The following describes these two methods.

Writing Code to Configure Device Driver Information

- **Define interaction policies with devices based on the common capability customization example.**

Compile an interaction policy based on common capabilities. The code of the general-purpose capability interaction policy for the NE8000 M8 software package is as follows:

```
# Configure the general-purpose driver of the device. The operation types that are not supported by the device are automatically converted. For example, "create,delete" is converted into "merge,remove". If syncToDel.value is set to true, the configured data of southbound devices can be deleted during data consistency verification.
def getCommonDriverInfo(self, aoccontext, request=None):
    common_driver = CommonDriverInfo()


     # Operation that is not supported by the device, which is automatically converted. For example, "create,delete" is converted to "merge,remove".
    common_driver.unsupportedOperations = "create,delete"
    syncToDel = common_driver.para.add()

    # Define the key value for synchronous deletion.
    syncToDel.key = "sync-to-del-enable"

    # Whether to delete southbound device configuration instances during data consistency verification.
    syncToDel.value = "true"
    return common_driver
```

- **Define NETCONF parameters based on the protocol type customization example.**

  NETCONF SND packages and NETCONF and CLI SND packages support the NETCONF protocol. Therefore, you need to define NETCONF parameters.

  The customization code of the protocol type for the NE8000 M8 software package is as follows:

```
# Configure the NETCONF driver of the device. Configure the device to support two-phase delivery. Set the device prototype. Most Huawei devices use huawei-v5. Enable unordered delivery for the packets to be delivered to devices.
def getNetconfDriverInfo(self, aoccontext, request=None):
    netconf_driver = NetconfDriverInfo()

    # Number of phases in which NETCONF configurations are delivered.
    netconf_driver.phase = "two"

    # Define the driver classification.
    netconf_driver.classification = "huawei-v5"

    """
    Define the packet conversion type. If the value is same, schema standard packets are delivered.
    If the value is match, YANG standard packets are delivered.
        netconf_driver.modelDiff = "same"
    """

    # Define the packet verification policy. When the value of this field is set, the packet is directly processed without verification.
    netconf_driver.testOption = "set"
    return netconf_driver
```

- **Define CLI parameters based on the protocol type customization example.**

  CLI SND packages and NETCONF and CLI SND packages support the CLI protocol. Therefore, you need to define CLI parameters.

  The NE8000 M8 software package does not involve this operation. Refer to the following code to perform customization for other software packages:

```
    def getCliDriverInfo(self, aoccontext, request=None):
        self.logger.info('getCliDriverInfo start.')
        cliDriverInfo = CliDriverInfo()
        self.logger.info(self.get_dict())

    # Assign the value of the key-value pair in the dictionary to the configuration item.
```



```
    for key, value in self.get_dict().items():
        cliDriverEntity = cliDriverInfo.cliDriverEntity.add()
        cliDriverEntity.key = key
        cliDriverEntity.value = value
    self.logger.info('getCliDriverInfo end.')
    return cliDriverInfo


# Read the information in the device driver file resources/cli-driver.xml to the dictionary.
def get_dict(self):
    try:
        path = os.path.join(self.resourceDir, "resources/cli-driver.xml")
        self.logger.info(self.resourceDir)
        self.logger.info(path)
        pro_file = open(path, 'r')
        properties = {}
        for line in pro_file:
            if line.find('=') > 0:
                strs = line.replace('\n', '').split('=')
                properties[strs[0]] = codecs.getdecoder("unicode_escape")(strs[1])[0]
    except IOError as e:
        self.logger.info('getCliDriverInfo start.')
        self.logger.info(e)
        return {}
    else:
        pro_file.close()
    self.logger.info(properties)
    return properties
```

Writing an XML File to Configure Device Driver Information

- NETCONF SND package: You can customize device driver information in the **resources>netconf-driver.xml** file.
- CLI SND package: You can customize device driver information in the **resources>cli-driver.xml** file.
- RESTCONF SND: You can customize device driver information in the **resources>restconf-driver.xml** file.
- NETCONF and CLI SND package: You can customize device driver information in the **resources>netconf-driver.xml** file. You can also configure a whitelist for transparent transmission of CLI commands in the **resources>CliPassthroughCommands.xml** file. Only CLI commands in the whitelist can be transparently transmitted to devices through northbound APIs, improving service security.

No example of writing an XML file to configure device driver information is provided for the NE8000 M8 software package.

Instead, you can refer to the following configuration example using the **resources>netconf-driver.xml** file. For details about the parameters, see the API reference document.

```
<?xml version="1.0" encoding="utf-8"?>
<netconf-driver xmlns="https://huawei.com/nce/netconf-driver">
  <netconfDriverInfo>
    <!-- Type: string -->
    <classification>huawei-v8</classification>
    <!-- Type: string -->
    <errorOption>one</errorOption>
    <!-- Type: int32 -->
    <maxPkgLength></maxPkgLength>

    <!-- Type: boolean -->
    <netconfLock>false</netconfLock>
    <!-- Type: string -->
```



```
  <errorOption>stop-on-error</errorOption>
  <!-- Type: string -->
  <testOption>set</testOption>
  <!-- Type: NetconfHelloEntity -->
  <netconfHelloEntity>
     <!-- Type: ENUM NetconfHelloType; Value: standardType, extendType, or defaultType-->
     <helloType>standardType</helloType>
     <extendEntityList>
        <!-- Type: string -->
        <extendEntity></extendEntity>
     </extendEntityList>
  </netconfHelloEntity>
  <!-- Type: string -->
  <modelDiff></modelDiff>
  <rpcPrefixList>
     <!-- Type: RpcPrefix -->
     <rpcPrefix>
        <!-- Type: string -->
        <rpcQname></rpcQname>
        <!-- Type: string -->
        <prefix></prefix>
     </rpcPrefix>
  </rpcPrefixList>
</netconfDriverInfo>
<commonDriverInfo>
  <!-- Type: boolean -->
  <rollbackLastErrPackage>false</rollbackLastErrPackage>
  <!-- Type: string -->
  <pathNeedPreProcessList>
     <pathNeedPreProcess></pathNeedPreProcess>
  </pathNeedPreProcessList>
  <!-- Type: string -->
  <pathNeedPostProcess>
     <pathNeedPostProcess></pathNeedPostProcess>
  </pathNeedPostProcess>
  <!-- Type: string -->
  <pathNeedEcsMappingList>
     <pathNeedEcsMapping></pathNeedEcsMapping>
  </pathNeedEcsMappingList>
  <!-- Type: string -->
  <pathNeedEcsRpcMappingList>
     <pathNeedEcsRpcMapping></pathNeedEcsRpcMapping>
  </pathNeedEcsRpcMappingList>
  <!-- Type: string -->
  <EcsDeviceDriverParaList>
     <!-- Type: string; Attribute value -->
     <EcsDeviceDriverPara key=""></EcsDeviceDriverPara>
  </EcsDeviceDriverParaList>
  <isNeedGlobalPush></isNeedGlobalPush>
  <isPathNeedPushList>
     <isPathNeedPush></isPathNeedPush>
  </isPathNeedPushList>
  <!-- Type: string -->
  <pathNeedEcsDBHookList>
     <pathNeedEcsDBHook></pathNeedEcsDBHook>
  </pathNeedEcsDBHookList>
  <checkSync>false</checkSync>
  <!-- Type: string, separated by commas (,) -->
  <unsupportedOperations>create</unsupportedOperations>

  <!-- Type: Map<String, String> -->
  <pathUnsupportedOperationsMap>
     <pathUnsupportedOperations key=""></pathUnsupportedOperations>
  </pathUnsupportedOperationsMap>
  <!-- Type: ENUM DeleteStrategy; Value: PATH_ADD or CONTAINER_WITH_PRESENCE -->
  <deleteStrategy></deleteStrategy>
  <uiSupportAbility>
     <supportOperationsList>
        <!-- Type: string -->
```



```
            <supportOperations></supportOperations>
        </supportOperationsList>
        <connectProtocolsList>
            <!-- Type: string -->
            <connectProtocols></connectProtocols>
        </connectProtocolsList>
    </uiSupportAbility>
    <!-- Type: ENUM ConfigDeliveryStrategy; Value: SYNC_DB_AND_SYNC_DEVICE or
SYNC_DB_AND_ASYNC_DEVICE -->
    <configDeliveryStrategy></configDeliveryStrategy>
    <!-- Type: ENUM OutOfSyncDeviceStrategyInHASwitching; Value: ALARM, SYNC_TO_DEVICE, or
CUSTOM_HOOK -->
    <outOfSyncDeviceStrategyInHASwitching></outOfSyncDeviceStrategyInHASwitching>
    <!-- Type: string -->
    <SwitchHACustomHook></SwitchHACustomHook>
    <ecsCheckHookList>
        <ecsCheckHook>
            <!-- Type: string -->
            <moduleName></moduleName>
            <!-- Type: string -->
            <appClass></appClass>
        </ecsCheckHook>
    </ecsCheckHookList>
    <ecsBehaviourHookList>
        <ecsBehaviourHook>
            <!-- Type: string -->
            <apiClass></apiClass>
            <!-- Type: string -->
            <implClass></implClass>
        </ecsBehaviourHook>
    </ecsBehaviourHookList>
    <!-- Type: string; Value: one-by-one or all-together -->
    <commitMode></commitMode>
  </commonDriverInfo>
  <customDriverInfo>
    <!-- Type: string -->
    <maxPkgLength></maxPkgLength>
    <!-- Type: ENUM LanguageType; Value: JAVA, GROOVY, or OTHER -->
    <CustomLanguageType></CustomLanguageType>
  </customDriverInfo>
  <netconf-keepalive-info>
        <!--keepAliveMsg is the keepalive packet for a NETCONF connection and must be escaped using
CDATA. After the escape, the packet can meet the NETCONF packet format requirements. The value of
message-id must be replaced with ${message ID}.-->
        <keepAliveMsg><![CDATA[<?xml version="1.0" encoding="UTF-8"?><rpc message-id="${messageId}"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><get-config><source><running/></source><filter
type="subtree"><system xmlns="http://www.huawei.com/netconf/vrp" content-version="1.0" format-
version="1.0"> <systemInfo><sysUpTime/></systemInfo></system></filter></get-config></rpc>]]></
keepAliveMsg>
        <!--keepAlivePeriod: Keepalive interval. The minimum value is 10, in seconds.-->
        <keepAlivePeriod>15</keepAlivePeriod>
    </netconf-keepalive-info>
</netconf-driver>
```

5.2.3.3.2 Configuring Service Customization Processing

When the device packet conversion capability provided by the OPS cannot meet the interaction requirements, you can use this API to customize device packet conversion processing.

The NE8000 M8 software package does not involve service customization processing. You can skip this section.

- For NETCONF SND packages and NETCONF and CLI SND packages, see the following code example to configure service customization processing:

```
def netconfTransformer(self, aoccontext, input=None):
    self.logger.info('netconfTransformer start.')
```



```
    netconf_msg = NetconfMsg()
    netconf_msg.msg = input.data
    self.logger.info('netconfTransformer end.')
    return netconf_msg
```

- For a CLI SND package, the customized code implements the logic for CLI-to-YANG conversion and YANG-to-CLI conversion. For details, see the following code example:

```
    def yangToCli(self, aoccontext, request):
        self.logger.info('yangToCli start.')
        out = CliCustomTransformOutput()
        path1 = '/huawei-ifm:interfaces/huawei-ifm:interface/(.+)/huawei-ifm:isis/huawei-ifm:authentication-
mode/huawei-ifm:md5'
        raw_url = urllib.parse.unquote_plus(request.path)
        path1_res = re.search(path1, raw_url)
        if request.oper_type == OperType.Value('DELETE') and path1_res:
            list_key = path1_res.group(1)
            out.data = 'interface ' + list_key + '\nundo isis authentication-mode'
            return out

        self.logger.info('yangToCli end.')
        out.data = ''
        return out


    def cliToYang(self, aoccontext, request):
        self.logger.info('cliToYang start.')
        self.logger.info('yangToCli current path : %s' % request.path)
        if request.path == '/huawei-bfd:bfd-conf':
            return CliToYangCustom.clitoyang_bfd_conf(request, self.logger)
        if request.path == '/huawei-bgp:bgp-conf':
            return CliToYangCustom.clitoyang_bgp_conf(request, self.logger)
        self.logger.info('cliToYang end.')
```

5.2.3.3.3 Configuring Data Pre-processing

The data format may vary according to vendors. For example, definitions of enumerated values of the controller and devices from different vendors may be different. After pre-processing is configured, data can be processed when being synchronized from devices to NCE, so that NCE can identify the data.

The NE8000 M8 software package does not involve data pre-processing. You can skip this section.

- For NETCONF SND packages and NETCONF and CLI SND packages, set data pre-processing by referring to the following code example:

```
def netconfPreprocess(self, aoccontext, request=None):
    self.logger.info('netconfPreprocess start.')
    doc = request.data

    # Perform pre-processing on data.
    new_doc = doc.replace("<viewName>", "<viewName>pre")
    netconfAtomicConfig = NetconfAtomicConfig()
    netconfAtomicConfig.data = new_doc
    self.logger.info('netconfPreprocess end.')
    return netconfAtomicConfig
```

5.2.3.3.4 Configuring Data Post-processing

The data format may vary according to vendors. After post-processing is configured, data can be processed when being synchronized from NCE to devices so that the devices can identify the data.

The NE8000 M8 software package does not involve data post-processing. You can skip this section.

- For NETCONF SND packages and NETCONF and CLI SND packages, set data post-processing by referring to the following code example:

```
def netconfPostprocess(self, aoccontext, request=None):
    self.logger.info('netconfPostprocess start.')
    doc = request.data

    # Perform post-processing on data.
    new_doc = doc.replace("<viewName>", "<viewName>post")
    netconfAtomicConfig = NetconfAtomicConfig()
    netconfAtomicConfig.data = new_doc
    self.logger.info('netconfPostprocess end.')
    return netconfAtomicConfig
```

5.2.3.3.5 Configuring Data Association

When data is delivered to devices, data association will be performed between the devices. In this case, NCE also needs to perform data association to ensure data synchronization with the devices.

The NE8000 M8 software package does not involve data association. You can skip this section.

The methods for setting data association for various SND package types are similar. For details, see the following code example:

```
def dbPostProcessBatch(self, aoccontext, request)
    self.logger.info('dbPostProcess start.')
    ecsDbConfigOut = EcsDbConfigOut()
    for ecsItem in request.data:
        ecsData = ecsItem.data
        ecsPath = ecsItem.path
        ecsopType = ecsItem.opType
        self.logger.info("ecsDbconfig  getData ddw:" + ecsData)
        self.logger.info("ecsDbconfig  getPath ddw:" + ecsPath)
        self.logger.info("ecsDbconfig  getOpType ddw:" + ecsopType)
        #  /huawei-ifm:interfaces/huawei-ifm:interface/asd/huawei-ifm:mpls/huawei-ifm:l2vc/primary

        # /huawei-ifm:interfaces/huawei-ifm:interface/Giga0000001/huawei-ifm:mpls/huawei-ifm:l2vc/
secondary
        if "huawei-ifm:l2vc/primary" in ecsPath and ecsopType == "DELETE":
            self.logger.info("ecsDbconfig  getOpType primary ddw:" + ecsopType)
            ifmKey = self.get_key(ecsPath)
            ecsDbconfig = ecsDbConfigOut.ecsDbConfig.add()
            ecsDbconfig.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + "/huawei-
ifm:mpls/huawei-ifm:l2vpn/huawei-ifm:service-name"
        elif "huawei-ifm:l2vc/secondary" in ecsPath and ecsopType == "DELETE":
            self.logger.info("ecsDbconfig  getOpType secondary ddw:" + ecsopType)
            ifmKey = self.get_key(ecsPath)
            ecsDbconfig1 = ecsDbConfigOut.ecsDbConfig.add()
            ecsDbconfig2 = ecsDbConfigOut.ecsDbConfig.add()
            ecsDbconfig1.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + "/huawei-
ifm:mpls/huawei-ifm:l2vpn/reroute"
            ecsDbconfig2.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + "/huawei-
ifm:mpls/huawei-ifm:l2vpn/redundancy"

    self.logger.info('dbPostProcess end.')
    return ecsDbConfigOut
```

5.2.3.3.6 Obtaining syncId

syncId indicates the ID of a data change record. You can compare the syncId on a device with that on NCE to check whether the data on the device is consistent with that on NCE. For example, if you have delivered services to the device on the CLI, data inconsistency may occur. If data is inconsistent, you can synchronize data between the device and NCE to keep data consistency.

The NE8000 M8 software package does not involve the obtaining of syncId. You can skip this section.

Obtain syncId by referring to the following code example:

```
from tool.CommandProxy import CommandProxy

def getSyncId(self, aoccontext, request):
    self.logger.info('getSyncId start.')
    # Read configuration.
    proxy = CommandProxy(request.deviceId, self.logger)
    response_data = proxy.send_command_list([ 'return', 'system-view','diagnose','display configflowid'],120)
    response_body = response_data.response if hasattr(response_data, 'response') else response_data
    lines = response_body.splitlines()
    result = lines[4].strip().split(' ')[0]
    syncIdInfoOutput = SyncIdInfoOutput()
    syncIdInfoOutput.SyncId = result
    self.logger.info(result)
    self.logger.info('getSyncId end.%s' % (syncIdInfoOutput.SyncId))
    return syncIdInfoOutput
```

5.2.3.4 Writing the Code for Device Feature Capability Customization

If you need to customize the inconsistency discovery scope and inconsistent data on devices and NCE, you need to compile the device feature capability customization code. If no customization is performed, full inconsistency discovery is performed for all features.

In this example, the NE8000 M8 software package does not involve device feature capability customization. By default, full inconsistency discovery is performed. You can skip this section.

The methods for customizing device feature capabilities for various SND package types are similar. For details, see the following code example:

```
    def getFeatures(self, aoccontext, request=None):
        self.logger.info('getFeatures start.')
        feature_msg = FeatureCfgsMsg()
        feature_msg.replace = True

        # Add a feature customization and compile the inconsistency discovery path.
        feature_msg.features.extend(self.build_feature('huawei-l3vpn', 'huawei-rtp', '(http://www.huawei.com/
netconf/vrp/huawei-l3vpn?revision=2018-06-11)l3vpn'))
        self.logger.info('getFeatures end.')
        return feature_msg


    def build_feature(self, name, depends, path):
        feature = Feature()

        # Feature name
        feature.name = name

        # Operation type
        feature.operType = Feature.MERGE
        feature.depends.extend([depends])
        function = Function()

        # Set the inconsistency comparison path.
        function.value = path

        # Set the path for collecting data from southbound devices.
        function.collectPath = path
        feature_name = name.replace('huawei-', '')

        # Customize data processing before performing synchronization from NCE to devices. If the value is
```

```
**True**, the preSyncToNe process is invoked. If the value is **False**, no customization is performed.
        function.preSyncToNe = True

        # Customize data processing before performing synchronization from devices to NCE. If the value is
**True**, the preSyncFromNe process is invoked. If the value is **False**, no customization is performed.
        function.preSyncFromNe = False

        # Customize data processing after the synchronization. If the value is **True**, the postSyncFromNe
process is invoked. If the value is **False**, no customization is performed.
        function.postSyncFromNe = False
        feature.functions.extend([function])
        return [feature]


    # Override the customization process before performing synchronization from NCE to devices.
    def preSyncToNe(self, request, aoccontext=None):
        self.logger.info('preSyncToNe start.')
        dataIn = request

         # Obtain the data path for service registration.
        regPath = dataIn.regPath

        # Obtain the feature name.
        featureName = dataIn.feature

# Obtain the transaction ID.
        transId = dataIn.transId
        self.logger.info("preSyncToNe begin regPath ={}, feature={}, transId={}", regPath, featureName, transId)

        # Obtain the inconsistent data returned after service processing.
        diffDatas = dataIn.diffDatas

        # Obtain the capacity of the inconsistent data list.
        diffSize = len(diffDatas)

        # If the inconsistency is null or the inconsistent data capacity is 0, no inconsistency exists and the
function directly returns null.
        if diffDatas is None or diffSize == 0:
            return None

        # Customize the data used for data consistency verification.
        dataOut = DiffDataOutMsg()

        # Traverse the inconsistent data list and process the inconsistent data.
        for data in diffDatas:
            # Obtain the inconsistent data in XML format.
            diffXml = data

            # Process inconsistent data.
            diffDataOut = diffData()

            # Whether the inconsistent data needs to be separately submitted to southbound devices. The
value **false** indicates that the configuration does not need to be separately submitted to the southbound
devices.
            diffDataOut.singleMsg = False
            diffDataOut.diffData = diffXml
            dataOut.diffDatas.extend([diffDataOut])
        self.logger.info('preSyncToNe end.')
        return dataOut
```

NOTE

If the new and old devices managed by the same SND package support different features, an error will occur during synchronization. For example, an SND package supports management of both features A and B, the old device supports only feature A, and the new device supports features A and B. When the system initiates synchronization, it attempts to synchronize both features A and B. As a result, the system reports an error indicating that the synchronization is abnormal for the old device.

5.2.3.5 Managing Resources

You can manage the driver configuration file **cli-driver.properties** and other resources of the software package.

For CLI SND packages, you need to customize **resources/cli-driver.properties** based on the characteristics of the target CLI for adaptation.

```
#
# Copyright (c) Huawei Technologies Co., Ltd. 2019. All rights reserved.
#
protocol=Ssh
userModeCommand=
defaultParser=basicPatternParser
keepAliveCommandParser=basicPatternParser
privilegedModeCommandAfterAbortCommand=false
userModePrompt=^<.*>$
privilegedModePrompt=^\\[.*\\]$|.*ontinue\\? \\[Y\\/N\\]
promptSkipRegex=(?s).*((Error)|(WARNING)|(MINOR)|(INFO)):[ ]{1}[^@prompt@.]{1,}(@prompt@){1}$
privilegedModeCommand=
userNameResponse=Login:
passwordResponse=Enter password:
initCommand=display clock
keepAliveCommand=dis clock utc
cliTxCreateCommand=return\nsystem-view\n
# For Huawei devices that support two-phase delivery, set the cliTxCommitCommand,
cliTxCancelCommand, and configModeCommand parameters based on the following example. For third-
party devices that support two-phase delivery, complete the configuration based on the device
implementation.
cliTxCommitCommand=commit \n
cliTxCancelCommand=clear configuration candidate\n
configModeCommand=\n
cliTxSaveCommand=return \n
abortCommand=return
cliTxCancelRetryTimes=3
connectionTimeout=20000
keepAliveInterval=60000
keepConnectedInterval=20000
cliType=
logData=false
errorMessagePattern=^Error:?[ ]{1}.*
ignoreMessagePattern=randomvalueshould#not#match
warningMessagePattern=^((WARNING)|(MINOR)|(INFO)):[ ]{1}.*
maxNumberReadsChannels=1
maxResponseSize=400
showConfigCommand=return\ndisplay current-configuration
showConfigSkipRegex=^(--|echo|#).*$
resetCommand=return
submodeCommand=system-view ${context}
operModeCommand=display
partialReadCommand=return\nsystem-view\n${context}\ndisplay this
indentationSequence=\u0020
exitSubmodeCommand=quit
submodeStartSequence=@indent@
submodeEndSequence=@dedent@
negateCommand=undo\u0020
shutdownCommand=\u0020shutdown
operationalDataReadCommand=/display ${context}
contextAccessType=readDatastore
contextSupportsSinglelineCommand=false
defaultDeleteOperationWithValue=false
supportsSinglelineCommand=false
cliInteractionMatchKey=Warning.*(base on this interface, continue| BGP process will be deleted. Continue|
BGP peer will be deleted. Continue| This operation will reset the peer session. Continue?| The destination
address and the mask do not match).*;^The password needs to be changed\\. Change now\\? \\[Y\\/N\\].*
cliInteractionGoOnCommond=Y;N
judgeContinueCondition=.*--- More ---*
partialReadSubmode=false
continueReadingCommand=\u0020
```



```
changePasswordPrompt=^The password needs to be changed\\. Change now\\? \\[Y\\/N\\].*
skipChangePasswordCmd=N
```

5.3 Verifying the SND Package

5.3.1 Verifying the YANG Files

After an SND package is developed, you need to verify the validity of the YANG files in the SND package.

Step 1 Decompress **yang-offline-util.zip**.

Step 2 Copy the YANG model files in the **yang** directory of the software package to the path where **yang-offlineutil.zip** is decompressed.

Step 3 Run the following command to check whether the YANG files are correct:

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar validate console path .

If the command output is empty, the YANG file format is correct.

```
C:\Users\demo>cd yang-offline-util
C:\Users\demo\yang-offline-util>Java -jar yang-offline-util.jar validate console path .
C:\Users\demo\yang-offline-util>
```

----End

5.4 Compiling an SND Package

Perform the following operations to develop a software package:

Step 1 In the PyCharm window, click **Terminal** at the bottom of the window to open the CLI.

Step 2 Run the following command to move the exported private key file to the **key** directory in the software package path:

(dem) C:\Users\demo\PycharmProjects\dem>copy path\to\privkey.asc .\key
\privkey.asc

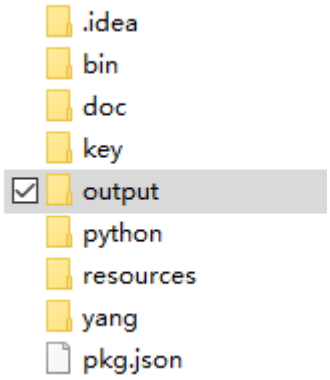
Step 3 Run the **makeFile.bat** file in the **bin** directory in the software package path to develop the software package.

Run the following commands in the CLI:

(dem) C:\Users\demo\PycharmProjects\dem>cd bin

(dem) C:\Users\demo\PycharmProjects\dem\bin>makeFile.bat

Step 4 If the following information is displayed in the command output, the software package is developed successfully. In this case, you can go to the **output** directory in the software package path to obtain the software package and its signature file.




```
2019-11-07 14:18:00,812 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Execute success
2019-11-07 14:18:00,819 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Clean dir success
2019-11-07 14:18:01,127 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] - [Sign]
Key length:3072
2019-11-07 14:18:01,180 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] -
[Sign]Generate signature file success.
2019-11-07 14:18:01,181 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Sign: Execute success
```

----End

Troubleshooting

Question: What can I do if an error message "NO JAVA_HOME" is displayed when I compile a software package?

Answer: You need to install the JDK software. You can download the JDK software from the **official website** and install it. JDK 1.8 is recommended.

6 Developing an SSP Package (EasyMap)

An SSP package provides the following functions: EasyMap, RPC, and Discover. EasyMap: decomposes network-layer services to NE-layer services. After devices are managed, the network layer directly delivers services to the devices.

This section uses a simple example to describe how to use the downloaded default SSP package to develop an EasyMap SSP package that implements the forward decomposition of network-layer services into NE-layer services. In this example, a user defines a L3VPN network service to be delivered to the NE8000 M8 (functioning as a PE) and opens the network service in the northbound direction.

Sample code packages for open programming in typical scenarios have been uploaded to the AOC. You can log in to the community to query and download sample code packages as needed. Development based on sample code packages improves efficiency and reduces difficulty. If no suitable sample code package is found, you can download the default SSP software package from the open programming environment for development.

When developing an SSP package, protect key information, such as user passwords, login tokens, sessions, and other personal data. Do not output such information in logs.

6.1 Creating an SSP Package Template

Step 1 Choose **Package Repo** from the main menu. Then, choose **Package Management** from the navigation pane, and click **Create Template** on the displayed page.

Step 2 In the displayed **Add** dialog box, set the attributes of the SSP package.

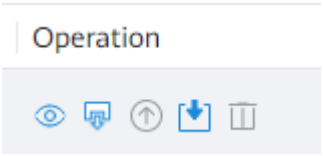
Set the attributes of the SSP package according to **Table 6-1**.

Table 6-1 Attributes of the L3VPN SSP package

Attribute	Value
Name	L3VPN
Version	1.0.0
Provider	HUAWEI
Package type	Specific Service plugin
Service name	l3vpn
Service point	l3vpn
Mapping type	python+template

Step 3 Click **OK**. In the dialog box that is displayed indicating that the software package is added successfully, click **OK**. The new software package is displayed in the software package list.

Step 4 Click in the **Operation** column of the software package record to download the software package to the local PC.



----End

6.2 Developing an SSP Package

6.2.1 Editing the SSP Package Configuration File

Step 1 Open PyCharm (the community edition is used as an example) and click **Create New Project**. The **New Project** dialog box is displayed. Expand the **Project Interpreter** area, confirm the configuration, and click **Create**. If a project has been created, you can directly open it in PyCharm.

Step 2 Decompress the downloaded software package to the directory where the project is located.

Step 3 In the navigation pane of the IDE, double-click the **pkg.json** file.

Step 4 Modify the parameters according to the following table.

Table 6-2 Parameter values in the **pkg.json** file

Parameter	Description	Value	Mandatory/ Optional
name	Software package name.	The value is user-defined and is a string of 1 to 64 characters consisting of digits, uppercase letters, lowercase letters, underscores (_), and hyphens (-).	Mandatory
version	Software package version number.	The version number must be in the xxxx.xxxx.xxxx format. The value of each *x* ranges from 0 to 9, and each segment supports a maximum of four *x*s.	Mandatory
description	Software package description.	The value is user-defined and has a maximum of 128 characters.	Optional
package-type	Package type.	Three enumerated values are available: **Specific Service plugin**, **Generic NE driver**, and **Specific NE driver**.	Mandatory
group	Package group information, which is used for sandbox process isolation.	The value is user-defined and has a maximum of 128 characters. If this parameter is not set, the default value **global** is used.	Optional
producer	Provider of the software package.	For SND and GND packages, only the packages with this parameter being **huawei** are supported currently.	Mandatory
augment-isolation	Model range ID.	The value is a Boolean value and can only be **true** or **false**. In the DCN single-NE scenario, set this parameter to **true**. In other scenarios, set this parameter to **false**.	Optional

Parameter	Description	Value	Mandatory/ Optional
nce-min-versions	Minimum version of the OPS to which the package is compatible with.	The value is in the format *x.x.x*. If the most significant version number is changed, an incompatible change occurs. The value **1.0.0** indicates that compatibility is maintained with all versions in the range from 1.0.0 (included) to 2.0.0 (excluded). You can enter multiple version segments, with one segment in each row. If this parameter is left empty, compatibility is maintained with all versions.	Optional
package-dependencies	Dependency on third-party packages.	If this parameter is set, the following parameters must be set: Third-party package name: Name of the third-party package on which the software package depends. The value can contain a maximum of 64 characters. Version: version number in x.x.x format.	Optional
snd-type	Type of the SND package.	The value **system** indicates a built-in SND package. Built-in SND package of the system. You are advised not to change the value of this parameter. If this parameter is not set, the default value **specific** is used.	Optional
snd-id	SND package ID. This parameter needs to be set only for SND packages. The value must be unique for each SND package.	The value is user-defined and has a maximum of 128 characters. This parameter is mandatory for SND packages and is optional for other types of packages.	Optional

Parameter	Description	Value	Mandatory/ Optional
devices	Information about the device matching the driver, which is specific to SND and GND packages. This parameter describes the software package and is not used for device management.	If you set this parameter, the following parameters must also be set: **vendor**: device vendor. The value can contain a maximum of 128 characters. **device-type**: device type. The value can contain a maximum of 128 characters. **device-version**: device version. The value can contain a maximum of 128 characters.	Optional
service-name	Service name. This parameter is available only to SSP packages.	The value is user-defined and has a maximum of 64 characters. This parameter is mandatory for SSP packages. The value must be unique for each SSP package. For other types of software packages, you do not need to set this parameter.	Optional

Parameter	Description	Value	Mandatory/ Optional
hooks	Callback mapping information. The hooks combination must be unique in a single package, but can be the same in different packages. Multiple hooks can be set when multiple functions are developed together.	If you set this parameter, set the following parameters: **type** (mandatory): callback type. For EasyMap, set **type** to **mapping**. For RPC, set **type** to **service-rpc**. For Discover, set **type** to **discover**. **key** (mandatory): callback key value, which can contain a maximum of 128 characters. **java-class-name** (optional): callback mapping class. This parameter is specific to Java. The value can contain a maximum of 256 characters. **python-class-name** (optional): callback mapping class. This parameter is specific to Python. The value contains a maximum of 256 characters. **groovy-class-name** (optional): callback mapping class. This parameter is specific to Groovy. The value contains a maximum of 256 characters. **template** (optional): template name. The value contains a maximum of 256 characters.	Optional

Check the file content. When the EasyMap network-layer service is decomposed into the NE-layer service, set **type** to **mapping**.

An example is as follows:

```
{
    "name": "L3VPN",
    "version": "1.0.0",
    "description": "l3vpn",
    "package-type": "ssp",
    "producer": "huawei",
    "service-name":"l3vpn",
    "nce-min-versions":[
        "1.0.0"
    ],

    "hooks": [
     {
       "type": "mapping",
       "key": "l3vpn",
       "python-class-name": "l3vpn.l3vpn.L3VPNService"
     }
```



```
  ]
}
```

----End

6.2.2 Compiling a Service YANG Model

In the navigation pane of the IDE, double-click the .yang file in the **yang** directory to compile a YANG model. By default, the YANG file name corresponds to the value of **service-name** in the **pkg.json** file, and the module name in the YANG model corresponds to the value of **service-name** in the **pkg.json** file. The value of **app:application-definition** in the YANG model corresponds to the value of **key** under **hooks** in the **pkg.json** file, that is, the service point.

Currently, only service points of the list and container types can be mounted to the root node of the application. If a service point is defined on a parent node, no service point can be defined on a child node.

Table 6-3 lists the parameters related to configuration delivery, which need to be defined in the YANG model.

Table 6-3 Input parameters

Type	Description	Parameter	Value
VPN service ID	Specify the device to which a VPN service is to be delivered. The device name and VPN name need to be extracted.	deviceName	PE1
		vrfName	5G-RAN
VPN parameters	Configure the RT and RD for the VPN to filter routing information. The RT and RD need to be extracted.	rd	100:01:00
		rt	100:11:00
Interface parameters	An IPv4 address needs to be configured for the sub-interface bound to the VPN. The sub-interface name, sub-interface description, sub-interface IP address, and sub-interface mask need to be extracted.	ifName	GigabitEthernet0/5/0.1
		description	connect to pe2
		ip	20.1.2.9
		Subnet mask	255.255.255.0
BGP peer parameters	Configure the IP address of the BGP peer. The IP address of the peer needs to be extracted.	peerAddress	2.2.2.2

The following is an example of the YANG model in the L3VPN SSP package:

```
module l3vpn {
    //Module name, which corresponds to the service name.
    namespace "http://example.com/l3vpn";
    prefix "l3vpn";

    import huawei-ac-applications {
        prefix app;
    }
    import ietf-inet-types {
        prefix inet;
    }

    description
        "The module for L3VPN example.";

    revision 2020-11-05 {
        description "Initial revision.";
    }

    augment "/app:applications"{
        list l3vpn {
            app:application-definition "l3vpn";
             /*app:application-definition defines a service point that can be identified by the system.
            The service point must be the same as the value of the key field in the hooks parameter in the
pkg.json file. */
            key "deviceName vrfName";
            //Service ID. In this example, the combination of the device name and VPN name is used as a key.
            leaf deviceName {
              type string {
                length "1..512";
              }
            }

            leaf vrfName {
              description "Defines a type of service component identifier.";
              type string {
                length 1..255 {
                  description "VPN Routing/Forwarding instance name, support 1-255 characters.";
                }
              }
              mandatory true;
            }
           //RD and RT defined for the VPN service in the YANG model.
           leaf rd {
              description "BGP route distinguisher";
              type string;
            }

            leaf rt {
              description "Route target extended community as per RFC4360";
              type string;
            }

           //Sub-interface name defined in the YANG model.
            leaf ifName {
              type string {
                length "0..255";
              }
            }
           //Sub-interface description defined in the YANG model.
            leaf description {
              type string {
                length "0..242";
              }
            }
           //Sub-interface IP address defined in the YANG model.
            leaf ip {
              type inet:ipv4-address;
```



```
        }
        //Sub-interface mask defined in the YANG model.
        leaf subnet {
          type inet:ipv4-address;
        }
        //Peer IP address defined in the YANG model.
        leaf peerAddress {
          type inet:ipv4-address;
        }
      }
    }
}
```

If you need to use functions such as user-defined data encryption, high-risk operation definition, YANG model blacklist masking, YANG model verification exemption, and user-defined model permission, refer to the following information. The sample software package does not involve these configurations.

Data Encryption

You can edit the **security.xml** file in the **yang** directory to determine whether to encrypt the data corresponding to the nodes in the YANG model.

If data encryption is enabled, data is encrypted when being saved to the database and is decrypted when being read from the database; when the configuration is delivered, the encrypted data fields are displayed as ******; when the northbound RESTCONF v1 API queries the encrypted data field, the value in ciphertext is returned; during synchronization, encrypted data fields are not synchronized; during data consistency verification, encrypted data fields are not delivered.

Note: When data is saved to the database, the key field of the list cannot be encrypted. In addition, only data of the string type can be encrypted.

Table 6-4 Parameters in the **security.xml** file

Node	Parent Node	Attribute	Description
security-node	/	N/A	Root node.
path	security-node	-	Security node path. Example: /a:b/c/d
-	-	type	Node type: leaf leaf-list typedef: customized type By default, the node is of the leaf type.

If the node type is leaf, only the <path> label needs to be defined. Generally, you do not need to set the attribute. The data corresponding to each <path> is encrypted. The value of **path** must start with the top-level node **<moduleName>:<nodeName>**, where *<moduleName>* is the module name of the

YANG file and *<nodeName>* is the name of the top-level node in the module. The top-level node of the path can be followed by child nodes, which are separated by a slash (/). An example is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<security-node>
   <path>/db-security-test:security-top/security-sec/security-sec-leaf-list</path>
   <path type="typedef">/db-security-test:security-len-typedef-leaf</path>
   <path>/db-security-test:security-top/security-list/security-list-leaf</path>
   <path>/dbtest:nodes/dbtest:node/db-security-test:augment-cont/augment-cont-security-leaf</path>
   <path>/dbtest:nodes/dbtest:node/db-security-test:augment-security-leaf</path>
</security-node>
```

High-Risk Operations

You can edit the **risk_operation.xml** file in the **yang** directory to define whether the RPC operation in the YANG model is a high-risk operation. When you perform a high-risk operation on the GUI, a confirmation dialog box is displayed. No confirmation dialog box will be displayed for non-high-risk operations.

All RPC operations are defined in the <rpcs> label. Each RPC high-risk operation corresponds to one <rpc> label. The value of the <rpc> label is in the format moduleName:rpcName. An example is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<risk-operation>
   <rpcs>
      <rpc>huawei-aaa:reactivateUser</rpc>
      <rpc>huawei-aaa:changeMyPassword</rpc>
      <rpc>huawei-aaa:changeMyIdleTimeout</rpc>
   </rpcs>
 </risk-operation>
```

Blacklist

You can edit the **blacklist.xml** file in the **yang** directory to define whether modules in the YANG model are shielded on the GUI.

The modules to be shielded are defined in <module>, in which the module name and module revision need to be set. An example is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<blacklist>
  <module>
     <name>huawei-cfg</name>
     <revision>2020-03-18</revision>
  </module>
  <module>
     <name>huawei-dhcp</name>
     <revision>2020-04-29</revision>
  </module>
</blacklist>
```

YANG Model Verification Exemption

You can edit the **multiRevisionModule.xml** file in the **yang** directory to define whether modules in the YANG model need to be verified during the upgrade.

Verification-free modules are defined in <modules>. Each module corresponds to one <module> label, which is set to the module name. An example is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<modules>
```



```
  <module>ietf-yang-types</module>
  <module>ietf-inet-types</module>
</modules>
```

User-Defined Model Permission

By writing a permission control file, you can customize operation permissions on nodes in YANG models. After a software package is successfully installed, the permission information is dynamically injected to the OPS. The user management function allows users with different responsibilities to be granted with appropriate permissions, preventing unauthorized and insecure operations. Permissions on software package models are automatically assigned to the default role whereas permissions assigned to user-defined roles need to be configured manually.

You can add the **permission.json** file to the **resources** directory and customize permissions on nodes in YANG models. If you have not written the **permission.json** file, the OPS automatically generates a level-0 permission control file for each module based on the YANG file during software package loading. The permission control file includes the create, delete, read, update, and execute permissions.

The OPS supports permission-based operations on container and list nodes in YANG models. You can define permissions on these nodes as needed. Customization of permissions at levels 0, 1, and 2 is supported. Level 0 indicates the permission control at the module level. If a user does not match permissions, the OPS executes permission control at level 0 by default. At level 1, a container or list node has only one level and does not have subnodes. At level 2, a container or list node has two levels and subnodes. During permission control, the OPS matches permissions based on the longest match rule. That is, permissions at a deeper level are matched first.

```
{
  "modules": [
  {
     "module-name":"hbng",
     "operations": [
     {
        "uri-pattern":"containerA",
        "method":" create/delete/read/update "
     },
     {
        "uri-pattern":"listA",
        "method":" create/delete/read/update "
     },
     {
        "uri-pattern":" listA/listC",
        "method":" create/delete/read/update"
     },
     {
        "uri-pattern":"resetStatictic",
        "method":"exec"
     }
     ]
  }
  ]
}
```

6.2.3 Developing the SSP Package Code

In the navigation pane of the IDE, double-click the .py file in the Python directory and write Python code to implement the EasyMap function logic.

Batch template operations and batch device configuration operations have a large impact scope. You are advised to carefully evaluate risks during code development. You can throw exceptions in UserException mode and view the exception information on the GUI.

In addition, if a service injection package provides general-purpose SDK capabilities or provides internal service interfaces, you can directly call the interfaces provided by the service injection package to quickly implement functions, improving development efficiency.

To implement the EasyMap function, derived classes need to inherit the NcsService base class and override the ncs_map method to implement the service logic.

NOTE

In an SSP package, the combination of a user-defined package name and class name must be globally unique. Otherwise, the package will fail to be activated.

The following is an example of the code for the L3VPN SSP package:

```
# Mandatory. NcsService is a parent class provided by the system for SSP packages to inherit and needs to be overridden to map service YANG data to Jinja2 templates. The devicemgr SDK API provides some interfaces for querying device information.
from aoc import NcsService, devicemgr
# The datastore SDK API provides interfaces for reading and writing database data.
from aoc import datastore
# AOCException is an exception class defined by the system, which displays user-defined exceptions on the GUI.
from aoc.exception.aocexceptions import AOCException

# L3VPNService inherits NcsService and overrides the ncs_map method.
class L3VPNService(NcsService):
    def ncs_map(self, request, aoccontext=None, template=None):
        l3vpn_dict = self.xmltodict(request.xml)

        # Check whether the main interface exists.
        self.checkInterfaceIsExist(request, aoccontext)

        # Update the VPN description.
        self.updateDes(l3vpn_dict)
        self.logger.info(str(l3vpn_dict))

        # Fill the service data defined by the service YANG into the Jinja2 template.
        res = self.render('l3vpn/servicepoint.j2', l3vpn_dict)
        self.logger.info(res)
        return res


    # Update the VRF description.
    # To obtain node data in the YANG model, you can use the x.y.z or x['y']['z'] method to operate model data more conveniently.
    def updateDes(self, l3vpn_dict):
        if '3G' in l3vpn_dict['l3vpn']['vrfName'] :
            l3vpn_dict.update({"vrfDes": '3G'})
            l3vpn_dict['l3vpn'].update({"description": '3G'})


    # Check whether the main interface exists.
    def checkInterfaceIsExist(self, request, aoccontext):
        device_id = devicemgr.query_neid(request.xmldictnode.l3vpn.deviceName)
        subIfName = request.xmldictnode.l3vpn.ifName
        pointIndex = subIfName.find('.')
        parentName = subIfName[0:pointIndex]
        self.logger.info('deviceId:%s, parentName:%s' % (device_id, parentName))
        if devicemgr.is_snd(device_id, 'NE8000M8SPC300_SND') or devicemgr.is_snd(device_id, 'NE8000_M8'):
            path = "huawei-ac-nes:inventory-cfg/nes/ne/" + device_id + "/huawei-ifm:ifm/interfaces/"
```

```
        # Query data in the configuration database.
        output = datastore.read_datastore_rdb(aoccontext, path)
        asnindex = str(output).find(parentName)
        if asnindex == -1:
            raise AOCException(parentName + ' not exist in device!')


# Add a Jinja2 template filter to map services to different devices in the Jinja2 template.
def add_filters(self):
    result = {"is_snd": self.is_snd}
    return result


def is_snd(self, nename, sndid):
    # Query the device ID by device name.
    neid = devicemgr.query_neid(nename)
    # Determine the SND package used by the device.
    return devicemgr.is_snd(neid, sndid)
```

6.2.4 Developing a Jinja2 Template

There are two approaches to developing a Jinja2 template:

- Reverse method: This method is applicable to scenarios where real devices have been managed and can be implemented in the following ways:
  - Obtaining packets through dry-run: Obtain NETCONF packets through dry-run, copy the packets to the Jinja2 template, and replace the input parameter values with parameter names.
  - Exporting data templates through inconsistency discovery: After devices are managed, synchronize device configurations, configure the required commands on the devices, and export the XML data template corresponding to the commands based on the inconsistency discovery capability for NE configurations. Then export the required YANG template on the device configuration page and compare it with the XML data template. In the YANG template, retain only the configuration to be delivered and delete the other configurations. Then, replace the input parameter values with parameter names.
- Forward method: This method is applicable to scenarios where no real device is available and Jinja2 templates are developed based on device commands and YANG models.

The following describes how to use the reverse method.

Exporting Data Templates Through Inconsistency Discovery

The procedure is as follows:

Step 1 Choose **Device Configuration** > **Device Configuration** from the main menu. On the **Device Configuration** page that is displayed, select a device, click **Synchronize**, and select **Synchronize from Device**.

Step 2 Select the device and click **Discover Inconsistencies**.

Step 3 After inconsistency discovery is complete, the value of **Sync Status** changes to **Discovered**. Click **View Inconsistencies** in the **Operation** column to access the inconsistency display page. Then click **Export Template**.

Step 4 On the **Device Configuration** page, click **Edit** in the **Operation** column of the corresponding device.

Step 5 Click the **Export** icon above the model node on the left. The export page is displayed.

Step 6 Select **huawei-ifm**, **Huawei-network-instance**, and **huawei-bgp**, enable **Merge template**, select **Structure only**, and click **OK**.

Step 7 Open the exported template and compare it with the template exported through inconsistency discovery. Retain only the configuration to be delivered and delete the other configurations. Then, replace the input parameter values with parameter names. Add the edited content to the **template>l3vpn>NE8000M8.j2** file.

```
<bgp xmlns:x="urn:huawei:yang:huawei-bgp"  x:tag="no_delete">
    <global>
        <yang-enable>true</yang-enable>
    </global>
</bgp>
<network-instance xmlns="urn:huawei:yang:huawei-network-instance">
    <instances>
        <instance>
            <name>{{l3vpn.vrfName}}</name>
            {%- if l3vpn.vrfDes %}
            <description>{{vrfDes}}</description>
            {%- endif %}
            <afs xmlns="urn:huawei:yang:huawei-l3vpn">
                <af>
                    <type>ipv4-unicast</type>
                    <route-distinguisher>{{ l3vpn.rd }}</route-distinguisher>
                    <tunnel-policy>LDP</tunnel-policy>
                    <vpn-targets>
                        <vpn-target>
                            <value>{{ l3vpn.rt }}</value>
                            <type>export-extcommunity</type>
                        </vpn-target>
                        <vpn-target>
                            <value>{{ l3vpn.rt }}</value>
                            <type>import-extcommunity</type>
                        </vpn-target>
                    </vpn-targets>
                </af>
            </afs>
            <bgp xmlns="urn:huawei:yang:huawei-bgp">
                <base-process>
                    <afs>
                        <af>
                            <type>ipv4uni</type>
                            <ipv4-unicast>
                                <import-routes>
                                    <import-route>
                                        <protocol>direct</protocol>
                                        <process-id>0</process-id>
                                    </import-route>
                                    <import-route>
                                        <protocol>static</protocol>
                                        <process-id>0</process-id>
                                    </import-route>
                                </import-routes>

                            </ipv4-unicast>
                        </af>
                    </afs>
                    <peers>
                        <peer>
                            <address>{{ l3vpn.peerAddress }}</address>
                            <remote-as>100</remote-as>
                        </peer>
```



```
            </peers>
          </base-process>
        </bgp>
      </instance>
    </instances>
</network-instance>
<ifm xmlns="urn:huawei:yang:huawei-ifm">
    <interfaces>
      <interface>
        <name>{{ l3vpn.ifName }}</name>
        <type>GigabitEthernet</type>
        <description>{{ l3vpn.description }}</description>
        <vrf-name>{{ l3vpn.vrfName }}</vrf-name>
        <ipv4 xmlns="urn:huawei:yang:huawei-ip">
          <addresses>
            <address>
              <ip>{{ l3vpn.ip }}</ip>
              <type>main</type>
              <mask>{{ l3vpn.subnet }}</mask>
            </address>
          </addresses>
        </ipv4>
      </interface>
    </interfaces>
</ifm>
```

Step 8 Map the service to devices of different models. Compile the logic for invoking the Jinja2 template of the corresponding device model, and add the edited content to the **template>l3vpn>servicepoint.j2** file.

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
    <nes>
      <ne>
        <neid>{{l3vpn.deviceName| to_ne_id}}</neid>
        {%- if l3vpn.deviceName | is_snd('NE8000M8SPC300_SND')  %}
          {% include 'l3vpn/NE8000M8.j2' %}
        {% elif l3vpn.deviceName | is_snd('NE8000M8_SND') %}
          {% include 'l3vpn/NE8000M8.j2' %}
        {% endif %}
      </ne>
    </nes>
</inventory-cfg>
```

----End

Obtaining Packets Through Dry-run

The procedure is as follows:

Step 1 Choose **Device Configuration** > **Device Configuration** from the main menu.. Go to the **Device Management** page and click the **Modify** button on the device to be configured.

Step 2 In the **Device Management** navigation pane, click **huawei-ifm**. In the **Interface** area, click **Add**, enter the interface name, and click **Create**.

Step 3 In the **Device Management** navigation pane, click **huawei-l3vpn**. In the **l3vpnInstance** area, click **Add**, enter the VRF name, and click **Create**.

Step 4 Click **Dry-Run** in the upper right corner to view and copy the NETCONF packets to be delivered to the device.

Step 5 Paste the copied NETCONF packet content to the **template>l3vpn>NE8000M8.j2** file in the directory of the software package.

Step 6 Modify the Jinja2 template based on service requirements. If the parameters are obtained from the northbound input, change the parameters to variables. If a parameter is set to a fixed value, use a constant.

Step 7 Map the service to devices of different models. Compile the logic for invoking the Jinja2 template of the corresponding device model, and add the edited content to the **template>l3vpn>servicepoint.j2** file. The Jinja2 template content is similar to that of the template exported through inconsistency discovery.

After the service instance is deleted, if key configurations such as main interfaces of SSP packages are deleted, service security issues may occur. You can add the **no_delete** or **no_delete_nested** attribute based on the service logic so that the marked data will not be deleted upon the deletion of the service instance.

- no_delete: Only the data source of the node marked with this label will be deleted, and the data of child nodes can be deleted.
- no_delete_nested: Only the data source of the node marked with this label and the data source of its child nodes will be deleted.

  An example is as follows:

```
<system  xmlns:ns0="urn:huawei:ac" ns0:tag="no_delete_nested">
  <inform-timeout>2</inform-timeout>
</system>
<ntpAuthKeyCfg xmlns:x="urn:huawei:ac" x:tag="no_delete">
```

----End

6.3 Verifying the SSP Package

6.3.1 Verifying the YANG Files

After an SSP package is developed, you can use the YANG model verification tool to verify the validity of the YANG files in the SSP package.

Step 1 Decompress **yang-offline-util.zip**.

Step 2 Copy the YANG model files in the **yang** directory of the SSP package to the directory where **yang-offline-util.zip** is decompressed.

Step 3 Run the following command to check whether the YANG files are correct:

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar validate console path .

If the command output is empty, the YANG file format is correct.

```
C:\Users\demo>cd yang-offline-util
C:\Users\demo\yang-offline-util>Java -jar yang-offline-util.jar validate console path .
C:\Users\demo\yang-offline-util>
```

----End

6.3.2 Performing a Unit Test

Step 1 Use the **yang-offline-util.zip** tool to generate empty NETCONF packets based on the YANG model. Run the following command.

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar generateSubtree .

If the command output is empty, the **subtree.xml** file is generated successfully.

Step 2 Open the **subtree.xml** file, and then set labels to proper values under **<application>**.

```
<l3vpn xmlns="http://example.com/L3VPN">
  <deviceName>NE8000</deviceName>
  <vrfName>5G_test</vrfName>
  <rd>100:1</rd>
  <rt>100:11</rt>
  <ifName>GigabitEthernet0/5/0.1</ifName>
  <description>wireless base</description>
  <ip>20.1.2.9</ip>
  <subnet>255.255.255.0</subnet>
  <peerAddress>2.2.2.2</peerAddress>
</l3vpn>
```

Step 3 Double-click the .py file in the **test** directory, and copy the code in the <application> label to the <l3vpn> label. The following is an example.

```
import unittest
import sys
from aoc.sys import devicemgr, datastore
from mock import Mock
 sys.path.insert(0, "../../python")
from l3vpn.l3vpn import L3VPNService

class Test(unittest.TestCase):
    xml = '''

  # Replace the following code in the l3vpn label based on actual situations.
 <l3vpn xmlns="http://example.com/L3VPN">
   <deviceName>NE8000</deviceName>
   <vrfName>4G_test</vrfName>
   <rd>100:1</rd>
   <rt>100:11</rt>
   <ifName>GigabitEthernet0/5/0.1</ifName>
   <description>wireless base</description>
   <ip>20.1.2.9</ip>
   <subnet>255.255.255.0</subnet>
   <peerAddress>2.2.2.2</peerAddress>
 </l3vpn>
  '''


   def mock_get_id(self, neName):
       if neName == 'CX600':
           return 'mock_neid:CX600'
       if neName == 'NE8000':
           return 'mock_neid:NE8000'
       return 'mock_neid:NE8000'


    def mock_sys_func(self):
       devicemgr.query_neid = self.mock_get_id
       mock_read_datastore_rdb = Mock(return_value='GigabitEthernet0/5/0')
       datastore.read_datastore_rdb = mock_read_datastore_rdb
       mock_write_datastore = Mock(return_value='success')
       datastore.write_datastore = mock_write_datastore
       devicemgr.is_snd = self.mock_is_snd


    def mock_is_snd(self, nename, sndid):
       if nename == 'mock_neid:NE8000' and sndid == 'NE8000M8_SND':
           return True
       if nename == 'mock_neid:NE8000' and sndid == 'NE8000M8SPC300_SND':
           return True
       return False
```



```
    def setUp(self):
        self.mock_sys_func()


    def test_l3vpn(self):
        result = L3VPNService().ncs_map_test(self.xml)
        print(result)
if __name__ == "__main__":
    unittest.main()
```

Step 4 Run the test code to view the generated packet and check whether the output is correct.

If the message "Process finished with exit code 0" is displayed, the unit test is successful. Otherwise, check whether the attributes in the Jinja2 template correspond to those in the service YANG model.

----End

6.4 Compiling an SSP Package

Perform the following operations to compile a software package:

Step 1 In the PyCharm window, click **Terminal** at the bottom of the window to open the CLI.

Step 2 Run the following command to move the exported private key file to the **key** directory in the software package path:

(dem) C:\Users\demo\PycharmProjects\dem>copy path\to\privkey.asc .\key \privkey.asc

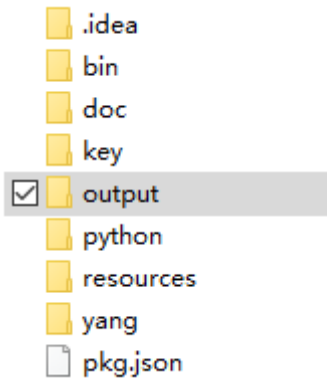
Step 3 Run the **makeFile.bat** file in the **bin** directory in the software package path to develop the software package.

Run the following commands in the CLI:

(dem) C:\Users\demo\PycharmProjects\dem>cd bin

(dem) C:\Users\demo\PycharmProjects\dem\bin>makeFile.bat

Step 4 If the following information is displayed in the command output, the software package is developed successfully. In this case, you can go to the **output** directory in the software package path to obtain the software package and its signature file.




```
2019-11-07 14:18:00,812 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Execute success
2019-11-07 14:18:00,819 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Clean dir success
2019-11-07 14:18:01,127 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] - [Sign]
Key length:3072
2019-11-07 14:18:01,180 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] -
[Sign]Generate signature file success.
2019-11-07 14:18:01,181 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Sign: Execute success
```

----End

Troubleshooting

Question: What can I do if an error message "NO JAVA_HOME" is displayed when I compile a software package?

Answer: You need to install the JDK software. You can download the JDK software from the **official website** and install it. JDK1.8 is recommended.

7 Developing an SSP Package for Service Restoration

An SSP package provides the following functions: EasyMap, RPC, and Discover. Discover re-organizes NE-layer services into network-layer services and is the reverse process of EasyMap. If NE-layer services exist before devices are managed, re-organize NE-layer services to the network layer after the devices are managed.

This section uses an example to describe how to use the downloaded default SSP package to develop a plug-in package for service restoration, which defines the logic for restoring services from the NE layer to the network layer to match services at the NE layer and network layer. SRTE has been enabled on the NE, and tunnel configurations have been delivered. SRTE service configurations on the NE are restored to the database.

Sample code packages for open programming in typical scenarios have been uploaded to the AOC. You can log in to the community to query and download sample code packages as needed. Development based on sample code packages improves efficiency and reduces difficulty. If no suitable sample code package is found, you can download the default SSP software package from the open programming environment for development.

When developing an SSP package, protect key information, such as user passwords, login tokens, sessions, and other personal data. Do not output such information in logs.

7.1 Creating an SSP Package Template

Step 1 Choose **Package Repo** from the main menu. Then, choose **Package Management** from the navigation pane, and click **Create Template** on the displayed page.

Step 2 In the displayed **Add** dialog box, set the attributes of the software package.

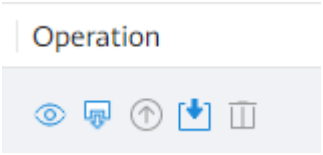
Set the attributes of the software package according to **Table 7-1**.

Table 7-1 Attributes of the software package

Attribute	Value
Name	SSP-SRTE
Version	4.0.30
Provider	huawei
Package type	Specific Service plugin
Service name	cmnetSrte
Mapping type	python+template
Service point	neTunnels

Step 3 Click **OK**. In the dialog box that is displayed indicating that the software package is added successfully, click **OK**. The new software package is displayed in the software package list.

Step 4 Click in the **Operation** column of the software package record to download the software package to the local PC.



----End

7.2 Developing an SSP Package

7.2.1 Editing the SSP Package Configuration File

Step 1 Open PyCharm (the community edition is used as an example) and click **Create New Project**. The **New Project** dialog box is displayed. Expand the **Project Interpreter** area, confirm the configuration, and click **Create**. If a project has been created, you can directly open it in PyCharm.

Step 2 Decompress the downloaded software package to the directory where the project is located.

Step 3 In the navigation pane of the IDE, double-click the **pkg.json** file.

Step 4 Modify the parameters according to the following table.

Table 7-2 Parameter values in the **pkg.json** file

Parameter	Description	Value	Mandatory/ Optional
name	Software package name.	The value is user-defined and is a string of 1 to 64 characters consisting of digits, uppercase letters, lowercase letters, underscores (_), and hyphens (-).	Mandatory
version	Software package version number.	The version number must be in the xxxx.xxxx.xxxx format. The value of each *x* ranges from 0 to 9, and each segment supports a maximum of four *x*s.	Mandatory
description	Software package description.	The value is user-defined and has a maximum of 128 characters.	Optional
package-type	Package type.	Three enumerated values are available: **Specific Service plugin**, **Generic NE driver**, and **Specific NE driver**.	Mandatory
group	Package group information, which is used for sandbox process isolation.	The value is user-defined and has a maximum of 128 characters. If this parameter is not set, the default value **global** is used.	Optional
producer	Provider of the software package.	For SND and GND packages, only the packages with this parameter being **huawei** are supported currently.	Mandatory
augment-isolation	Model range ID.	The value is a Boolean value and can only be **true** or **false**. In the DCN single-NE scenario, set this parameter to **true**. In other scenarios, set this parameter to **false**.	Optional

Parameter	Description	Value	Mandatory/ Optional
nce-min-versions	Minimum version of the OPS to which the package is compatible with.	The value is in the format *x.x.x*. If the most significant version number is changed, an incompatible change occurs. The value **1.0.0** indicates that compatibility is maintained with all versions in the range from 1.0.0 (included) to 2.0.0 (excluded). You can enter multiple version segments, with one segment in each row. If this parameter is left empty, compatibility is maintained with all versions.	Optional
package-dependencies	Dependency on third-party packages.	If this parameter is set, the following parameters must be set: Third-party package name: Name of the third-party package on which the software package depends. The value can contain a maximum of 64 characters. Version: version number in x.x.x format.	Optional
snd-type	Type of the SND package.	The value **system** indicates a built-in SND package. Built-in SND package of the system. You are advised not to change the value of this parameter. If this parameter is not set, the default value **specific** is used.	Optional
snd-id	SND package ID. This parameter needs to be set only for SND packages. The value must be unique for each SND package.	The value is user-defined and has a maximum of 128 characters. This parameter is mandatory for SND packages and is optional for other types of packages.	Optional

Parameter	Description	Value	Mandatory/ Optional
devices	Information about the device matching the driver, which is specific to SND and GND packages. This parameter describes the software package and is not used for device management.	If you set this parameter, the following parameters must also be set: **vendor**: device vendor. The value can contain a maximum of 128 characters. **device-type**: device type. The value can contain a maximum of 128 characters. **device-version**: device version. The value can contain a maximum of 128 characters.	Optional
service-name	Service name. This parameter is available only to SSP packages.	The value is user-defined and has a maximum of 64 characters. This parameter is mandatory for SSP packages. The value must be unique for each SSP package. For other types of software packages, you do not need to set this parameter.	Optional

Parameter	Description	Value	Mandatory/ Optional
hooks	Callback mapping information. The hooks combination must be unique in a single package, but can be the same in different packages. Multiple hooks can be set when multiple functions are developed together.	If you set this parameter, set the following parameters: **type** (mandatory): callback type. For EasyMap, set **type** to **mapping**. For RPC, set **type** to **service-rpc**. For Discover, set **type** to **discover**. **key** (mandatory): callback key value, which can contain a maximum of 128 characters. **java-class-name** (optional): callback mapping class. This parameter is specific to Java. The value can contain a maximum of 256 characters. **python-class-name** (optional): callback mapping class. This parameter is specific to Python. The value contains a maximum of 256 characters. **groovy-class-name** (optional): callback mapping class. This parameter is specific to Groovy. The value contains a maximum of 256 characters. **template** (optional): template name. The value contains a maximum of 256 characters.	Optional

An example is as follows:

```
{
 "name": "SSP-SRTE",
 "version": "4.0.30",
 "description": "srte for cmnet",
 "package-type": "ssp",
 "producer": "huawei",
 "service-name": "cmnetSrte",
 "group": "cmnet",
 "nce-min-versions": [
  "1.0.0",
  "2.0.0"
 ],
 "hooks": [
  {
   "type": "mapping",
   "key": "neTunnels",
   "python-class-name": "com.huawei.cmnet.srte.aoc_ncs_srte.AocNcsSrteService"
  },
  {
   "type": "discover",
   "key": "neTunnels",
```



```
    "python-class-name": "com.huawei.cmnet.srte.srte_restore.SrteRestoreService"
  }
 ]
}
```

----End

7.2.2 Compiling a Service YANG Model

In the navigation pane of the IDE, double-click the .yang file in the **yang** directory to compile a YANG model. By default, the YANG file name corresponds to the value of **service-name** in the **pkg.json** file, and the module name in the YANG model corresponds to the value of **service-name** in the **pkg.json** file. The value of **app:application-definition** in the YANG model corresponds to the value of **key** under **hooks** in the **pkg.json** file, that is, the service point.

Currently, only service points of the list and container types can be mounted to the root node of the application. If a service point is defined on a parent node, no service point can be defined on a child node.

The YANG model used for forward network service provisioning is the same as that used for reverse service restoration. The YANG model is constructed based on the northbound input of the service model. Each module and node in the YANG model generate a northbound UI for you to set parameters.

Table 7-3 lists the parameters related to configuration delivery, which need to be defined in the YANG model.

Table 7-3 Input parameters

Type	Description	Parameter	Value
SRTE tunnel configuration	Create a tunnel interface and configure tunnel parameters and protocols. • **deviceName**: device to which the SRTE service is to be delivered. • **tunnelName** and **tunnel-id**: name and ID of the SRTE tunnel. **description**: tunnel description. • **bandwidth**: tunnel bandwidth. • **setup-priority**: setup priority of a tunnel.	deviceName	PE1
		signal-protocol	segment-routing
		tunnel-id	4000
		destIp	10.10.110.114
		tunnelName	Tunnel4000
		description	SRTETunnel
		statistic	true
		reserved-for-binding	true
		bandwidth	ct0 1
		setup-priority	0
		pce-delegate	active
		hotstandby	hotstandby

Type	Description	Parameter	Value
Tunnel interface configuration	• **mtu**: maximum transmission unit (MTU) of the tunnel interface. • **ip address**: IP address of the interface. The IP address can be configured in two formats: IP address +subnet or unnumbered. In unnumbered mode, the tunnel is directly bound to an interface. In this example, the tunnel is bound to interface Tunnel2009.	mtu	46
		ip-address	unnumbered interface Tunnel2009
MPLS BFD configuration	• **ability**: whether to enable the BFD function. • **mode**: BFD mode. • **min-tx-interval**: minimum interval at which BFD packets are sent. • **min-rx-interval**: minimum interval at which BFD packets are received.	ability	enable
		mode	seamless
		min-tx-interval	3
		min-rx-interval	3
IGP configuration	• **shortcut-type**: The tunnel is used to perform enhanced SPF calculation. • **igpMetricValue**: metric of the tunnel in SPF calculation for IGP shortcut.	shortcut-type	3
		igpMetricValue	3
Tunnel path configuration	• **hop-limit**: maximum number of hops along a tunnel. • **explicit-path-name**: name of a tunnel path. • **path-type**: primary/ backup mode of the tunnel path.	hop-limit	20
		explicit-path-name	expath_0110/ expath_0111
		path-type	expath_0111 secondary

The following is an example of part of the SRTE tunnel configuration in the YANG model in the cmnetSrte software package:


```
module cmnetSrte {
    ...
   grouping ne-tuls {
      list neTunnel {
      app:application-definition "neTunnels";
         key "deviceId tunnelName";
         uses srte-tunnel;
      }
   }

   augment "/app:applications" {
      uses ne-tuls;
}

    // SRTE tunnel configuration.
    grouping srte-tunnel {
      leaf deviceId {
         type string {
            length "1..max";
         }
         description "device id";
      }
      leaf tunnelName {
         type string {
            length "1..255";
         }
         mandatory true;
      }
      leaf tunnelId {
         type uint32 {
            range "1..65535";
         }
      }
      leaf destIp {
         type string;
         mandatory true;
      }
      leaf ipAddressType {
         type ip-types;
      }
      leaf description {
         type string {
            length "0..1024";
         }
      }
      leaf controlMode {
         type control-modes;
         mandatory true;
      }
...
```

Data Encryption

You can edit the **security.xml** file in the **yang** directory to determine whether to encrypt the data corresponding to the nodes in the YANG model.

If data encryption is enabled, data is encrypted when being saved to the database and is decrypted when being read from the database; when the configuration is delivered, the encrypted data fields are displayed as ******; when the northbound RESTCONF v1 API queries the encrypted data field, the value in ciphertext is returned; during synchronization, encrypted data fields are not synchronized; during data consistency verification, encrypted data fields are not delivered.

Note: When data is saved to the database, the key field of the list cannot be encrypted. In addition, only data of the string type can be encrypted.

Table 7-4 Parameters in the **security.xml** file

Node	Parent Node	Attribute	Description
security-node	/	N/A	Root node.
path	security-node	-	Security node path. Example: /a:b/c/d
-	-	type	Node type: leaf leaf-list typedef: customized type By default, the node is of the leaf type.

If the node type is leaf, only the <path> label needs to be defined. Generally, you do not need to set the attribute. The data corresponding to each <path> is encrypted. The value of **path** must start with the top-level node **<moduleName>:<nodeName>**, where *<moduleName>* is the module name of the YANG file and *<nodeName>* is the name of the top-level node in the module. The top-level node of the path can be followed by child nodes, which are separated by a slash (/). An example is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<security-node>
   <path>/db-security-test:security-top/security-sec/security-sec-leaf-list</path>
   <path type="typedef">/db-security-test:security-len-typedef-leaf</path>
   <path>/db-security-test:security-top/security-list/security-list-leaf</path>
   <path>/dbtest:nodes/dbtest:node/db-security-test:augment-cont/augment-cont-security-leaf</path>
   <path>/dbtest:nodes/dbtest:node/db-security-test:augment-security-leaf</path>
</security-node>
```

High-Risk Operations

You can edit the **risk_operation.xml** file in the **yang** directory to define whether the RPC operation in the YANG model is a high-risk operation. When you perform a high-risk operation on the GUI, a confirmation dialog box is displayed. No confirmation dialog box will be displayed for non-high-risk operations.

All RPC operations are defined in the <rpcs> label. Each RPC high-risk operation corresponds to one <rpc> label. The value of the <rpc> label is in the format moduleName:rpcName. An example is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<risk-operation>
   <rpcs>
      <rpc>huawei-aaa:reactivateUser</rpc>
      <rpc>huawei-aaa:changeMyPassword</rpc>
      <rpc>huawei-aaa:changeMyIdleTimeout</rpc>
   </rpcs>
 </risk-operation>
```


Blacklist

You can edit the **blacklist.xml** file in the **yang** directory to define whether modules in the YANG model are shielded on the GUI.

The modules to be shielded are defined in <module>, in which the module name and module revision need to be set. An example is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<blacklist>
  <module>
     <name>huawei-cfg</name>
     <revision>2020-03-18</revision>
  </module>
  <module>
     <name>huawei-dhcp</name>
     <revision>2020-04-29</revision>
  </module>
</blacklist>
```

YANG Model Verification Exemption

You can edit the **multiRevisionModule.xml** file in the **yang** directory to define whether modules in the YANG model need to be verified during the upgrade.

Verification-free modules are defined in <modules>. Each module corresponds to one <module> label, which is set to the module name. An example is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<modules>
   <module>ietf-yang-types</module>
   <module>ietf-inet-types</module>
 </modules>
```

User-Defined Model Permission

By writing a permission control file, you can customize operation permissions on nodes in YANG models. After a software package is successfully installed, the permission information is dynamically injected to the OPS. The user management function allows users with different responsibilities to be granted with appropriate permissions, preventing unauthorized and insecure operations. Permissions on software package models are automatically assigned to the default role whereas permissions assigned to user-defined roles need to be configured manually.

You can add the **permission.json** file to the **resources** directory and customize permissions on nodes in YANG models. If you have not written the **permission.json** file, the OPS automatically generates a level-0 permission control file for each module based on the YANG file during software package loading. The permission control file includes the create, delete, read, update, and execute permissions.

The OPS supports permission-based operations on container and list nodes in YANG models. You can define permissions on these nodes as needed. Customization of permissions at levels 0, 1, and 2 is supported. Level 0 indicates the permission control at the module level. If a user does not match permissions, the OPS executes permission control at level 0 by default. At level 1, a container or list node has only one level and does not have subnodes. At level 2, a container or list node has two levels and subnodes. During permission control, the OPS matches permissions based on the longest match rule. That is, permissions at a deeper level are matched first.


```
{
   "modules": [
   {
      "module-name":"hbng",
      "operations": [
      {
         "uri-pattern":"containerA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":"listA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":" listA/listC",
         "method":" create/delete/read/update"
      },
      {
         "uri-pattern":"resetStatictic",
         "method":"exec"
      }
      ]
   }
   ]
}
```

7.2.3 Developing the SSP Package Code

In the navigation pane of the IDE, double-click the .py file in the Python directory and write Python code to implement the Discover function logic.

Batch template operations and batch device configuration operations have a large impact scope. You are advised to carefully evaluate risks during code development. You can throw exceptions in UserException mode and view the exception information on the GUI.

In addition, if a service injection package provides general-purpose SDK capabilities or provides internal service interfaces, you can directly call the interfaces provided by the service injection package to quickly implement functions, improving development efficiency.

To implement the Discover function, derived classes need to inherit the NcsService base class and override the Discover method to implement the service logic.

```
# (Mandatory) Import the NcsService class. NcsService is the parent class provided by aoc_api for the SSP
package to inherit.
from ipaddress import IPv4Network
from aoc import NcsService, devicemgr
from aoc.ncs.filter import ipaddr

class SrteRestoreService (NcsService):
   def discover(self, discover_input, aoc_context):
      """
      Device discovery function
      :param discover_input: input parameters for device discovery
      :param aoc_context: context
      :return: XML packet after conversion
      """
      time_begin = time.time()
      self.logger.info("[SrTeRestoreService][discover] discoverInput is: %s,"
                       " aoc_context is: %s" % (discover_input, aoc_context))

      # Obtain device information from discover_input.
      device_info_list = []
      if discover_input.deviceInfo is not None:
         device_info_list = discover_input.deviceInfo
```



```
        self.logger.info("[SrTeRestoreService][discover] device_info_list is: %s"
                         % device_info_list)

        # Read the context information and continue the processing based on the progress of the previous
round.
        discovery_id = discover_input.discoveryId
        self.cache_context = discover_input.cacheContext
        self.logger.info("start real discovery: %s---%s" % (discovery_id, time.time()))

        result = DiscoverOutput()
        # Traverse device information in the input information list.
        for device_info in device_info_list:
            device_id = device_info.deviceId

            self.logger.info("[SrTeRestoreService][discover] device_info.attributes is: %s" %
device_info.attributes)
            # Determine the number of services to be processed on the device. If the number is greater than 0
and less than or equal to the maximum value, update the maximum value.
            if device_info.attributes and self.handler_max_size >= int(device_info.attributes):
                self.handler_max_size = int(device_info.attributes)

            # Read the RDB of the device and query the total number of ifm configurations of the device.
            self.read_ifm_count_from_rdb(aoc_context, device_id)
            # get the template by device info
            template = self.find_ne_template(device_id)

            if template is None:
                raise ValueError("The device doesn't support [deviceId: %s]" % device_id)

            # Read the RDB of the device and query the ifm configurations of the corresponding device. 100
records are queried each time.
            ifm_dicts = self.read_ifm_from_rdb(aoc_context, device_id)
            # build ne template
            if ifm_dicts and ifm_dicts['interface']:
                # Process the SRTE tunnel template content and return the template content set.
                ifm_list = self.handler_template_content(ifm_dicts, aoc_context, device_id)
                self.logger.info("[SrTeRestoreService][discover] ifm_list is %s" % ifm_list)

                # Build the data to be restored.
                sr_tes_restore_template_content = {
                    "ifm_list": ifm_list,
                    "ne_id": device_id
                }

                # Perform reverse mapping on data based on the template of the corresponding NE to restore
services. Set template to the Jinja2 template for service restoration.
                tunnels = DictNode(xmltodict.parse)
                    self.render(template, sr_tes_restore_template_content),
                    encoding="utf-8", strip_whitespace=True))
                self.logger.info("[SrTeRestoreService][discover] tunnels is %s" % tunnels)
                if tunnels['items']:
                    # Process the response returned to the AOC.
                    self.handler_response(tunnels, device_id, result)

            # Update context information to generate a mark for the next round of service restoration. If the
number of records exceeds the upper limit or the data processing is complete, the service restoration
process ends.
            if int(self.cache_context["offset"]) < int(self.cache_context["ifm_count"]):
                result.goOn = True
            for key in self.cache_context:
                self.logger.debug("interface %s" % self.cache_context[key])
                result.cacheContext[key] = self.cache_context[key]

        self.logger.info("[SrTeRestoreService][discover] result %s" % result)
        time_end = time.time()
        self.logger.info("[SrTeRestoreService][discover] cost time is %s seconds" % (time_end - time_begin))
        return result
```

```
    # The code automatically determines whether to perform full or incremental service restoration. No
other code is required.
    def handler_template_content(self, ifm_dicts, aoc_context, device_id):
    """
    Process the SRTE tunnel template content and return the template content set.
    :param ifm_dicts: ifm_dicts.
    :param cache_context: discover context.
    :param aoc_context: context.
    :param device_id: device ID.
    :return: ifm_list
    """
    ifm_list = []
    # Query the network instance.
    ipv4s_and_ipv6s = self.read_ipv4_and_ipv6_from_data_store(device_id, aoc_context)
    ipv4s = ipv4s_and_ipv6s['ipv4s']
    ipv6s = ipv4s_and_ipv6s['ipv6s']
    # Query existing tunnels on the AOC based on the device ID.
    ifm_name_dict = self.valid_ifm_exist(aoc_context, device_id)
    for ifm in ifm_dicts['interface']:
        self.logger.info("[SrTeRestoreService][handler_template_content]:1")
        # The value segment-routing of signal-protocol indicates the SRTE tunnel.
        if self.is_segment_routing(ifm):
            self.logger.info("[SrTeRestoreService][handler_template_content]:is_segment_routing")
            # If data already exists on the AOC page, you do not need to restore it.
            if ifm_name_dict and ifm['name'] in ifm_name_dict:
                continue
            self.logger.info("[SrTeRestoreService][handler_template_content]:in ifm_name_dict")
            # Determine whether the interface is restored based on cacheContext.
            if self.cache_context is not None and self.cache_context[ifm['name']] is not None and \
                    self.cache_context[ifm.name] == 'exist':
                self.cache_context[ifm['name']] = 'exist'
                continue
            self.logger.info("[SrTeRestoreService][handler_template_content]:not exist")
            # Parse the RDB result.
            if ifm['name'] in ipv4s:
                ifm['ipv4'] = ipv4s[ifm['name']]
            if ifm['name'] in ipv6s:
                ifm['ipv6'] = ipv6s[ifm['name']]

            # Update the ifm information in the context, indicating that the configuration has been restored.
            self.cache_context[ifm['name']] = 'exist'
            self.logger.info("[SrTeRestoreService][handler_template_content] ifm:%s" % ifm['name'])
            sr_te_restore_template_content = {
                "interface": ifm
            }
            self.init_size = self.init_size + 1
             # Check whether the maximum number of configurations to be processed at a time is reached.
            if self.init_size > self.handler_max_size:
                break
            ifm_list.append(sr_te_restore_template_content)
    return ifm_list
```

7.2.4 Developing a Jinja2 Template

To develop a Jinja2 template, you can export the required YANG template on the service management page and fill variables in the YANG template as parameters. For details about the variables, see the Jinja2 template for forward delivery.

Step 1 Choose **Service Management** from the main menu. The **Service Management** page is displayed.

Step 2 Click  , select **NeTunnel**, enable **Merge template**, select **Structure only**, and click **OK**.

Step 3 Open the exported template. You can see that the file contains the data structure. Add the logic for processing and add the configuration data to be restored as

variables. Add the edited content to the **template>cmnetSrte.j2** file. The following is an example of some of the code:

```
{#Nest the items label to facilitate concurrent rendering of multiple service configurations.#}
<items>
{%- for node in  ifm_list %}
<item>
{#Ensure that the data structure of the label is the same as that in the exported service configuration
template.#}
<neTunnel xmlns="urn:icmcc:params:xml:ns:yang:cmcc-ip-srte">
   <deviceId>{{ ne_id }}</deviceId>
   {%- set interface = node.interface %}
{#Add a judgment statement to perform processing on data.#}
{#Variables come from NE configuration data. You can refer to the data structure settings in the Jinja2
template for configuration delivery or the template exported from the NE configuration page.#}
{%- if interface['name'] %}
   <tunnelName>{{ interface['name']}}</tunnelName>
   {%- if interface['tunnel-protocol'] %}
   {%- if interface['tunnel-protocol']['te-tunnel'] %}
   {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr'] %}
   <igpAttr>
      {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr']['hold-time'] %}
      <advertiseHoldTime>{{ interface['tunnel-protocol']['te-tunnel']['igp-attr'][ 'hold-time'] }}</
advertiseHoldTime>
      {%- endif %}
      {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr']['relative'] %}
      <igpMetricType>relative</igpMetricType>
      <igpMetricValue>{{ interface['tunnel-protocol']['te-tunnel']['igp-attr']['relative'] }}</igpMetricValue>
      {%- endif %}
      {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr']['absolute'] %}
      <igpMetricType>absolute</igpMetricType>
      <igpMetricValue>{{ interface['tunnel-protocol']['te-tunnel']['igp-attr']['absolute'] }}</igpMetricValue>
      {%- endif %}
      {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr']['shortcut-type'] %}
      <shortcutType>{{ interface['tunnel-protocol']['te-tunnel']['igp-attr']['shortcut-type'] }}</shortcutType>
      {%- endif %}
      {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr']['adv-enable'] %}
      {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr']['adv-enable'] == 'false' %}
      <enableAdvertise>disable</enableAdvertise>
      {%- else %}
      <enableAdvertise>enable</enableAdvertise>
      {%- endif %}
      {%- endif %}
   </igpAttr>
   {%- endif %}
   {%- endif %}
   {%- endif %}
...
```

----End

7.3 Verifying the SSP Package

7.3.1 Verifying the YANG Files

After an SSP package is developed, you can use the YANG model verification tool to verify the validity of the YANG files in the SSP package.

Step 1 Decompress **yang-offline-util.zip**.

Step 2 Copy the YANG model files in the **yang** directory of the SSP package to the directory where **yang-offline-util.zip** is decompressed.

Step 3 Run the following command to check whether the YANG files are correct:

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar validate console path .

If the command output is empty, the YANG file format is correct.

```
C:\Users\demo>cd yang-offline-util
C:\Users\demo\yang-offline-util>Java -jar yang-offline-util.jar validate console path .
C:\Users\demo\yang-offline-util>
```

----End

7.3.2 Performing a Unit Test

Step 1 Use the **yang-offline-util.zip** tool to generate empty NETCONF packets based on the device YANG models (**ifm** and **network_instance**).

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar generateSubtree .

If the command output is empty, the **subtree.xml** file is generated successfully.

Step 2 Open the **subtree.xml** file, and then set labels to proper values under **<application>**.

```
<interface>
    <mtu>500</mtu>
    <name>Tunnel1</name>
    <description>1234567890</description>
    <tunnel-protocol xmlns="urn:huawei:yang:huawei-tunnel-management">
      <te-tunnel xmlns="urn:huawei:yang:huawei-mpls-te">
        <sr-te>
          <hotstandby>
            <wtr-time>10</wtr-time>
            <path-overlap>false</path-overlap>
            <revertive-mode>revertive</revertive-mode>
          </hotstandby>
          <bfd-for-lsp xmlns="urn:huawei:yang:huawei-mpls-te-bfd">
            <detect-multiplier>3</detect-multiplier>
            <min-rx-interval>3</min-rx-interval>
            <ability>enable</ability>
            <min-tx-interval>3</min-tx-interval>
          </bfd-for-lsp>
          <bfd-for-tunnel xmlns="urn:huawei:yang:huawei-mpls-te-bfd">
            <detect-multiplier>3</detect-multiplier>
            <min-rx-interval>3</min-rx-interval>
            <mode>seamless</mode>
            <ability>enable</ability>
            <min-tx-interval>3</min-tx-interval>
          </bfd-for-tunnel>
          <bandwidth>1</bandwidth>
          <hold-priority>0</hold-priority>
          <pce-delegate>active</pce-delegate>
          <lsp-paths>
            <lsp-path>
              <explicit-path-name>expath_0110</explicit-path-name>
              <type>primary</type>
              <include-any>0x0</include-any>
              <exclude-any>0x0</exclude-any>
              <hop-limit>20</hop-limit>
            </lsp-path>
            <lsp-path>
              <explicit-path-name>expath_0111</explicit-path-name>
              <type>hot-standby</type>
              <include-any>0x0</include-any>
              <exclude-any>0x0</exclude-any>
              <hop-limit>32</hop-limit>
            </lsp-path>
          </lsp-paths>
```



```
      <setup-priority>0</setup-priority>
    </sr-te>
    <igp-attr xmlns="urn:huawei:yang:huawei-igp">
      <hold-time>0</hold-time>
      <shortcut-type>ospf</shortcut-type>
      <absolute>1</absolute>
      <adv-enable>false</adv-enable>
    </igp-attr>
    <common-attributes>
      <lsp-tp-outbound>true</lsp-tp-outbound>
      <signal-protocol>segment-routing</signal-protocol>
      <reserved-for-binding>true</reserved-for-binding>
      <egress-lsr-id>10.10.110.114</egress-lsr-id>
      <tunnel-id>1</tunnel-id>
      <statistic-enable>true</statistic-enable>
    </common-attributes>
  </te-tunnel>
 </tunnel-protocol>
</interface>
<interface>
 <mtu>500</mtu>
 <name>Tunnel2</name>
 <description>1234567890</description>
 <tunnel-protocol xmlns="urn:huawei:yang:huawei-tunnel-management">
  <te-tunnel xmlns="urn:huawei:yang:huawei-mpls-te">
   <sr-te>
    <hotstandby>
     <wtr-time>10</wtr-time>
     <path-overlap>false</path-overlap>
     <revertive-mode>revertive</revertive-mode>
    </hotstandby>
    <bfd-for-lsp xmlns="urn:huawei:yang:huawei-mpls-te-bfd">
     <detect-multiplier>3</detect-multiplier>
     <min-rx-interval>3</min-rx-interval>
     <ability>enable</ability>
     <min-tx-interval>3</min-tx-interval>
    </bfd-for-lsp>
    <bfd-for-tunnel xmlns="urn:huawei:yang:huawei-mpls-te-bfd">
     <detect-multiplier>3</detect-multiplier>
     <min-rx-interval>3</min-rx-interval>
     <mode>seamless</mode>
     <ability>enable</ability>
     <min-tx-interval>3</min-tx-interval>
    </bfd-for-tunnel>
    <bandwidth>1</bandwidth>
    <hold-priority>0</hold-priority>
    <pce-delegate>active</pce-delegate>
    <lsp-paths>
     <lsp-path>
      <explicit-path-name>expath_0110</explicit-path-name>
      <type>primary</type>
      <include-any>0x0</include-any>
      <exclude-any>0x0</exclude-any>
      <hop-limit>20</hop-limit>
     </lsp-path>
     <lsp-path>
      <explicit-path-name>expath_0111</explicit-path-name>
      <type>hot-standby</type>
      <include-any>0x0</include-any>
      <exclude-any>0x0</exclude-any>
      <hop-limit>32</hop-limit>
     </lsp-path>
    </lsp-paths>
    <setup-priority>0</setup-priority>
   </sr-te>
   <igp-attr xmlns="urn:huawei:yang:huawei-igp">
    <hold-time>0</hold-time>
    <shortcut-type>ospf</shortcut-type>
    <absolute>1</absolute>
```

```
        <adv-enable>false</adv-enable>
      </igp-attr>
      <common-attributes>
        <lsp-tp-outbound>true</lsp-tp-outbound>
        <signal-protocol>segment-routing</signal-protocol>
        <reserved-for-binding>true</reserved-for-binding>
        <egress-lsr-id>10.10.110.114</egress-lsr-id>
        <tunnel-id>1</tunnel-id>
        <statistic-enable>true</statistic-enable>
      </common-attributes>
    </te-tunnel>
  </tunnel-protocol>
</interface>'''
```

Step 3 Double-click the .py file in the **test** directory, copy the code in the <application> label to the <hvpnService> label, and supplement the code for simulation. The following is an example.

```
import unittest
import sys
from aoc.base.aoccontext import AocContext
from aoc.sys import devicemgr, datastore
from mock import Mock
sys.path.insert(0, "../../python")
from com.huawei.cmnet.srte.srte_restore import SrteRestoreService
from aoc.sys.sys_model_pb2.querydevicesoutput_pb2 import QueryDevicesPagedResult
from aoc.ncs.ncs_model_pb2.discover_pb2 import DiscoverInput


deviceId = "1111222222333344"
class Test(unittest.TestCase):

  # The local test cannot invoke the real environment data. Use the simulation method to invoke the simulation data.
    def mock_sys_func(self):
        devicemgr.query_basic_data_from_db = self.mock_query_basic_data_from_db
        datastore.read_datastore_rdb = self.mock_read_datastore_rdb

    # Simulate the queried device information.
    def mock_query_basic_data_from_db(self,neid='', nename='', nemanageip=''):
        response_body = QueryDevicesPagedResult()
        add_entity = response_body.queryDevicesPagedEntity.add()
        add_entity.neId = "neId"
        add_entity.operateName = "operateName"
        add_entity.hardWare = "NE40E"
        add_entity.softWare = "V800R21C00"
        add_entity.manageIp = "10.10.10.10"
        add_entity.type = "NE40E"
        add_entity.manufacturer = "HUAWEI"
        return response_body


    def setUp(self):
        self.mock_sys_func()


    # Simulate the invocation of the service restoration interface to trigger service restoration.
    def test_restore_srte(self):
        discoverinput = DiscoverInput()
        deviceInfo = discoverinput.deviceInfo.add()
        deviceInfo.deviceId = deviceId
        aoc_context = AocContext()
        aoc_context.transactionId = '112123'
        result = SrteRestoreService(None, '../../').discover(discoverinput, aoc_context)
        print(result)
        for sercfg in result.serviceConfig:
            print(sercfg.serviceData)

  # Simulate the packet returned by the device. The packet is long. The following is an example of only part of the packet.
```



```
    def mock_read_datastore_rdb(self, aoccontext, path, query_para=None):
        ifm_path = '/huawei-ac-nes:inventory-cfg/nes/ne/' + deviceId + \
                   '/huawei-ifm:ifm/interfaces/interface'
        network_instance_path = '/huawei-ac-nes:inventory-cfg/nes/ne/' + deviceId + \
                                '/huawei-network-instance:network-instance/' \
                                'instances/instance/_public_'
        valid_path = '/huawei-ac-applications:applications/cmcc-ip-srte:neTunnel/'

      # #Replace the code in labels such as <interface> based on the subtree.xml file and the actual
situation.
        ifm_result_context = '''
    <interface>
      <mtu>500</mtu>
      <name>Tunnel1</name>
      <description>1234567890</description>
      <tunnel-protocol xmlns="urn:huawei:yang:huawei-tunnel-management">
        <te-tunnel xmlns="urn:huawei:yang:huawei-mpls-te">
          <sr-te>
            <hotstandby>
              <wtr-time>10</wtr-time>
              <path-overlap>false</path-overlap>
              <revertive-mode>revertive</revertive-mode>
            </hotstandby>
            <bfd-for-lsp xmlns="urn:huawei:yang:huawei-mpls-te-bfd">
              <detect-multiplier>3</detect-multiplier>
              <min-rx-interval>3</min-rx-interval>
              <ability>enable</ability>
              <min-tx-interval>3</min-tx-interval>
            </bfd-for-lsp>
            <bfd-for-tunnel xmlns="urn:huawei:yang:huawei-mpls-te-bfd">
              <detect-multiplier>3</detect-multiplier>
              <min-rx-interval>3</min-rx-interval>
              <mode>seamless</mode>
              <ability>enable</ability>
              <min-tx-interval>3</min-tx-interval>
            </bfd-for-tunnel>
            <bandwidth>1</bandwidth>
            <hold-priority>0</hold-priority>
            <pce-delegate>active</pce-delegate>

    ...


        if path == ifm_path:
            if query_para:
                return ifm_result_with_para
            return ifm_result
        if path == network_instance_path:
            return network_instance_result
        if path == valid_path:
            return valid_result


if __name__ == "__main__":
    unittest.main()
```

Step 4 Run the test code to view the generated packet and check whether the output is correct.

If the message "Process finished with exit code 0" is displayed, the unit test is successful. Otherwise, check whether the attributes in the Jinja2 template correspond to those in the service YANG model.

----End

7.4 Compiling an SSP Package

Perform the following operations to compile a software package:

Step 1 In the PyCharm window, click **Terminal** at the bottom of the window to open the CLI.

Step 2 Run the following command to move the exported private key file to the **key** directory in the software package path:

(dem) C:\Users\demo\PycharmProjects\dem>copy path\to\privkey.asc .\key\privkey.asc

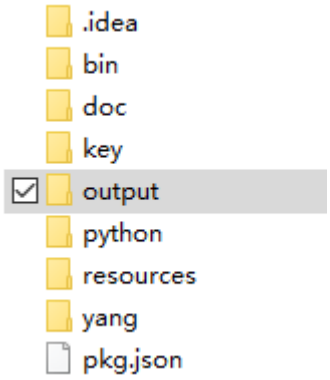
Step 3 Run the **makeFile.bat** file in the **bin** directory in the software package path to develop the software package.

Run the following commands in the CLI:

(dem) C:\Users\demo\PycharmProjects\dem>cd bin

(dem) C:\Users\demo\PycharmProjects\dem\bin>makeFile.bat

Step 4 If the following information is displayed in the command output, the software package is developed successfully. In this case, you can go to the **output** directory in the software package path to obtain the software package and its signature file.




```
2019-11-07 14:18:00,812 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Execute success
2019-11-07 14:18:00,819 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Clean dir success
2019-11-07 14:18:01,127 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] - [Sign]
Key length:3072
2019-11-07 14:18:01,180 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] -
[Sign]Generate signature file success.
2019-11-07 14:18:01,181 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Sign: Execute success
```

----End

Troubleshooting

Question: What can I do if an error message "NO JAVA_HOME" is displayed when I compile a software package?

Answer: You need to install the JDK software. You can download the JDK software from the **official website** and install it. JDK1.8 is recommended.

8 Developing an RPC SSP Package

An SSP package provides the following functions: EasyMap, RPC, and Discover. With RPC, you can define functions. If the standard NETCONF or YANG configuration model cannot meet requirements, you can flexibly define functions, such as query operations.

This chapter uses a simple example to describe how to develop an RPC SSP package for querying device information based on the downloaded default SSP package to obtain device packets.

Sample code packages for open programming in typical scenarios have been uploaded to the AOC. You can log in to the community to query and download sample code packages as needed. Development based on sample code packages improves efficiency and reduces difficulty. If no suitable sample code package is found, you can download the default SSP software package from the open programming environment for development.

When developing an SSP package, protect key information, such as user passwords, login tokens, sessions, and other personal data. Do not output such information in logs.

8.1 Creating an SSP Package Template

Step 1 Choose **Package Repo** from the main menu. Then, choose **Package Management** from the navigation pane, and click **Create Template** on the displayed page.

Step 2 In the displayed **Add** dialog box, set the attributes of the SSP package.

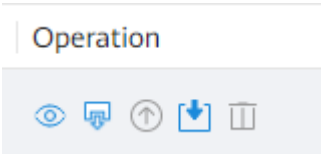
Set the attributes of the SSP package according to **Table 8-1**.

Table 8-1 Attributes of the RPC SSP package

Attribute	Value
Name	RPC
Version	1.0.0
Provider	HUAWEI
Package type	Specific Service plugin
Service name	RPC
Service point	RPC
Mapping type	python+template

Step 3 Click **OK**. In the dialog box that is displayed indicating that the software package is added successfully, click **OK**. The new software package is displayed in the software package list.

Step 4 Click in the **Operation** column of the software package record to download the software package to the local PC.



----End

8.2 Developing an SSP Package

8.2.1 Editing the SSP Package Configuration File

Step 1 Open PyCharm (the community edition is used as an example) and click **Create New Project**. The **New Project** dialog box is displayed. Expand the **Project Interpreter** area, confirm the configuration, and click **Create**. If a project has been created, you can directly open it in PyCharm.

Step 2 Decompress the downloaded software package to the directory where the project is located.

Step 3 In the navigation pane of the IDE, double-click the **pkg.json** file.

Step 4 Modify the parameters according to the following table.

Table 8-2 Parameter values in the **pkg.json** file

Parameter	Description	Value	Mandatory/ Optional
name	Software package name.	The value is user-defined and is a string of 1 to 64 characters consisting of digits, uppercase letters, lowercase letters, underscores (_), and hyphens (-).	Mandatory
version	Software package version number.	The version number must be in the xxxx.xxxx.xxxx format. The value of each *x* ranges from 0 to 9, and each segment supports a maximum of four *x*s.	Mandatory
description	Software package description.	The value is user-defined and has a maximum of 128 characters.	Optional
package-type	Package type.	Three enumerated values are available: **Specific Service plugin**, **Generic NE driver**, and **Specific NE driver**.	Mandatory
group	Package group information, which is used for sandbox process isolation.	The value is user-defined and has a maximum of 128 characters. If this parameter is not set, the default value **global** is used.	Optional
producer	Provider of the software package.	For SND and GND packages, only the packages with this parameter being **huawei** are supported currently.	Mandatory
augment-isolation	Model range ID.	The value is a Boolean value and can only be **true** or **false**. In the DCN single-NE scenario, set this parameter to **true**. In other scenarios, set this parameter to **false**.	Optional

Parameter	Description	Value	Mandatory/ Optional
nce-min-versions	Minimum version of the OPS to which the package is compatible with.	The value is in the format *x.x.x*. If the most significant version number is changed, an incompatible change occurs. The value **1.0.0** indicates that compatibility is maintained with all versions in the range from 1.0.0 (included) to 2.0.0 (excluded). You can enter multiple version segments, with one segment in each row. If this parameter is left empty, compatibility is maintained with all versions.	Optional
package-dependencies	Dependency on third-party packages.	If this parameter is set, the following parameters must be set: Third-party package name: Name of the third-party package on which the software package depends. The value can contain a maximum of 64 characters. Version: version number in x.x.x format.	Optional
snd-type	Type of the SND package.	The value **system** indicates a built-in SND package. Built-in SND package of the system. You are advised not to change the value of this parameter. If this parameter is not set, the default value **specific** is used.	Optional
snd-id	SND package ID. This parameter needs to be set only for SND packages. The value must be unique for each SND package.	The value is user-defined and has a maximum of 128 characters. This parameter is mandatory for SND packages and is optional for other types of packages.	Optional

Parameter	Description	Value	Mandatory/ Optional
devices	Information about the device matching the driver, which is specific to SND and GND packages. This parameter describes the software package and is not used for device management.	If you set this parameter, the following parameters must also be set: **vendor**: device vendor. The value can contain a maximum of 128 characters. **device-type**: device type. The value can contain a maximum of 128 characters. **device-version**: device version. The value can contain a maximum of 128 characters.	Optional
service-name	Service name. This parameter is available only to SSP packages.	The value is user-defined and has a maximum of 64 characters. This parameter is mandatory for SSP packages. The value must be unique for each SSP package. For other types of software packages, you do not need to set this parameter.	Optional

Parameter	Description	Value	Mandatory/ Optional
hooks	Callback mapping information. The hooks combination must be unique in a single package, but can be the same in different packages. Multiple hooks can be set when multiple functions are developed together.	If you set this parameter, set the following parameters: **type** (mandatory): callback type. For EasyMap, set **type** to **mapping**. For RPC, set **type** to **service-rpc**. For Discover, set **type** to **discover**. **key** (mandatory): callback key value, which can contain a maximum of 128 characters. **java-class-name** (optional): callback mapping class. This parameter is specific to Java. The value can contain a maximum of 256 characters. **python-class-name** (optional): callback mapping class. This parameter is specific to Python. The value contains a maximum of 256 characters. **groovy-class-name** (optional): callback mapping class. This parameter is specific to Groovy. The value contains a maximum of 256 characters. **template** (optional): template name. The value contains a maximum of 256 characters.	Optional

An example is as follows:

```
{
   "name": "RPC",
   "version": "1.0.15",
   "description": "This is a RPC service",
   "package-type": "ssp",
   "service-name":"RPC",
   "group": "xxxx",
   "producer": "Huawei",
   "nce-min-versions":[

 ],

   "hooks": [
    {
      "type": "service-rpc",
      "key": "(http://example.com/RPC?revision=2018-12-09)read_device_vrf",
      "python-class-name": "RPCSSP.RpcService.AocNcsAaaService"
    }
```



```
  ]
}
```

----End

8.2.2 Compiling a Service YANG Model

In the navigation pane of the IDE, double-click the .yang file in the **yang** directory to compile a YANG model. By default, the YANG file name corresponds to the value of **service-name** in the **pkg.json** file, and the module name in the YANG model corresponds to the value of **service-name** in the **pkg.json** file. The value of **app:application-definition** in the YANG model corresponds to the value of **key** under **hooks** in the **pkg.json** file, that is, the service point.

Currently, only service points of the list and container types can be mounted to the root node of the application. If a service point is defined on a parent node, no service point can be defined on a child node.

If the standard NETCONF or YANG configuration model cannot meet requirements, you can customize RPC functions to define the function input and output. RPC is used to define one-off operations that do not require data saving, such as the ping command.

This section describes how to define the function of querying the device interface status by defining the input **device-id** and **interface** and the output **interface** and **ifV4State**.

```
module RpcService {
   namespace "http://example.com/RPC";
   prefix "RPC";

   import huawei-ac-applications {
      prefix app;
   }
   description
      "The module for RpcService example.";

   revision 2018-12-09 {
      description "Initial revision.";
   }

   augment "/app:applications"{
      list RPC {
         app:application-definition "aaa";
         key "instance_name";
         leaf instance_name {
            type string;
         }
      }
   }
   //Define the input device-id and interface.
   rpc read_device_vrf {
   input {
         leaf device-id {
            mandatory true;
            type string;
         }
         container interfaces {
            list interface {
               key name;
               leaf name {
                  type string {
                    length "1..max";
                  }
                }
```



```
            }
          }
        }
        // Define the output interface and ifV4State.
        output {
            container interfaces {
                list interface {
                    key name;
                    leaf name {
                        type string {
                          length "1..max";
                        }
                    }
                    leaf ifV4State {
                        type string;
                    }
                }
            }
        }
    }
}
```

Data Encryption

You can edit the **security.xml** file in the **yang** directory to determine whether to encrypt the data corresponding to the nodes in the YANG model.

If data encryption is enabled, data is encrypted when being saved to the database and is decrypted when being read from the database; when the configuration is delivered, the encrypted data fields are displayed as ******; when the northbound RESTCONF v1 API queries the encrypted data field, the value in ciphertext is returned; during synchronization, encrypted data fields are not synchronized; during data consistency verification, encrypted data fields are not delivered.

Note: When data is saved to the database, the key field of the list cannot be encrypted. In addition, only data of the string type can be encrypted.

Table 8-3 Parameters in the **security.xml** file

Node	Parent Node	Attribute	Description
security-node	/	N/A	Root node.
path	security-node	-	Security node path. Example: /a:b/c/d
-	-	type	Node type: leaf leaf-list typedef: customized type By default, the node is of the leaf type.

If the node type is leaf, only the <path> label needs to be defined. Generally, you do not need to set the attribute. The data corresponding to each <path> is

encrypted. The value of **path** must start with the top-level node **<moduleName>:<nodeName>**, where *<moduleName>* is the module name of the YANG file and *<nodeName>* is the name of the top-level node in the module. The top-level node of the path can be followed by child nodes, which are separated by a slash (/). An example is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<security-node>
   <path>/db-security-test:security-top/security-sec/security-sec-leaf-list</path>
   <path type="typedef">/db-security-test:security-len-typedef-leaf</path>
   <path>/db-security-test:security-top/security-list/security-list-leaf</path>
   <path>/dbtest:nodes/dbtest:node/db-security-test:augment-cont/augment-cont-security-leaf</path>
   <path>/dbtest:nodes/dbtest:node/db-security-test:augment-security-leaf</path>
</security-node>
```

High-Risk Operations

You can edit the **risk_operation.xml** file in the **yang** directory to define whether the RPC operation in the YANG model is a high-risk operation. When you perform a high-risk operation on the GUI, a confirmation dialog box is displayed. No confirmation dialog box will be displayed for non-high-risk operations.

All RPC operations are defined in the <rpcs> label. Each RPC high-risk operation corresponds to one <rpc> label. The value of the <rpc> label is in the format moduleName:rpcName. An example is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<risk-operation>
   <rpcs>
      <rpc>huawei-aaa:reactivateUser</rpc>
      <rpc>huawei-aaa:changeMyPassword</rpc>
      <rpc>huawei-aaa:changeMyIdleTimeout</rpc>
   </rpcs>
 </risk-operation>
```

Blacklist

You can edit the **blacklist.xml** file in the **yang** directory to define whether modules in the YANG model are shielded on the GUI.

The modules to be shielded are defined in <module>, in which the module name and module revision need to be set. An example is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<blacklist>
  <module>
     <name>huawei-cfg</name>
     <revision>2020-03-18</revision>
  </module>
  <module>
     <name>huawei-dhcp</name>
     <revision>2020-04-29</revision>
  </module>
</blacklist>
```

YANG Model Verification Exemption

You can edit the **multiRevisionModule.xml** file in the **yang** directory to define whether modules in the YANG model need to be verified during the upgrade.

Verification-free modules are defined in <modules>. Each module corresponds to one <module> label, which is set to the module name. An example is as follows:


```
<?xml version="1.0" encoding="utf-8"?>
<modules>
   <module>ietf-yang-types</module>
   <module>ietf-inet-types</module>
 </modules>
```

User-Defined Model Permission

By writing a permission control file, you can customize operation permissions on nodes in YANG models. After a software package is successfully installed, the permission information is dynamically injected to the OPS. The user management function allows users with different responsibilities to be granted with appropriate permissions, preventing unauthorized and insecure operations. Permissions on software package models are automatically assigned to the default role whereas permissions assigned to user-defined roles need to be configured manually.

You can add the **permission.json** file to the **resources** directory and customize permissions on nodes in YANG models. If you have not written the **permission.json** file, the OPS automatically generates a level-0 permission control file for each module based on the YANG file during software package loading. The permission control file includes the create, delete, read, update, and execute permissions.

The OPS supports permission-based operations on container and list nodes in YANG models. You can define permissions on these nodes as needed. Customization of permissions at levels 0, 1, and 2 is supported. Level 0 indicates the permission control at the module level. If a user does not match permissions, the OPS executes permission control at level 0 by default. At level 1, a container or list node has only one level and does not have subnodes. At level 2, a container or list node has two levels and subnodes. During permission control, the OPS matches permissions based on the longest match rule. That is, permissions at a deeper level are matched first.

```
{
   "modules": [
   {
      "module-name":"hbng",
      "operations": [
      {
         "uri-pattern":"containerA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":"listA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":" listA/listC",
         "method":" create/delete/read/update"
      },
      {
         "uri-pattern":"resetStatictic",
         "method":"exec"
      }
      ]
   }
   ]
}
```


8.2.3 Developing the SSP Package Code

In the navigation pane of the IDE, double-click the .py file in the Python directory and write Python code to implement the RPC function logic.

Batch template operations and batch device configuration operations have a large impact scope. You are advised to carefully evaluate risks during code development. You can throw exceptions in UserException mode and view the exception information on the GUI.

In addition, if a service injection package provides general-purpose SDK capabilities or provides internal service interfaces, you can directly call the interfaces provided by the service injection package to quickly implement functions, improving development efficiency.

When you customize the RPC function, derived classes need to inherit the NcsService base class and override the ncs_rpc method to implement the customized function logic based on service requirements. In this way, the processing result can be returned.

```
# Mandatory. Import the NcsService class. NcsService is the parent class provided by aoc_api for the SSP package to inherit. The devicemgr SDK API can be used to query data from devices in real time.
from aoc import NcsService, devicemgr

class AocNcsAaaService(NcsService):
    # Query the SRTE status.
    INTERFACE_PATH = "/huawei-ifm:ifm"

    # Set the device query path.
    PROPERTIES_SUBTREE = """
            <interfaces>
              <interface>
                <ifDynamicInfo>
                  <ifV4State/>
                </ifDynamicInfo>
              </interface>
            </interfaces>
          """

    RESULT_TEMPALTE = """
          <read_device_vrf xmlns="http://example.com/RPC">
             <interfaces>
                %s
             </interfaces>
          </read_device_vrf>
          """

    RESULT_VRF_ONE_TEMPALTE = """
                <interface>
                   <name>%s</name>
                   <ifV4State>%s</ifV4State>
                </interface>
          """

    # RPC logic
    def ncs_rpc(self, request, context=None, template=None):

        # Obtain the input.
        request_xml = self.xmltodictnode(request.rpcXML.strip())

        # Obtain the device ID.
        device_id = request_xml.get("read_device_vrf").get("device-id")
        srte_list = request_xml.get("read_device_vrf").get("interfaces").get("interface")
        if isinstance(srte_list, dict):
            srte_list = [srte_list]
```



```
    # Query device information based on the device ID.
    output = self.query_device_by_device_id(device_id)

    # Query the return value.
    result = self.result_Construction(output, srte_list)
    return result


# Construct the response body.
def result_Construction(self, output, srte_list):

    # Obtain the interface status on the device.
    all_ifm_status_up = self.filter_ifm_status(output)

    # Construct the response body and return the configured interface status.
    srte_all_status = ""
    for stre in srte_list:
        name = stre.get("name")
        status = 0
        if name in all_ifm_status_up:
            status = all_ifm_status_up[name]
        srte_all_status = srte_all_status + (self.RESULT_VRF_ONE_TEMPALTE % (name, status))
    self.logger.info("[srte-status] srtes status is %s", srte_all_status)
    return (self.RESULT_TEMPALTE % srte_all_status).strip()


# Set the interface status.
def filter_ifm_status(self, output):
    all_ifm_status_up = {}

    # Obtain the interface status.
    if output.result is not None:
        ifm_dict = self.xmltodictnode(output.result.strip())
        if ifm_dict.get("ifm") and ifm_dict.get("ifm").get("interfaces"):
            interface_dict = ifm_dict.get("ifm").get("interfaces").get("interface")
            for interface in interface_dict:
                ifm_name = interface.get("ifName")
                ifV4State = interface.get("ifDynamicInfo").get("ifV4State")
                all_ifm_status_up[ifm_name] = ifV4State
    return all_ifm_status_up


# Query device information based on the device ID.
def query_device_by_device_id(self, device_id):

    # Query device information and obtain packets from the device.
    properties = {
        "subtree": self.PROPERTIES_SUBTREE.strip()
    }
    output = devicemgr.query_data_from_device(device_id, 'OPERATIONAL',
                                              self.INTERFACE_PATH, properties, 60000)
    return output
```

NOTE

To view the result returned by a user-defined RPC operation, you are advised to invoke the northbound API.

8.2.4 Developing a Jinja2 Template

In Python code, northbound packets are not transferred to the Jinja2 template for conversion. Therefore, you do not need to develop a Jinja2 template.

8.3 Verifying the SSP Package

8.3.1 Verifying the YANG Files

After an SSP package is developed, you can use the YANG model verification tool to verify the validity of the YANG files in the SSP package.

Step 1 Decompress **yang-offline-util.zip**.

Step 2 Copy the YANG model files in the **yang** directory of the SSP package to the directory where **yang-offline-util.zip** is decompressed.

Step 3 Run the following command to check whether the YANG files are correct:

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar validate console path .

If the command output is empty, the YANG file format is correct.

```
C:\Users\demo>cd yang-offline-util
C:\Users\demo\yang-offline-util>Java -jar yang-offline-util.jar validate console path .
C:\Users\demo\yang-offline-util>
```

----End

8.3.2 Performing a Unit Test

Step 1 Use the **yang-offline-util.zip** tool to generate empty NETCONF packets based on the YANG model. Run the following command.

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar generateSubtree .

If the command output is empty, the **subtree.xml** file is generated successfully.

Step 2 Open the **subtree.xml** file, and then set labels to proper values under **<application>**.

```
<RPC xmlns="http://example.com/RPC">
   <instance_name>NE1</instance_name>
</RPC>
```

Step 3 Double-click the **.py** file in the **test** directory, and copy the code in the <application> label to the <hvpnService> label. The following is an example.

```
import unittest
import sys
sys.path.insert(0, "../../Python")
from HVPNService.HVPNService import AocNcs_servicepoint

class Test(unittest.TestCase):
   xml = '''

  # Replace the following code in the hvpnService label based on actual situations.
   <RPC xmlns="http://example.com/RPC">
      <instance_name>NE1</instance_name>
   </RPC>
   '''


   def test_case1(self):
      result = AocNcs_servicepoint().ncs_map_test(self.xml)
      print(result)

if __name__ == "__main__":
   unittest.main()
```

Step 4 Run the test code to view the generated packet and check whether the output is correct.

If the message "Process finished with exit code 0" is displayed, the unit test is successful. Otherwise, check whether the attributes in the Jinja2 template correspond to those in the service YANG model.

----End

8.4 Compiling an SSP Package

Perform the following operations to compile a software package:

Step 1 In the PyCharm window, click **Terminal** at the bottom of the window to open the CLI.

Step 2 Run the following command to move the exported private key file to the **key** directory in the software package path:

(dem) C:\Users\demo\PycharmProjects\dem>copy path\to\privkey.asc .\key \privkey.asc

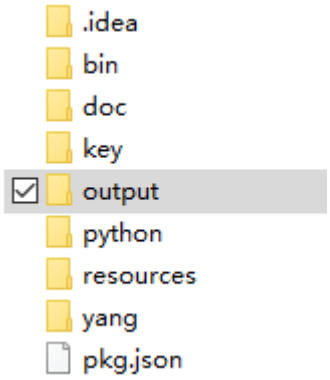
Step 3 Run the **makeFile.bat** file in the **bin** directory in the software package path to develop the software package.

Run the following commands in the CLI:

(dem) C:\Users\demo\PycharmProjects\dem>cd bin

(dem) C:\Users\demo\PycharmProjects\dem\bin>makeFile.bat

Step 4 If the following information is displayed in the command output, the software package is developed successfully. In this case, you can go to the **output** directory in the software package path to obtain the software package and its signature file.




```
2019-11-07 14:18:00,812 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Execute success
2019-11-07 14:18:00,819 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Clean dir success
2019-11-07 14:18:01,127 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] - [Sign]
Key length:3072
2019-11-07 14:18:01,180 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] -
[Sign]Generate signature file success.
2019-11-07 14:18:01,181 INFO
```



```
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Sign: Execute success
```

----End

Troubleshooting

Question: What can I do if an error message "NO JAVA_HOME" is displayed when I compile a software package?

Answer: You need to install the JDK software. You can download the JDK software from the **official website** and install it. JDK1.8 is recommended.

9 Importing and Installing Software Packages

After all software packages are developed, import them to the OPS and install them for online commissioning or use.

9.1 System Login

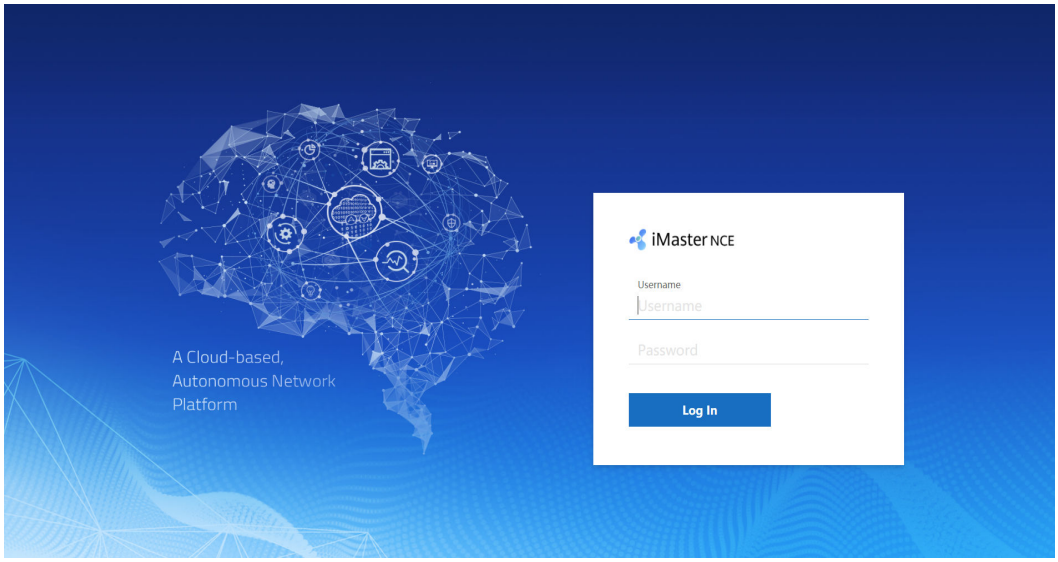
This section describes how to log in to the system using a browser. The table below lists the browsers that are supported.

Browser	Version
Microsoft Edge (Chromium Inside)	MicrosoftEdge_89.0.774.27_Beta
Chrome Stable Channel	Chrome 73.0.3683 and later
Firefox	• Firefox 65 and later • Firefox 68 ESR and later

Logging In to the Agile Open Container App

Step 1 Log in to the NCE O&M plane. Access the O&M plane at **https://***IP address of the O&M plane*:**31943**. Enter the user name and password and click **Log In**.

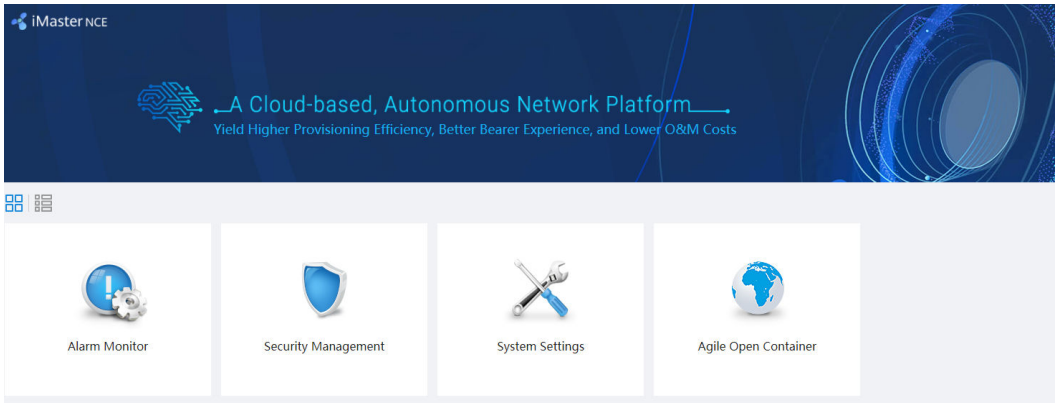
Figure 9-1 Logging in to the NCE O&M plane

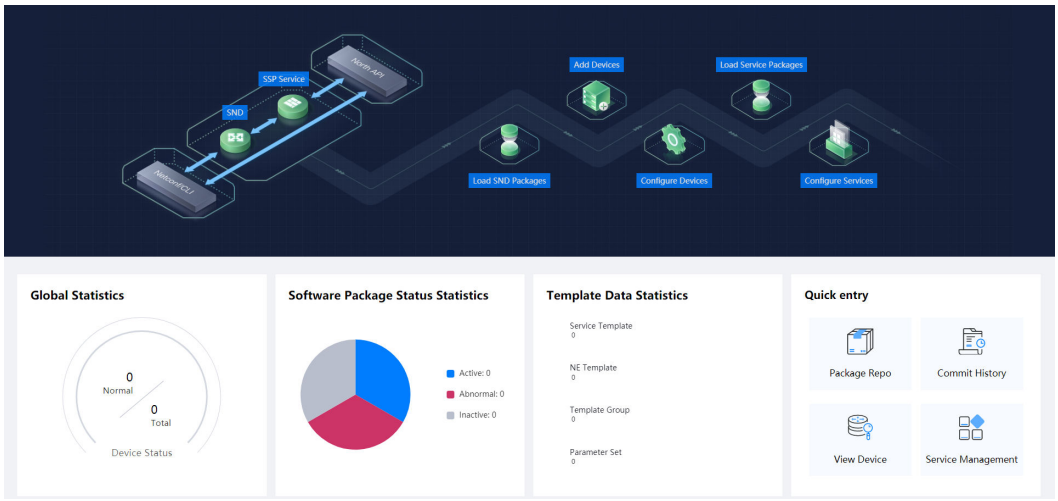


NOTE

- You need to change the password upon the first login. Keep the new password properly. To improve system security, you are advised to periodically change the password to prevent security risks such as brute force cracking.
- The IP address of the O&M plane is the client login IP address configured on the Common_Service node. If the Common_Service node is deployed in a cluster, the IP address is set to the floating IP address of the cluster. If the Common_Service node is deployed in single-node mode, the IP address is the client login IP address of the node.

Step 2 Log in to the system and click the **Agile Open Container** app. The Agile Open Container home page is displayed.





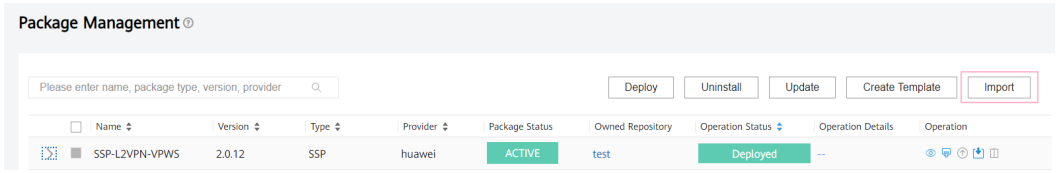
Step 3 On the home page, click the corresponding shortcut entry or click any shortcut entry based on the actual application scenario to access the main menu.

----End

9.2 Importing and Activating a Software Package

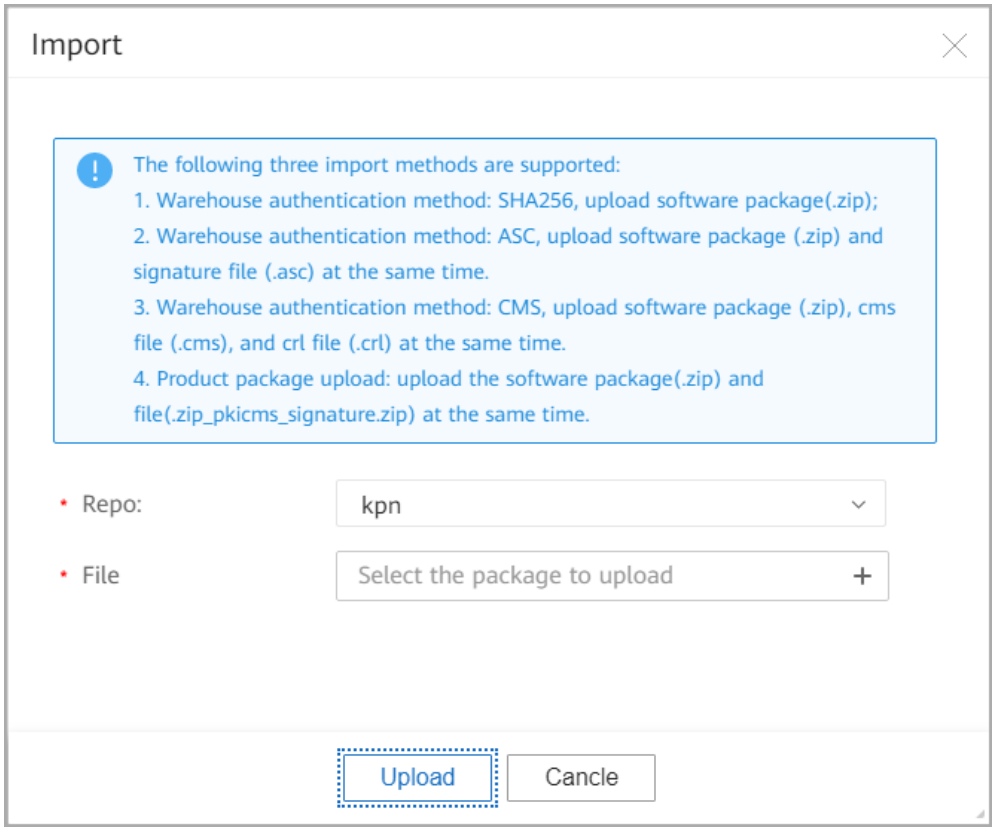
Step 1 Choose **Package Repo** from the main menu. Then, choose **Package Management** from the navigation pane, and click **Import** on the displayed page.

Figure 9-2 Importing a software package



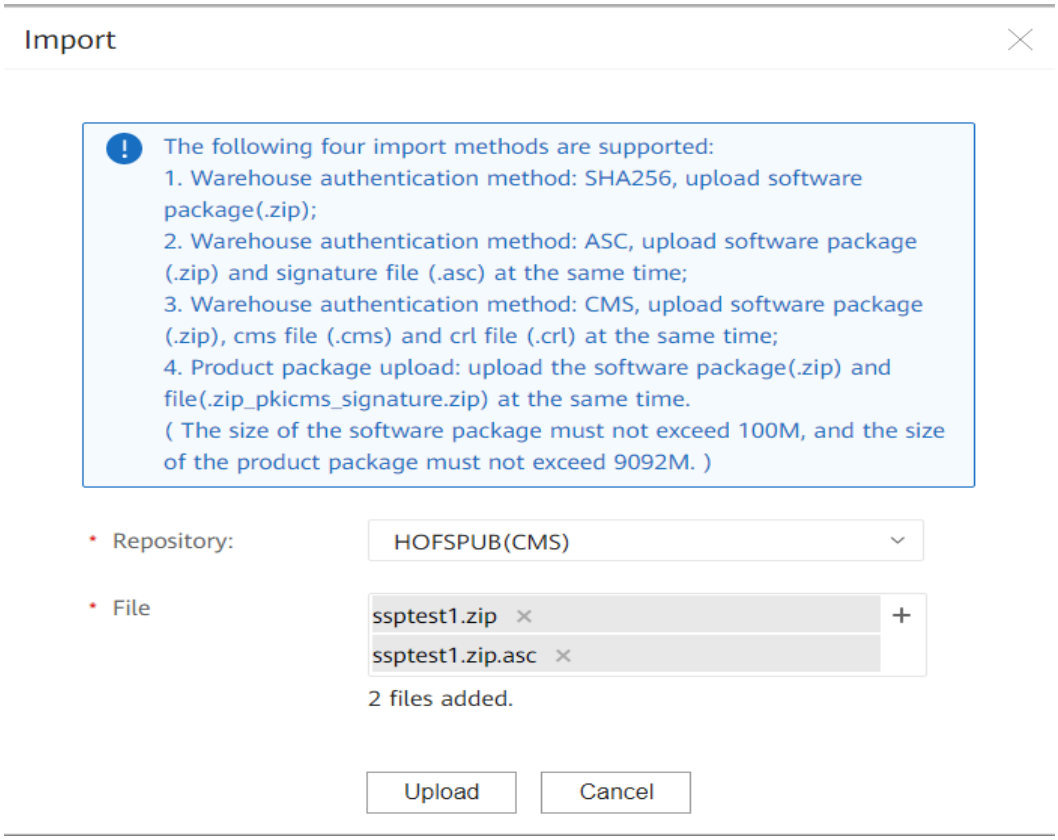
Step 2 On the displayed page, select the software package and signature file to be imported.

Figure 9-3 Selecting the software package to be imported



Step 3 Click **Upload**.

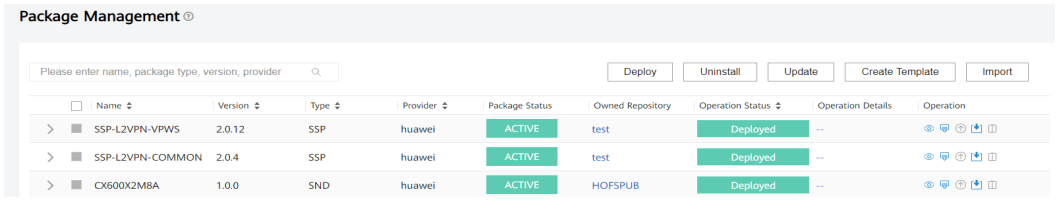
Figure 9-4 Clicking **Upload**



After the software package is imported, you can view the imported package on the **Package Management** page.

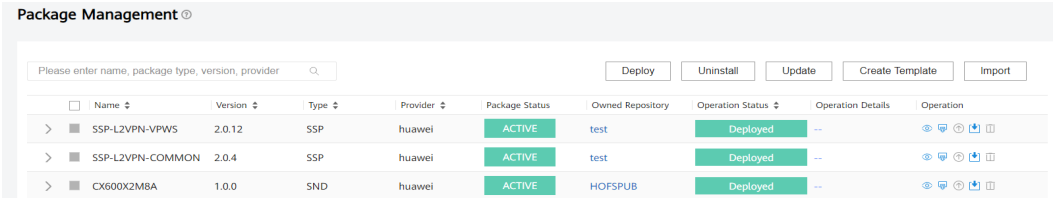
Step 4 Choose **Package Repo** from the main menu. Then, choose **Package Management** from the navigation pane, and click on the displayed page.

Figure 9-5 Installing a software package



Step 5 Wait for a few minutes until the software package is installed. If the software package status changes to Activated, the software package is successfully installed. If other information is displayed, rectify the fault based on the failure information in the Details column and reinstall the software.

Figure 9-6 Software package import success



----End

10 Software Package Upgrade

Compatible upgrade means that compatibility is maintained with the changes made to YANG models in the software package during the upgrade. Incompatible upgrade means that compatibility is not maintained with the changes made to YANG models in the software package during the upgrade. In this mode, special processing needs to be performed during the upgrade.

Incompatible Upgrade

The procedure of incompatible upgrade is described as follows:

Step 1 Create a migration package template and edit the package configuration file.

1. Create a migration package based on the default template. On the **Package Management** page, click **Create Template**. Set the package type to **Snd Migrate Plugin** and the provider and device type to be the same as those of the SND package to be upgraded. Click **Export** to download the created template package to the local PC. The following table lists the attributes of the migration package.

   Table 10-1 Migration package attributes

Attribute	Value
Name	NE8000M8_Migrate
Version	1.0.0
Provider	HUAWEI
Package type	Snd Migrate Plugin
Mapping type	python
Device type	NetEngine 8000 M8

2. Edit the **pkg.json** file. Set **from-snd-id** to the value of **snd-id** of the SND package before the upgrade and **to-snd-id** to the value of **snd-id** of the SND package after the upgrade.

```
"from-snd-id":"migrate_test_new_domain",
"to-snd-id":"migrate_test_new_domain2",
```


Step 2 Compare the models before and after the upgrade to check for incompatibilities.

1. Use the incompatible-nodes-check tool to identify the incompatibilities. On the CLI, run the following command:

   incompatible-nodes-check-*.*.*-jar-with-dependencies.jar [*Old YANG directory*] [*New YANG directory*] [*Result output directory*]

2. The tool will stop at the point where the node name or hierarchy of a new YANG model changes compared with the old YANG model. You can modify the new YANG model. For example, change the node names to be the same, or add the missing container node to the new YANG model. (Note that the path of **augment** of other nodes also needs to be modified.)
3. After eliminating the found incompatibility, continue to run the tool until all incompatibilities are identified.

Step 3 Check the comparison result and preliminarily identify the real incompatibilities.

Depending on service scenarios, the tool may detect compatibilities as incompatibilities. For example, if int32 is converted into int8, the tool identifies an incompatibility. However, if the original service data complies with int8, data can still be migrated properly.

Step 4 Identify all incompatibilities and perform corresponding operations according to **Table 10-2**.

- In simple incompatibility scenarios, configure corresponding rules in the **migration_rules.xml** file by referring to **Table 10-3**. (Note: You do not need to perform any configuration in the **migration_rules.xml** file if migration rules are not required.) Obtain the **migration_rules.xml** file from the **resources** folder in the migration package. After the configuration is complete, place the file back to the migration package. The following is an example of migration rules:

```
<?xml version="1.0" encoding="utf-8"?>
<migration-rules>
   <migration-rule>
      <rule>RETAIN_VALUE</rule>
      <migration-item>
         <source-path>/huawei-l3vpn:l3vpn/l3vpnName</source-path>
      </migration-item>
   </migration-rule>
   <migration-rule>
      <rule>CHANGE_VALUE</rule>
      <migration-item>
         <source-path>/huawei-l3vpn:l3vpn/instances/instance/qosDomAppNode/dscp-values</source-
path>
      </migration-item>
   </migration-rule>
   <migration-rule>
      <rule>REMOVE_LEVEL</rule>
      <migration-item>
         <source-path>/huawei-l3vpn:l3vpn/instances</source-path>
      </migration-item>
   </migration-rule>
   <migration-rule>
      <rule>NODE_MAP</rule>
      <migration-item>
         <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/
instanceChildlist</source-path>
         <des-path>/huawei-l3vpn:l3vpn/instance/instanceChildContainer/instanceChildList</des-path>
      </migration-item>
   </migration-rule>
   <migration-rule>
      <rule>SECURITY</rule>
```



```
      <migration-item>
         <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/
unSecurityNode</source-path>
      </migration-item>
   </migration-rule>
   <migration-rule>
      <rule>KEY_CHANGE</rule>
      <migration-item>
         <source-path>/huawei-l3vpn:l3vpn/instances/instance</source-path>
      </migration-item>
   </migration-rule>
</migration-rules>
```

- In complex incompatibility scenarios, configure the **COMPLEX** rule in the migration rule file **migration_rules.xml**. Data of the corresponding nodes and subnodes is not migrated to the child table. In addition, the service callback interface needs to process the complex incompatibility nodes.

  The following is an example of **COMPLEX** rules:

```
<migration-rule>
    <rule>COMPLEX</rule>
    <migration-item>
       <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/
unSecurityNodeList1</source-path>
    </migration-item>
</migration-rule>
```

Table 10-2 Rule configuration

Rule	Incompatible Item	Example	Description

KEY_CHANGE	New key indexes	Old YANG model
		list {
		key "key1 key2"
		leaf key1
		leaf key2
		leaf key3
		leaf key4
		}
		New YANG model
		list {
		key "key1 key2 key3"
		leaf key1
		leaf key2
		leaf key3
		}
			If the key value after migration can be found based on the original key value, the scenario is considered as a simple incompatibility scenario. Otherwise, the scenario is considered as a complex incompatibility scenario.

Rule	Incompatible Item	Example	Description

	Key index deletion	Same as the old YANG model in the preceding example
		New YANG model
		list {
		key "key1"
		leaf key1
		leaf key2
		}
			If the key value after migration can be found based on the original key value, the scenario is considered as a simple incompatibility scenario. Otherwise, the scenario is considered as a complex incompatibility scenario.

	Key index change: A key index is changed to that of an existing node.	Same as the old YANG model in the preceding example
		New YANG model
		list {
		key "key4 key2 key3"
		leaf key1
		leaf key2
		...
		}
			Simple or complex incompatibility scenario

NODE_MAP	Leaf node name change	-	Simple incompatibility scenario
NODE_MAP	New container path	-	Simple incompatibility scenario
CHANGE_VALUE	Leaf node value change	-	Simple incompatibility scenario
SECURITY	Leaf node encryption	-	Simple incompatibility scenario

Rule	Incompatible Item	Example	Description
RETAIN_VALUE	Scenario where the value remains unchanged (The unit and value range may change, but the value remains unchanged.)	-	Simple incompatibility scenario
REMOVE_LEVEL	Container path deletion	-	Simple incompatibility scenario
ORDERED_BY_CHANGE	Change from **user** to **system** for **ordered-by** in the list	-	Simple incompatibility scenario

COMPLEX	Key index change: A key index is changed to that of a non-existing node.	Old YANG model
		list {
		key "key1 key2"
		leaf key1
		leaf key2
		leaf key3
		leaf key4
		}
		New YANG model
		list {
		key "key1' key2 key3"
		leaf key1'
		leaf key2
		}
			Complex incompatibility scenario

	Node deletion	-	Complex incompatibility scenario
	Module name change		Complex incompatibility scenario
	Deletion of **augment**		Complex incompatibility scenario

Rule	Incompatible Item	Example	Description

	Deletion of **choice**	choice {
		list
		}
		->
		list
			If the table structure changes, the scenario is considered as a complex incompatibility scenario.

	Case deletion	choice {
		case {
		leaf1
		leaf2
		}}
		->
		choice {
		leaf1
		leaf2
		}
			If multiple nodes in a case are split into multiple cases, ensure that only one case node in the new data has a value. The scenario is considered as a complex incompatibility scenario.

	Addition of **choice**	list
		leaf
		->
		choice {
		list
		}
		leaf
			If a new table is added under **choice** that is added, the scenario is considered as a complex incompatibility scenario.

Exemption rule	Name change of **choice**	Old choice: aaa; New choice: bbb	Simple incompatibility scenario
	Case name change	Old case: bbb; New case: ddd	Simple incompatibility scenario

Rule	Incompatible Item	Example	Description

	Deletion of **choice**	The following is an example that does not involve table structure changes:
		choice {
		leaf
		}
		-> leaf
			If **choice** contains only the leaf node and **container** does not involve the table creation node, the deletion of **choice** is compatible. In this case, this scenario is considered as the simple incompatibility scenario with exemption rules.

	Addition of **choice**	leaf
		->
		choice {
		leaf
		}
			If no new table is added to under **choice**, this scenario is considered as the simple incompatibility scenario with exemption rules.

	New case	choice {
		leafa
		leafb
		}
		->
		choice {
		case {
		leafa
		leafb
		}
		}
			Simple incompatibility scenario

	namespace	-	Simple incompatibility scenario
	Addition or deletion of the **presence** attribute	-	Simple incompatibility scenario

Table 10-3 Example of the migration rule file

Rule	Example of the Migration Rule File Note: rule indicates the rule name; source-path indicates the path of the old YANG node; des-path indicates the path of the new YANG node.
KEY_CHANGE	<migration-rule> <rule>KEY_CHANGE</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance</source-path> </migration-item> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/huawei-if-ip:ipv4-ifs/ipv4</source-path> </migration-item> </migration-rule>
NODE_MAP	<migration-rule> <rule>NODE_MAP</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/instanceChildlist</source-path> <des-path>/huawei-l3vpn:l3vpn/instance/instanceChildContainer/instanceChildList</des-path> </migration-item> </migration-rule>
CHANGE_VALUE	<migration-rule> <rule>CHANGE_VALUE</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/qosDomAppNode/dscp-values</source-path> </migration-item> </migration-rule>
SECURITY	<migration-rule> <rule>SECURITY</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/unSecurityNode</source-path> </migration-item> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/unSecurityNodeList</source-path> </migration-item> </migration-rule>
RETAIN_VALUE	<migration-rule> <rule>RETAIN_VALUE</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/l3vpnName</source-path> </migration-item> </migration-rule>
REMOVE_LEVEL	<migration-rule> <rule>REMOVE_LEVEL</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances</source-path> </migration-item> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/instanceChildContainer2</source-path> </migration-item> </migration-rule>

Rule	Example of the Migration Rule File Note: rule indicates the rule name; source-path indicates the path of the old YANG node; des-path indicates the path of the new YANG node.
COMPLEX	<migration-rule> <rule>COMPLEX</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/unSecurityNodeList1</source-path> </migration-item> </migration-rule>

Step 5 In complex incompatibility scenarios, compile callback scripts for services. Currently, Python and Java interfaces are supported, and only the JSON database is supported.

The following uses the Python interface as an example. Open the .py file in the **python** directory of the migration package and compile the callback code.

```
from aoc.snd.snd_model_pb2.migrateInfo_pb2 import CustomMigrateOutput
from aoc.snd.snd_model_pb2.migrateInfo_pb2 import ValueChangeOutput
from aoc.sys.datastore_migrate import *

class CustomMigrate():
    """
    template
    """


    def __init__(self, logger):
        self.logger = logger


    def custom_migrate(self, aoccontext, request=None):
        """
        template
        """
        self.logger.info('custom_migrate start.')
        node_path = '/huawei-l3vpn:l3vpn'
        query_source_param = {'ne_id': '8d394835-cb84-38f3-a4d5-36a7f2074b50'}

        # Read the corresponding data source.
        sources = read_datastore_sources(aoccontext, node_path, query_source_param)
        query_old_data_param = {'ne_id': '8d394835-cb84-38f3-a4d5-36a7f2074b50', 'old_domain_id':
'migrate_old'}
        write_new_data_param = {'ne_id': '8d394835-cb84-38f3-a4d5-36a7f2074b50', 'new_domain_id':
'migrate_new'}

        # Process data sources one by one.
        for source in sources:

            # Read the old data in the old SND data source.
            nodeDataWithSourceResult = read_migrate_node_data(aoccontext, path, query_old_data_param,
source)
            """
            modify old data: nodeDataWithSourceResult: Customize the logic for converting old data into new
data.
            """
            self.logger.info('read_migrate_node_data, nodeDataWithSourceResult type = %s' %
type(nodeDataWithSourceResult))

            # Write new data to the new SND data source.
            result = write_migrate_node_data(aoccontext, write_new_data_param, source,
nodeDataWithSourceResult)
```



```
        self.logger.info('result. %s' % result)
    return bytearray(str(result), encoding='utf8')


# Automatically invoke the new interface to change the value in the CHANGE_VALUE rule.
def value_change(self, aoccontext, request=None):
    """
    template
    """
    self.logger.info("value_change start.%s, %s" % (type(aoccontext), type(request)))
    out = ValueChangeOutput()
    if request.path == '/l3vpn:l3vpn/interface/ip/binding/vpn-instance':
        out.newData = '1234'
    else:
        out.newData = request.oldData
    self.logger.info("value_change end.")
    return out
```

Step 6 Compile the migration package, and import and activate the migration package. During device upgrade, operations are automatically performed based on the rules and callback scripts configured in the migration package.

----End

Compatible/Incompatible Upgrade Specifications

The following table lists the compatible and incompatible upgrade specifications.

Table 10-4 Compatible/Incompatible upgrade specifications

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

schema node (container, list, leaf, leaf-list, anyxml, anydata, choice, case, rpc, action, input, output, notification)	An optional schema node is added.	An optional schema node that does not exist in the original YANG interface is added to the new YANG interface.	Compatible	Before modification:
				container topology-config {
				leaf id {
				type string;
				}
				}
				After modification:
				container topology-config {
				leaf id {
				type string;
				}
				leaf name { type string; } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	A mandatory schema node is added.	A mandatory schema node (for example, key or mandatory) that does not exist in the original YANG interface is added to the new YANG interface.	Incompatible	Before modification:
				container topology-config {
				leaf id {
				type string;
				}
				}
				After modification:
				container topology-config {
				leaf id {
				type string;
				}
				leaf name { mandatory true; type string; } }

	A schema node is deleted.	A schema node in the original YANG interface cannot be found in the new YANG interface.	Incompatible	Before modification: container topology-config { leaf id { type string; } leaf name { type string; } } After modification: container topology-config { leaf id { type string; } }

typedef,feature	Addition	Add a typedef/ feature definition.	Compatible	Before modification: N/A
				After modification:
				typedef my-type {
				status deprecated;
				type int32;
				}

	Deletion	Delete a typedef/ feature definition.	Incompatible	Before modification: typedef my-type { status deprecated; type int32; } After modification: N/A

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

Data type	The value type changes between the integer types (int8, int16, int32, int64, uint8, uint16, uint32, and uint64) and the value range is extended.	For example, the value type is changed from int8 to int32, which extends the value range. This scenario is considered as a compatible change.	Compatible	Before modification:
				leaf id {
				type uint32;
				}
				After modification:
				leaf id {
				type uint64; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	The value type is changed (except the preceding scenarios) or the value type does not change but the range is narrowed.	1. The value type changes between the integer types but the value range after the change is narrowed (the value range after the change cannot cover that before the change, for example, from int8 to uint16).
		2. The value type changes between non-integer types or between integer and non-integer types. (For example, the value type is changed from string to leafref or from uint to string.)
		3. The data type remains unchanged, but the value range is narrowed (the range after the change cannot cover the range before the change, for example, from 0..20 of
			Incompatible	Example 1:
				Before modification:
				container topology-config {
				leaf id {
				type string;
				}
				}
				After modification:
				container topology-config {
				leaf id {
				type uint32; } }
				Example 2:
				Before modification:
				typedef interface-ref { type unit8; }
				After modification:
				typedef interface-ref { type leafref { path "/if:interfaces/if:interface/if:name"; } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
		the string type to 1..30 of the string type.)		
Quantity range	The quantity range is extended.	The quantity range of the same element is changed. For example, the value of **min-elements** decreases or the value of **max-elements** increases.	Compatible	Before modification: leaf-list id { type string; min-elements 2 max-elements 5 } After modification: leaf-list id { type string; min-elements 1 max-elements 6 }}
	The quantity range is narrowed.	The quantity range of the same element is narrowed. For example, the value of **min-elements** increases or the value of **max-elements** decreases.	Incompatible	Before modification: leaf-list id { type string; min-elements 1 max-elements 6 } After modification: leaf-list id { type string; min-elements 2 max-elements 5 }}
Value range	The value range is extended.	The value range of the same attribute is extended.	Compatible	Before modification: type uint8 { range "0..60"; } After modification: type uint8 { range "0..63"; }
	The value range is narrowed.	The value range of the same attribute is narrowed.	Incompatible	Before modification: type uint8 { range "0..64"; } After modification: type uint8 { range "0..63"; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
Length	The length range is extended.	The length of the same attribute is extended.	Compatible	Before modification: type string { length "0..64"; } After modification: type string { range "0..70"; }
	The length range is narrowed.	The length of the same attribute is narrowed.	Incompatible	Before modification: type string { length "0..64"; } After modification: type string { range "0..50"; }
	The length range label is deleted.	The length range label of the same attribute is deleted.	Compatible	Before modification: type string { length "0..64"; } After modification: type string { }

	The length range label is added.	The length range label of the same attribute is added.	Incompatible	Before modification:
				type string {
				}
				After modification:
				type string {
				range "0..50"; }

	The fixed length changes.	An attribute uses a fixed length, which changes. The change is incompatible regardless of whether the fixed length increases or decreases.	Incompatible	A fixed length is considered as a length range with the upper and lower limits being the same. The fixed length is changed to a variable length (for example, due to service design solution changes), which is considered as an incompatible change.

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
	The fixed length label is deleted.	The fixed length label of the same attribute is deleted.	Incompatible	
	The fixed length label is added.	The fixed length label of the same attribute is added.	Incompatible	

Enumeration	An enumeration item is added.	An enumeration item is added for the same attribute.	Compatible	Before modification:
				type enumeration {
				enum aaa;
				}
				After modification:
				type enumeration {
				enum aaa;
				enum bbb; }

	An enumeration item is deleted.	An enumeration item is deleted for the same attribute.	Incompatible	Before modification: type enumeration { enum aaa; enum bbb; } After modification: type enumeration { enum aaa; }

	An enumeration item is modified.	The enumeration items of the same attribute are changed.	Incompatible	Before modification:
				type enumeration {
				enum aaa;
				}
				After modification:
				type enumeration {
				enum bbb; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	An enumerated value is added.	An enumerated value is added for the same attribute.	Incompatible	Compatible change:
				Before modification:
				type enumeration {
				enum one;
				}
				After modification:
				type enumeration {
				enum one {
				value 0; } }
				Incompatible change:
				Before modification:
				type enumeration { enum one; }
				After modification:
				type enumeration { enum one { value 1; } }

	An enumerated value is deleted.	An enumerated value is deleted for the same attribute.	Incompatible	Compatible change:
				Before modification:
				type enumeration {
				enum aaa {
				value 0;
				}
				enum bbb { value 1; } }
				After modification:
				type enumeration { enum aaa { value 0; } enum bbb; }
				Incompatible change:
				Before modification:
				type enumeration { enum aaa { value 0; } enum bbb { value 3; } }
				After modification:
				type enumeration { enum aaa { value 0; } enum bbb; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	An enumerated value is changed.	The enumerated values of the same attribute are changed.	Incompatible	Before modification:
				type enumeration {
				enum aaa {
				value 0; } }
				After modification:
				type enumeration { enum aaa { value 1; } }

Default value	The default value is changed.	The default value of the same attribute is changed.	Incompatible	Before modification: leaf spfInitInterval { type "uint32"; default 50; } After modification: leaf spfInitInterval { type "uint32"; default 40; }
	The default value label is deleted.	The default value label of the same attribute is deleted.	Incompatible	Before modification: leaf spfInitInterval { type "uint32"; default 50; } After modification: leaf spfInitInterval { type "uint32"; }

	The default value label is added.	The default value label of the same attribute is added.	Incompatible	Before modification:
				leaf spfInitInterval {
				type "uint32";
				}
				After modification:
				leaf spfInitInterval {
				type "uint32";
				default 40; }

Regular expression	The regular expression changes.	The regular expression of the same attribute is changed.	Incompatible	Before modification: type string { pattern "[0-9\.]*"; } After modification: type string { pattern "[0-9]*"; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
	The regular expression label is deleted, or one or more regular expressions are deleted.	The regular expression label of the same attribute is deleted, meaning that the range is expanded.	Compatible	Before modification: type string { pattern "[0-9\.]*"; } After modification: type string { }

	The regular expression label is added, or one or more regular expressions are added.	The regular expression label of the same attribute is added, meaning that the range is narrowed.	Incompatible	Before modification:
				type string {
				}
				After modification:
				type string {
				pattern "[0-9]*"; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	The modifier statement of the regular expression pattern is added or deleted.	The modifier statement of the same regular expression pattern is added or deleted.	Incompatible	Before modification:
				type string {
				length "1..max";
				pattern '[a-zA-Z_][a-zA-Z0-9\-_.]*';
				pattern '[xX][mM][lL].*' {
				modifier invert-match;
				}
				}
				After modification:
				type string {
				length "1..max";
				pattern '[a-zA-Z_][a-zA-Z0-9\-_.]*';
				pattern '[xX][mM][lL].*';
				}

Constraints on **key**	Key constraints are deleted.	Key constraints of the same list are deleted.	Incompatible	Before modification: list link { key "id name"; uses link-config; } After modification: list link { key "id"; uses link-config; }
	Key constraints are added.	Key constraints of the same list are added.	Incompatible	Before modification: list link { key "id"; uses link-config; } After modification: list link { key "id name"; uses link-config; }
	Key constraints are modified.	Key constraints of the same list are modified.	Incompatible	Before modification: list link { key "id"; uses link-config; } After modification: list link { key "name"; uses link-config; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
	The key sequence is changed.	The key sequence of the same list is changed.	Incompatible	Before modification: list link { key "protocol ip"; leaf protocol { type string; } leaf ip { type string; } } After modification: list link { key "ip protocol"; leaf ip { type string; } leaf protocol { type string; } }
Constraints on **mandatory**	Mandatory constraints are deleted.	The mandatory field of the same attribute is changed from **true** to **false**.	Compatible	Before modification: leaf name { mandatory true; type string; } After modification: leaf name { mandatory false; type string; }
	Mandatory constraints are added.	The mandatory field of the same attribute is changed from **false** to **true**.	Incompatible	Before modification: leaf name { mandatory false; type string; } After modification: leaf name { mandatory true; type string; }

		The mandatory field is added and set to **true** for an attribute of the same object.	Incompatible	Before modification:
				leaf name {
				type string;
				}
				After modification:
				leaf name {
				mandatory true; type string; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

Constraints on **status**	Status constraints are used.	The status constraint of the same attribute is changed from **obsolete** to **current** or **deprecated**.
		The status constraint of the same attribute is changed from **deprecated** to **current**.
			Compatible	Before modification:
				leaf matchMode {
				type rtpMatchMode;
				mandatory true;
				status obsolete;
				description "Matching mode of nodes.";
				}
				After modification:
				leaf matchMode {
				type rtpMatchMode;
				mandatory true;
				status current;
				description "Matching mode of nodes.";
				}

	The status constraint is deprecated or obsolete.	The status constraint of the same attribute is changed from **current** to **obsolete** or **deprecated**.
		The status constraint of the same attribute is changed from **deprecated** to **obsolete**.
			Incompatible	Before modification:
				leaf matchMode {
				type rtpMatchMode;
				mandatory true;
				status current;
				description "Matching mode of nodes.";
				}
				After modification:
				leaf matchMode {
				type rtpMatchMode;
				mandatory true;
				status deprecated;
				description "Matching mode of nodes.";
				}

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
Unique constraints	Unique constraints are deleted.	Unique constraints of the same attribute are deleted.	Compatible	Before modification: list server { ... unique "ip port"; ... } After modification: list server { ... ... }

	Unique constraints are added.	Unique constraints of the same attribute are added.	Incompatible	Before modification:
				list server {
				...
				...
				}
				After modification:
				list server {
				...
				unique "ip port"; ... }

	Unique constraints are modified.	The number of unique attributes of the same object increases.	Compatible	Before modification:
				list server {
				...
				unique "ip";
				...
				}
				After modification:
				list server {
				...
				unique "ip port"; ... }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

		The number of unique attributes of the same object decreases.	Incompatible	Before modification:
				list server {
				...
				unique "ip port";
				...
				}
				After modification:
				list server {
				...
				unique "ip"; ... }

		The unique attribute of the same object is changed.	Incompatible	Before modification: list server { ... unique "ip port"; ... } After modification: list server { ... unique "ip port-b"; ... }
Constraints on **config**	Config constraints are added.	The config constraint of the same attribute (including the inheritance scenario) is changed from **true** to **false**.	Incompatible	Before modification: container network-topology-oper { config true; ... } After modification: container network-topology-oper { config false; ... }
	Config constraints are deleted.	The config constraint of the same attribute (including the inheritance scenario) is changed from **false** to **true**.	Compatible	Before modification: container network-topology-oper { config false; ... } After modification: container network-topology-oper { config true; ... }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

Bits constraints	The position value of bits is added.	The position value of bits with the same bit field attribute is added.	Incompatible	Incompatible change:
				Before modification:
				type bits {
				bit disable-nagle;
				}
				After modification:
				type bits {
				bit disable-nagle {
				position 1; } }
				Compatible change:
				Before modification:
				type bits { bit disable-nagle; }
				After modification:
				type bits { bit disable-nagle { position 0; } }

	The position value of bits is deleted.	The position value of bits with the same bit field attribute is deleted.	Incompatible	Incompatible change:
				Before modification:
				type bits {
				bit disable-nagle {
				position 1; } }
				After modification:
				type bits { bit disable-nagle; }
				Compatible change:
				Before modification:
				type bits { bit disable-nagle { position 0; } }
				After modification:
				type bits { bit disable-nagle; }

	The position value of bits is changed.	The position value of bits with the same bit field attribute is changed.	Incompatible	Before modification:
				type bits {
				bit disable-nagle {
				position 0; } }
				After modification:
				type bits { bit disable-nagle{ position 1; } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	New bit items are added.	Bit items of the same bit field attribute are added.	Compatible	Before modification:
				type bits {
				bit disable-nagle {
				position 0; } }
				After modification:
				type bits { bit disable-nagle { position 0; }; bit auto-sense-speed { position 1; } }

	Bit items are deleted.	Bit items of the same attribute are deleted.	Incompatible	Before modification:
				type bits {
				bit disable-nagle {
				position 0;
				}
				bit auto-sense-speed { position 1; } }
				After modification:
				type bits { bit disable-nagle { position 0; } }

	Bit items are changed.	Bit items of the same attribute are changed.	Incompatible	Before modification:
				type bits {
				bit disable-nagle {
				position 0;
				}
				bit auto-sense-speed {
				position 1;
				}
				}
				After modification:
				type bits {
				bit disable-nagle {
				position 0;
				}
				bit auto-sense-speed-xxx { position 1; } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
Proprietary extension syntax: support-filter	The value of support-filter is changed from **true** to **false**.	The value of support-filter is changed from **true** to **false** for a non-key leaf node. (The default value **false** is used if support-filter is not set.)	Incompatible	Before modification: leaf flag { type string; ext:support-filter true; } After modification: leaf flag { type string; }
	The value of support-filter is changed from **false** to **true**.	The value of support-filter is changed from **false** to **true** for a non-key leaf node. (The default value **false** is used if support-filter is not set.)	Compatible	Before modification: leaf flag { type string; } After modification: leaf flag { type string; ext:support-filter true; }
Proprietary extension syntax: value-meaning, item, and meaning	The mapping between item and meaning in value-meaning is changed.	In the type definition of a leaf node, the mapping between the item and meaning in the **value-meaning** attribute is changed.	Incompatible	Before modification: leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } ext:item 1 { ext:meaning ICMP; } } } } After modification: leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } ext:item 1 { ext:meaning IGMP; } } } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
	value-meaning is added or a certain item and meaning are added.	In the type definition of a leaf node, the **value-meaning** attribute is added, or an item and meaning are added.	Compatible	Before modification: leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } } } } After modification: leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } ext:item 1 { ext:meaning ICMP; } } } }
	value-meaning is deleted or a certain item and meaning are deleted.	In the type definition of a leaf node, the **value-meaning** attribute is deleted, or an item and meaning are deleted.	Incompatible	Before modification: leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } ext:item 1 { ext:meaning ICMP; } } } } After modification: leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } } } }
Proprietary extension syntax: case-sensitivity	**case-sensitivity** is deleted.	The value of the **case-sensitivity** attribute of a leaf node is deleted, or the value is changed from another value to **lower-and-upper**. (The default value is **lower-and-upper** if **case-sensitivity** is not set.)	Incompatible	Before modification: leaf def { type string; ext:case-sensitivity upper2lower; } After modification: leaf def { type string; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
	case-sensitivity is added.	The value of the **case-sensitivity** attribute of a leaf node is added, or the value is changed from **lower-and-upper** to another value. (The default value is **lower-and-upper** if **case-sensitivity** is not set.)	Incompatible	Before modification: leaf def { type string; } After modification: leaf def { type string; ext:case-sensitivity upper2lower; }
	The value of the **case-sensitivity** attribute is changed.	The value of the **case-sensitivity** attribute of a leaf node is changed.	Incompatible	Before modification: leaf def { type string; ext:case-sensitivity lower2upper; } After modification: leaf def { type string; ext:case-sensitivity upper2lower; }

Proprietary extension syntax: value-range	The value range specified by **value-range** is expanded.	The value range specified by **value-range** is expanded.	Compatible	Before modification:
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				ext:value-range "0..63"; } }
				After modification:
				leaf dscpValues { type pub-type: id-range { pattern "xxx"; ext:value-range "0..128"; } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	The value range specified by **value-range** is narrowed.	The value range specified by **value-range** is narrowed.	Incompatible	Before modification:
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				ext:value-range "0..63"; } }
				After modification:
				leaf dscpValues { type pub-type: id-range { pattern "xxx"; ext:value-range "10..63"; } }

	The **value-range** attribute is added.	The value range specified by **value-range** is added.	Incompatible	Before modification:
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				}
				}
				After modification:
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				ext:value-range "0..63"; } }

	The **value-range** attribute is deleted.	The **value-range** attribute is deleted.	Incompatible	Before modification:
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				ext:value-range "0..63"; } }
				After modification:
				leaf dscpValues { type pub-type: id-range { pattern "xxx"; } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

Private extended syntax: masklen, bit, and position	The value range of **masklen** is expanded.	The value range of **masklen** is expanded for a leaf node of the user-defined type bits8, bits16, bits32, or bits64.	Compatible	Before modification:
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 3; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }
				After modification:
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen 7; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }

	The value range of **masklen** is narrowed.	The value range of **masklen** is narrowed for a leaf node of the user-defined type bits8, bits16, bits32, or bits64.	Incompatible	Before modification:
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 7; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }
				After modification:
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen 3; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	The original mapping between the bit and position is changed.	The original mapping between the bit and position is changed for a leaf node of the user-defined type bits8, bits16, bits32, or bits64.	Incompatible	Before modification:
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 7;
				ext:bit Sun {
				position 0;
				}
				ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }
				After modification:
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen7; ext:bit Sun { position 0; } ext:bit Fri { position 1; } ext:bit Tue { position 2; } ... } }

	The mapping between a bit and a position is added.	The mapping between a bit and a position is added for a leaf node of the user-defined type bits8, bits16, bits32, or bits64.	Compatible	Before modification:
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 7;
				ext:bit Sun {
				position 0;
				}
				ext:bit Mon { position 1; } ... } }
				After modification:
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen7; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	The mapping between a bit and a position is deleted.	The mapping between a bit and a position is deleted for a leaf node of the user-defined type bits8, bits16, bits32, or bits64.	Incompatible	Before modification:
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 7;
				ext:bit Sun {
				position 0;
				}
				ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }
				After modification:
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen7; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ... } }

Proprietary extension syntax: task-name	**task-name** is added.	The **task-name** label of the same module is added.	Compatible	Before modification: module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; ... container aaa { ... } } After modification: module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; ... ext:task-name "aaa"; container aaa { ... } }
	task-name is deleted.	The **task-name** label of the same module is deleted.	Incompatible	Before modification: module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; ... ext:task-name "aaa"; container aaa { ... } } After modification: module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; ... container aaa { ... } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
	task-name is changed.	The value of the **task-name** attribute of the same module is changed.	Incompatible	Before modification: module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; ... ext:task-name "aaa"; container aaa { ... } } After modification: module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; ... ext:task-name "aaaom"; container aaa { ... } }
Proprietary extension syntax: node-ref	The **node-ref** parameter is added.	The **node-ref** parameter is added for the same RPC.	Compatible	Before modification: rpc mm { ext:node-ref "/aa/bbs/bb"; input { leaf name { type string; mandatory true; } } } After modification: rpc mm { ext:node-ref "/aa/bbs/bb"; ext:node-ref "/aa/ccs/cc"; input { leaf name { type string; mandatory true; } } }
	The **node-ref** parameter is deleted.	The **node-ref** parameter is deleted for the same RPC. An RPC may have multiple **node-ref** parameters, one or all of which are deleted.	Incompatible	Before modification: rpc mm { ext:node-ref "/aa/bbs/bb"; ext:node-ref "/aa/ccs/cc"; input { leaf name { type string; mandatory true; } } } After modification: rpc mm { ext:node-ref "/aa/bbs/bb"; input { leaf name { type string; mandatory true; } } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
	node-ref is changed.	The value of the same **node-ref** attribute changes.	Incompatible	Before modification: rpc mm { ext:node-ref "/aa/bbs/bb"; input { leaf name { type string; mandatory true; } } } After modification: rpc mm { ext:node-ref "/aa/ccs/cc"; input { leaf name { type string; mandatory true; } } }
Proprietary extension syntax: operation-exclude	The **operation-exclude** parameter is added.	The **operation-exclude** attribute is added for a node, or the value of the **operation-exclude** attribute is added.	Incompatible	Before modification: container aaa { leaf bbb { type uint32; } } After modification: container aaa { leaf bbb { ext:operation-exclude update; type uint32; } }
	operation-exclude is deleted.	The **operation-exclude** attribute is deleted for a node, or the value of the **operation-exclude** attribute is deleted.	Compatible	Before modification: container aaa { leaf bbb { ext:operation-exclude update|create; type uint32; } } After modification: container aaa { leaf bbb { ext:operation-exclude update; type uint32; } }
	The **operation-exclude** parameter is modified.	The **operation-exclude** parameter is modified, including changing the **when** condition or **filter** criteria.	Incompatible	Before modification: list aaa { ext:operation-exclude create|delete { ext:filter "name = '_public_'"; } ... } After modification: list aaa { ext:operation-exclude create|delete { ext:filter "name = '_default_'"; } ... }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
Proprietary extension syntax: dynamic-default	No default value is changed to **dynamic-default**.	A leaf node changes from no default value to **dynamic-default**.	Incompatible	Before modification: container aaa { leaf bbb { type uint32; } } After modification: container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when "../ccc = 'false'"; } } } }
	dynamic-default is changed to no default value.	A leaf node is changed from **dynamic-default** to no default value.	Incompatible	Before modification: container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when "../ccc = 'true'"; } } } } After modification: container aaa { leaf bbb { type uint32; } }
	A static default value is changed to **dynamic-default**.	A leaf node changes from the static default value to **dynamic-default**.	Incompatible	Before modification: container aaa { leaf bbb { type uint32; default 12; } } After modification: container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when "../ccc = 'false'"; } } } }
	dynamic-default is changed to a static value.	A leaf node is changed from **dynamic-default** to a static default value.	Incompatible	Before modification: container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when "../ccc = 'true'"; } } } } After modification: container aaa { leaf bbb { type uint32; default 12; } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
	The dynamic default value is changed.	The default value of the **dynamic-default** attribute or the **when** condition of **default-value** is changed.	Incompatible	Before modification: container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when "../ccc = 'true'"; } } } } After modification: container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when "../ccc = 'false'"; } } } }
Proprietary extension syntax: generated-by	**generated-by** is added, modified, or deleted.		Incompatible	
Proprietary extension syntax: refine-ext	The **operation-exclude** field is added to **refine-ext**.	**operation-exclude** is added to **refine-ext**.	Incompatible	
	The **operation-exclude** field is deleted from **refine-ext**.	**operation-exclude** is deleted from **refine-ext**.	Compatible	

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
	operation-exclude in **refine-ext** is changed.	**operation-exclude** in **refine-ext** is changed for a node.	Incompatible	
	generated-by in **refine-ext** is added, deleted, or modified.	**generated-by** in **refine-ext** of a node is changed.	Incompatible	
Description label	The description label is changed.	The description label of the same attribute is changed, including the description label in the proprietary syntax.	Compatible	Before modification: description "xxx"; After modification: description "xxx xxx";
leafref constraint	The **leafref** parameter is added.	The **leafref** relationship is added.	Incompatible	Before modification: typedef interface-ref { type unit8; } After modification: typedef interface-ref { type leafref { path "/if:interfaces/if:interface/if:name"; } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
	leafref is deleted.	The **leafref** relationship is deleted.	Compatible	Before modification: typedef interface-ref { type leafref { path "/if:interfaces/if:interface/if:name"; } } After modification: typedef interface-ref { type string; }

	The path of the same field changes.	The path associated with the same object in **leafref** is changed.	Incompatible	Before modification:
				typedef interface-ref {
				type leafref {
				path "/if:interfaces/if:interface/if:ip"; } }
				After modification:
				typedef interface-ref { type leafref { path "/if:interfaces/if:interface/if:name"; } }

	The value of **require-instance** is changed from **true** to **false**.	The value of **require-instance** in the **leafref** substatement is changed from the default value **true** to **false**.	Compatible	Before modification:
				leaf ip {
				type leafref {
				path "/if:interfaces/if:interface/if:ip";
				}
				}
				After modification:
				leaf ip {
				type leafref {
				path "/if:interfaces/if:interface/if:ip";
				require-instance false;
				}
				}

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	The value of **require-instance** is changed from **false** to **true**.	The value of **require-instance** in the **leafref** substatement is changed from **false** to **true**.	Incompatible	Before modification:
				leaf ip {
				type leafref {
				path "/if:interfaces/if:interface/if:ip";
				require-instance false;
				}
				}
				After modification:
				leaf ip {
				type leafref {
				path "/if:interfaces/if:interface/if:ip";
				}
				}

decimal64	**fraction-digits** is changed.	**fraction-digits** of the decimal64 type is changed, for example, from 2 to 3.	Incompatible	Before modification:
				typedef my-decimal {
				type decimal64 {
				fraction-digits 2;
				range "1 .. 3.14 | 10 | 20..max";
				}
				}
				After modification:
				typedef my-decimal {
				type decimal64 {
				fraction-digits 3;
				range "1 .. 3.14 | 10 | 20..max";
				}
				}

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

units label	The value of the units label is changed.	The value of the units label of the same attribute is changed.	Incompatible	Before modification:
				leaf interval {
				type uint16;
				default 30;
				units minutes;
				}
				After modification:
				leaf interval {
				type uint16;
				default 30;
				units seconds; }

	The units label is deleted.	The units label of the same attribute is deleted.	Compatible	Before modification: leaf interval { type uint16; default 30; units minutes; } After modification: leaf interval { type uint16; default 30; }

	The units label is added.	The units label of the same attribute is added.	Incompatible	Before modification:
				leaf interval {
				type uint16;
				default 30;
				}
				After modification:
				leaf interval {
				type uint16;
				default 30;
				units minutes; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
reference label	**reference** is changed.	The value of **reference** of the same attribute is changed.	Compatible	Before modification: typedef uri { type string; reference "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax"; ... } After modification: typedef uri { type string; reference "RFC 5040: Uniform Resource Identifier (URI): Generic Syntax"; ... }

	The reference label is deleted.	The reference label of the same attribute is deleted, meaning that the range is expanded.	Compatible	Before modification:
				typedef uri {
				type string;
				reference
				"RFC 3986: Uniform Resource Identifier (URI): Generic Syntax";
				...
				}
				After modification:
				typedef uri {
				type string; }

	The reference label is added.	The reference label of the same attribute is added, meaning that the range is narrowed.	Compatible	Before modification: typedef uri { type string; } After modification: typedef uri { type string; reference "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax"; ... }
when/ must	when/ must is changed.	**when/must** of the same attribute is changed.	Incompatible	Before modification: leaf isFlowKey { when "../../setId = 2"; } After modification: leaf isFlowKey { when "setName = 3"; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
	The when/ must label is deleted.	The when/ must label of the same attribute is deleted, meaning that the range is expanded.	Compatible	Before modification: leaf isFlowKey { when "../../setId = 2"; } After modification: leaf isFlowKey { ..... }

	The when/ must label is added.	The when/ must label of the same attribute is added, meaning that the range is narrowed.	Incompatible	Before modification:
				leaf isFlowKey {
				...
				}
				After modification:
				leaf isFlowKey {
				when "setName = 3"; }

	error-message of the must condition is added or deleted.	error-message of the must condition is added or deleted.	Compatible	Before modification:
				must "ifType != 'ethernet' or " +
				"(ifType = 'ethernet' and ifMTU = 1500)" {
				error-message "An ethernet MTU must be 1500";
				}
				After modification:
				must "ifType != 'ethernet' or " +
				"(ifType = 'ethernet' and ifMTU = 1500)";

	error-app-tag of the must condition is added or deleted.	error-app-tag of the must condition is added or deleted.	Compatible	Before modification:
				must "ifType != 'ethernet' or " +
				"(ifType = 'ethernet' and ifMTU = 1500)" {
				error-app-tag "must-violation";
				}
				After modification:
				must "ifType != 'ethernet' or " +
				"(ifType = 'ethernet' and ifMTU = 1500)";

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
if-feature label	The value of the if-feature label is changed.	The value of the if-feature label of the same attribute is changed.	Incompatible	Before modification: container location { if-feature has-gps; leaf longitude { mandatory true; ... } } After modification: container location { if-feature has-gps-xxx; leaf longitude { mandatory true; ... } }
	The if-feature label is deleted.	The if-feature label of the same attribute is deleted.	Compatible	Before modification: container location { if-feature has-gps; leaf longitude { mandatory true; ... } } After modification: container location { leaf longitude { mandatory true; ... } }

	The if-feature label is added.	The if-feature label of the same attribute is added.	Incompatible	Before modification:
				container location {
				leaf longitude {
				mandatory true;
				...
				}
				}
				After modification:
				container location {
				if-feature has-gps; leaf longitude { mandatory true; ... } }

submodule label	The value of the submodule label is changed.	The value of the submodule label of the same module is changed.	Incompatible	Before modification: submodule huawei-ac-ne-staticrt-staticrtbase After modification: submodule huawei-ac-ne-staticrt-staticrtbase-xxx

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
	The submodule label is added.	The submodule label of the same module is added.	Compatible	
	The submodule label is deleted.	The submodule label of the same module is deleted.	Incompatible	

belongs-to	belongs-to is added, deleted, or changed.	The belongs-to attribute of the same sub-module is changed.	Incompatible	Before modification:
				submodule huawei-qos-cbqos {
				belongs-to huawei-qos {
				prefix qos;
				}
				After modification:
				submodule huawei-qos-cbqos {
				belongs-to huawei-acl {
				prefix qos;
				}

module	The value of the module label is changed.	The value of the module label of the same module is changed.	Incompatible	Before modification: module huawei-ac-ne-staticrt After modification: module huawei-ac-ne-staticrt-xxx
prefix	The prefix statement is modified.	The prefix statement of the same module is changed.	Incompatible	Before modification: prefix "ac-ne-ifm"; After modification: prefix "ac-ne-ifm-xxx";

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
yang-version	yang-version is changed.	The value of yang-version can be either of the following: yang-version 1; yang-version 1.1; If this attribute is not explicitly specified, the default value of yang-version is 1. If the value of yang-version changes from 1 to 1.1 or from 1.1 to 1, the change is incompatible.	Incompatible	Before modification: yang-version 1.1; After modification: yang-version is not explicitly specified.

organization/ contact/ revision	organization/ contact/ revision is added or deleted.	organization/ contact/ revision is added or deleted.	Compatible	Example:
				organization
				"Huawei Technologies Co., Ltd.";
				contact
				"Huawei Industrial Base
				Bantian, Longgang
				Shenzhen 518129
				People's Republic of China
				Website: http://www.huawei.com
				Email: support@huawei.com";
				revision 2018-07-02 {
				description "
				Initial version.";
				}

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example
case	A case is added.	A case that does not exist is added to a node.	Compatible	Before modification: choice interface-type { case a { leaf ethernet { ... } } } After modification: choice interface-type { case a { leaf ethernet { ... } } case b { leaf loopback { ...} } }
	A case is deleted.	A case is deleted from a node.	Incompatible	Before modification: choice interface-type { case a { leaf ethernet { ... } } case b { leaf loopback { ...} } } After modification: choice interface-type { case a { leaf ethernet { ... } } }
namespace	namespace is changed.	The value of the namespace label is changed.	Incompatible	Before modification: namespace "urn:huawei:yang:huawei-ac-ne-ifm"; After modification: namespace "urn:huawei:yang:huawei-ac-ne-ifm-xxx";

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

Parameters carried in notification	Elements are added.	Parameters are suffixed to notification. They cannot be inserted from the middle.	Compatible	Before modification:
				notification node-down{
				leaf id {
				type leafref {
				path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id";
				}
				}
				}
				After modification:
				notification node-down{
				leaf id {
				type leafref {
				path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id";
				}
				}
				leaf admin-status {
				type types:admin-status;
				}
				}

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	Elements are deleted.	Parameters of the same notification are deleted.	Incompatible	Before modification:
				notification node-down{
				leaf id {
				type leafref {
				path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id";
				}
				}
				leaf admin-status { type types:admin-status; } }
				After modification:
				notification node-down{ leaf id { type leafref { path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id"; } } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	Elements are changed.	Parameters of the same notification are changed.	Incompatible	Before modification:
				notification node-down{
				leaf id {
				type leafref {
				path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id";
				}
				}
				leaf admin-status { type types:admin-status; } }
				After modification:
				notification node-down{ leaf id { type leafref { path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id"; } } leaf admin-status-xxx { type types:admin-status; } }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

identity	identity is added.	identity is added.	Compatible	Before modification:
				identity authentication-method {
				description
				"Base identity for user authentication methods.";
				}
				identity radius {
				base authentication-method;
				description
				"Indicates user authentication using RADIUS.";
				}
				After modification:
				identity authentication-method {
				description
				"Base identity for user authentication methods.";
				}
				identity radius {
				base authentication-method;
				description
				"Indicates user authentication using RADIUS.";
				}
				identity local-users { base authentication-method; description "Indicates password-based authentication of locally configured users."; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	identity is deleted.	identity is deleted.	Incompatible	Before modification:
				identity authentication-method {
				description
				"Base identity for user authentication methods.";
				}
				identity radius {
				base authentication-method;
				description
				"Indicates user authentication using RADIUS.";
				}
				identity local-users { base authentication-method; description "Indicates password-based authentication of locally configured users."; }
				After modification:
				identity authentication-method { description "Base identity for user authentication methods."; } identity radius { base authentication-method; description "Indicates user authentication using RADIUS."; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	identity is changed.	identity is changed.	Incompatible	Before modification:
				identity authentication-method {
				description
				"Base identity for user authentication methods.";
				}
				identity radius { base authentication-method; description "Indicates user authentication using RADIUS."; }
				After modification:
				identity authentication-method { description "Base identity for user authentication methods."; }
				identity radius-xxx { base authentication-method; description "Indicates user authentication using RADIUS."; }

base	base is added.	A base statement is added under the same identity.	Compatible	Before modification:
				identity des {
				base "crypto:crypto-alg";
				description "DES crypto algorithm.";
				}
				After modification:
				identity des {
				base "crypto:crypto-alg";
				base "crypto:symmetric-key";
				description "DES crypto algorithm.";
				}

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	base is deleted.	A base statement is deleted from the same identity.	Incompatible	Before modification:
				identity des {
				base "crypto:crypto-alg";
				base "crypto:symmetric-key";
				description "DES crypto algorithm.";
				}
				After modification:
				identity des {
				base "crypto:crypto-alg";
				description "DES crypto algorithm.";
				}

	base is changed.	A base statement is changed under the same identity.	Incompatible	Before modification:
				identity des {
				base "crypto:crypto-alg";
				description "DES crypto algorithm.";
				}
				After modification:
				identity des {
				base "crypto:symmetric-key";
				description "DES crypto algorithm.";
				}

extension	extension is added.	extension is added.	Compatible	Before modification:
				extension item {
				argument value;
				}
				After modification:
				extension item {
				argument value;
				}
				extension index { argument value; }

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

	extension is deleted.	extension is deleted.	Incompatible	Before modification:
				extension item {
				argument value;
				}
				extension index { argument value; }
				After modification:
				extension item { argument value; }

	extension is changed.	extension is changed.	Incompatible	Before modification:
				extension item {
				argument value;
				}
				After modification:
				extension item-xxx {
				argument value;
				}

argument	argument is changed.	The argument substatement of extension is modified. The argument substatement specifies the parameter name.	Incompatible	Before modification:
				extension c-define {
				argument index;
				}
				After modification:
				extension c-define {
				argument value;
				}

	argument is added or deleted.	argument is deleted, meaning that parameters of extension are deleted.
		argument is added, meaning that parameters of extension are added.
			Incompatible	Before modification:
				extension c-define {
				argument index;
				}
				After modification:
				extension c-define;

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

yin-element	yin-element is changed.	The value of yin-element changes from **true** or **false** or from **false** to **true**. The default value is **false**.	Incompatible	Before modification:
				extension c-define {
				argument index {
				yin-element true;
				}
				}
				After modification:
				extension c-define {
				argument index;
				}

revision-date	Addition	The revision-date substatement is added to the import and include statements.	Incompatible	include huawei-ac-ne-evnbgp-type {
				revision-date 2016-01-05;
				}

	Deletion	The revision-date substatement is deleted from the import and include statements.	Incompatible	
	Change	The revision-date substatement is modified in the import and include statements.	Incompatible	

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

augment	Addition	augment is added.	Compatible	The following definitions are added:
				augment "/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/l3vpn:site-network-access/l3vpn:service/l3vpn:qos/l3vpn:qos-profile/l3vpn:qos-profile" {
				case custom-unicom {
				container inbound-classes {
				uses class-profile;
				}
				container outbound-classes {
				uses class-profile;
				}
				}
				}

	Deletion and change	augment is deleted or changed.	Incompatible	The following definitions are modified or deleted:
				augment "/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/l3vpn:site-network-access/l3vpn:service/l3vpn:qos/l3vpn:qos-profile/l3vpn:qos-profile" {
				case custom-unicom {
				container inbound-classes {
				uses class-profile;
				}
				container outbound-classes {
				uses class-profile;
				}
				}
				}

Change Item	Change Name	Change Description	Compatible/ Incompatible	YANG Example

refine	Addition, deletion, or change	Addition, deletion, and change are considered as incompatible.	Incompatible	container connection {
				container source {
				uses target {
				refine "address" {
				description "Source IP address";
				}
				refine "port" {
				description "Source port number";
				}
				}
				}

presence	Change	The presence description is changed.	Compatible	Before modification:
				container "ssh" { presence "Enables SSH"; description "SSH service specific configuration"; // more leafs, containers and stuff here... }
				After modification:
				container "ssh" { presence "Enable SSH configuration"; description "SSH service specific configuration"; // more leafs, containers and stuff here... }

	Addition or deletion	Addition or deletion of the **presence** attribute	Incompatible	container "ssh" {
				presence "Enables SSH";
				description "SSH service specific configuration";
				// more leafs, containers and stuff here...
				}

ordered-by	Addition, deletion, or change	Addition, deletion, and change are considered as incompatible.	Incompatible	leaf-list cipher {
				type string;
				ordered-by user;
				description "A list of ciphers";
				}

The following table lists the compatible and incompatible upgrade specifications.

11 Service Security

The following lists the service security precautions for the AOC:

- Mapping template customization:

  The nodelete label can be added. The nodelete label is used to determine whether to delete device configurations triggered upon service deletion or modification.

  You can add the no_delete or no_delete_nested attribute based on the service logic so that the marked data will not be deleted upon the deletion of the service instance.

  – no_delete: A node marked with this label will not be deleted, but its child nodes can be deleted.

  – no_delete_nested: A node marked with this label and its child nodes will not be deleted.

  With this label, data will not be deleted, but data sources can be deleted. Data source (configuration source): Interface 0/0/1 and interface attributes attr1 (value 1) and attr2 (value 2) are configured for SSP instance 1 in the SSP package. Interface 0/0/1 and attributes attr1 and attr2 are labeled with the path of SSP instance 1, which is the data source.
- Data consistency customization: If a user enables the function of deleting device configurations during data consistency verification, the device configuration will be deleted during consistency verification.
- Whether the forwarder configuration can be deleted during consistency verification. The value **true** indicates that the forwarder configuration can be deleted, and the value **false** indicates that the forwarder configuration cannot be deleted.

  Example:

  ```
  syncToDel.key = "sync-to-del-enable"
    syncToDel.value = "true"
  ```
- Rollback logic:

  – Transaction rollback is supported: Based on the device's transaction capabilities, when an error is reported during command execution, the **cancel** command is executed. (The **cancel** command can be configured through **getCliDriverInfo** in the SND package.)

  – Transaction rollback is not supported: The system identifies commands that have been successfully executed based on the results returned by the

device, generates reverse packets based on these commands, and delivers the reverse packets to the device.

