Datacom Network Open Programmability V100R020C00

Development Guide

Issue	02
Date	2020-12-30

HUAWEI TECHNOLOGIES CO., LTD.

Copyright © Huawei Technologies Co., Ltd. 2020. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions

HUAWEI 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

Contents

1 Basic Knowledge

This section describes the basic knowledge required for open programmability system development, including the NETCONF, YANG, Jinja2, Python, and software package structure.

1.1 Understanding Programming Languages and Protocols

Before performing open programmability development, you need to master the programming languages and protocols listed in the following table.

Table 1-1 Open programmability programming languages and protocols

Item	Introduction	Learning Link
NETCONF	Network Configuration (NETCONF) protocol is a network device management protocol, which is similar to SNMP. NETCONF provides a framework mechanism for adding, modifying, and deleting network device configurations, and querying configurations, status, and statistics. NETCONF is a protocol defined by the IETF for installing, maintaining, and deleting configuration data on NEs. NETCONF operations are realized on the Remote Procedure Call (RPC) layer based on Extensible Markup Language (XML) data encoding. Open programmability uses NETCONF to communicate with NEs.	**https://tools.ietf.org/html/rfc6241**

Item	Introduction	Learning Link
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.	**https://tools.ietf.org/html/rfc7950**
Jinja2	Jinja2 is a modern and designer-friendly template engine for Python. Open programmability uses Jinja2 to quickly process service packets using templates.	**http://jinja.palletsprojects.com/en/2.10.x/**
Python	Python is an object-oriented high-level programming language. Due to its simple syntax, strong readability, wide application scope, and smooth learning curve, Python is used as the main software package development language.	**https://docs.python.org/3/tutorial/index.html**

1.2 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.

| |---Python Directory for storing the code developed using Python.

| |--- Code developed using Python

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

| |---template Directory for storing a template.

| |---resources Directory for storing resources.

| |---doc Directory for storing related documents, which is optional.

1.3 Log Records

1.3.1 Viewing Logs

Log Path

You can view logs at **/opt/oss/log/NCECOMMONE/OpenEMTestService/log/users/${user}.log**.

{$user} indicates the login user name, for example, /opt/oss/log/NCECOMMONE/OpenEMTestService/log/users/admin.log.

Log Format

[transId: first seven digits of a transaction ID] [module: module name] [pkgName: package name] [serviceName: service name] [deviceId: device ID]log content

Logging Modes in Running and Design States

- Logs in running state: Logs of each user are recorded only in the corresponding directory under **ExtendedRTService**. The absolute path of these logs is **/opt/oss/log/NCE/ExtendedPkgRTService**.

  The name of a user log file is in the format oss.{$group_name}.{$pkg_name $pkg_version}.trace, for example, **oss.group1.helloworld1.0.0.trace**.

  - {$group_name} indicates the name of the group where the third-party package developed by users is located.
  - {pkg_namepkg_version} indicates the name and version of the third-party package developed by users.
- Logs in design state: In debug mode, user logs are recorded in the specified paths of ExtendedRTService and OpenEMService.

1.3.2 Setting the Log Level

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.

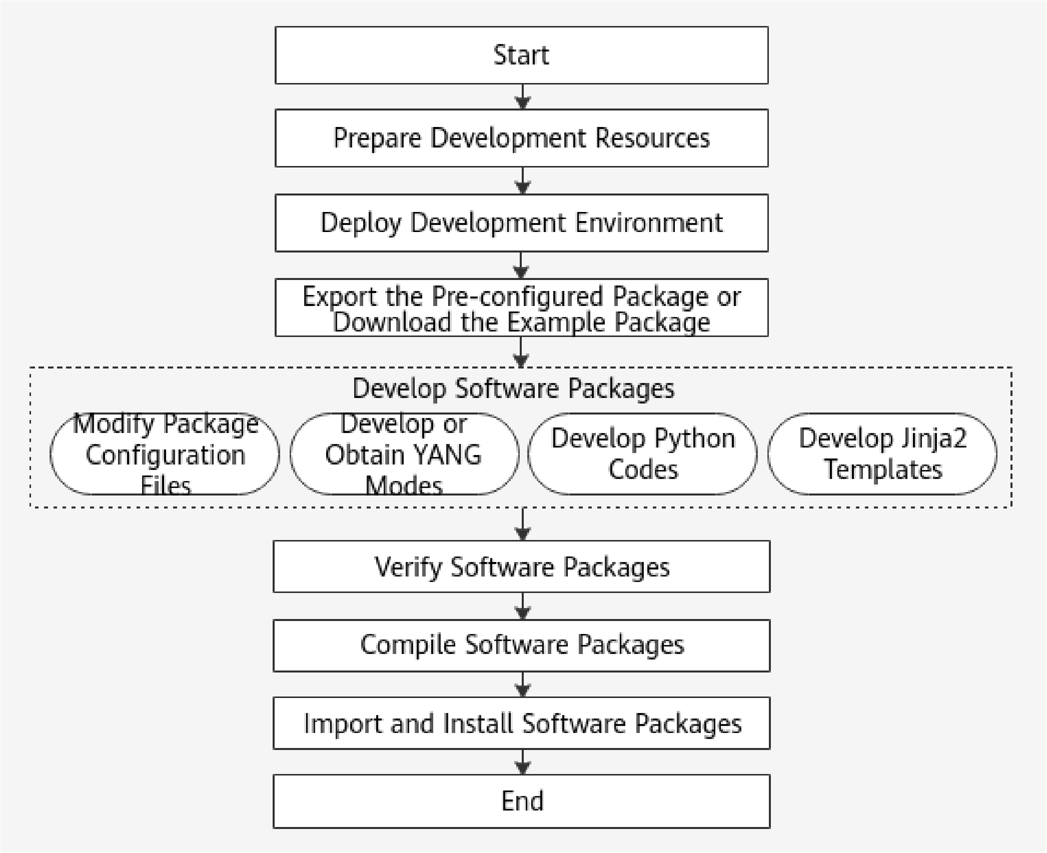
{$group_name} indicates the name of the group to which the user-developed third-party package belongs.

{$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


Start
Prepare Development Resources
Deploy Development Environment
Export the Pre-configured Package or Download the Example Package
Develop Software Packages
Modify Package Configuration Files
Develop or Obtain YANG Modes
Develop Python Codes
Develop Jinja2 Templates
Verify Software Packages
Compile Software Packages
Import and Install Software Packages
End

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://devzone.huawei.com/apistudio/sample/aoc/apiSdk.html** and download the SDK file.
2	yang-offline-util.zip	Offline verification tool for YANG models.	Log in to **https://devzone.huawei.com/apistudio/sample/aoc/apiSdk.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://devzone.huawei.com/apistudio/sample/aoc/apiSdk.html** and obtain the required sample code packages.

No.	Name	Description	How to Obtain
4	Open programmability mini software package	Provides an open, programmable, and lightweight software package. After downloading and installing the software package, you can configure devices and services on the local computer.	Log in to **https://devzone.huawei.com/apistudio/sample/aoc/apiSdk.html** and obtain the software package.

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 configured Python logic meets expected requirements in offline mode, you need to install the **open and programmable 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

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 SDK installation package from the matching software resource package.

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**.

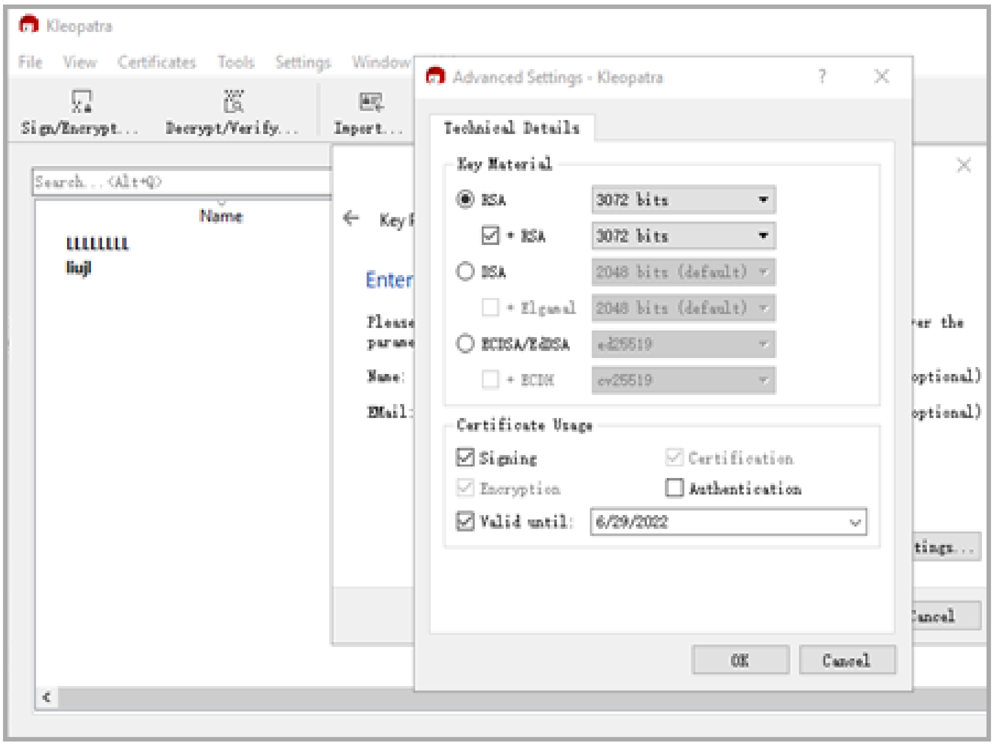
Step 3 Set **Name** and **Email** in **Enter Details**.

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



Key Material

● RSA 3072 bits

☑ + RSA 3072 bits

○ DSA 2048 bits (default)

□ + Elgamal 2048 bits (default)

○ ECDSA/EdDSA ed25519

□ + ECDH cv25519

Certificate Usage

☑ Signing ☑ Certification

☑ Encryption □ Authentication

☑ Valid until: 6/29/2022

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

4.5 Installing the Open Programmability Mini Software Package

To install the **mini software package** and start the service, perform the following steps:

Step 1 Move the obtained open programmable mini software package to a path that does not contain Chinese characters. It is recommended that the path not contain too many levels.

Step 2 Decompress the software package to the current directory.

Step 3 Double-click **start.bat** to start the open programmable mini service. The startup process takes about 3 to 5 minutes. The CLI cannot be closed after the service is started and is closed when the service is stopped.

Step 4 Enter **https://127.0.0.1:32018/aocwebsite** in the address box of the browser and press **Enter**. If the open programmability page is displayed, the service is started successfully.

----End

5 Developing an SSP Package

An SSP package can provide the following functions:

- EasyMap: decomposes network-layer services to NE-layer services. After devices are managed, the network layer directly delivers services to the devices.
- RPC: enables users to define functions as needed. The standard NETCONF or YANG configuration model cannot meet requirements. You can flexibly define functions, such as query operations.
- Discover: re-organizes NE-layer services into network-layer services. If NE-layer services exist before devices are managed, re-organize NE-layer services to the network layer after the devices are managed.

Sample code packages for open programming in typical scenarios have been uploaded to the open programming developer community of NCE. 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 logs.

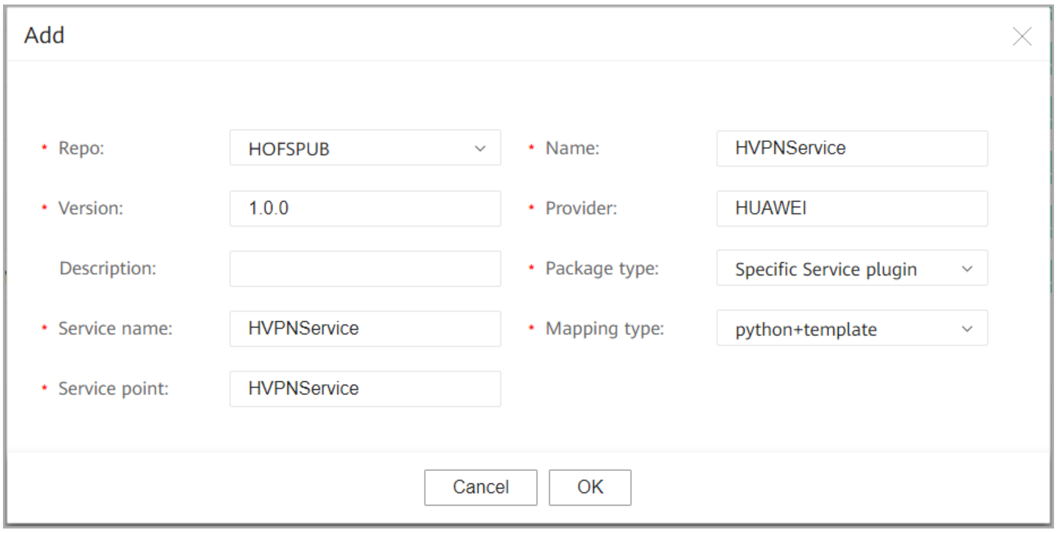
This chapter uses a simple example to describe how to develop an SSP package based on the default SSP package. In this example, the SSP package is used to create a VPN and bind sub-interfaces to the VPN.

5.1 Creating an SSP Package Template

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

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

Set **Package type** based on the software package type. In this example, since an SSP package is to be developed, select **Specific Service plugin** from the drop-down list.

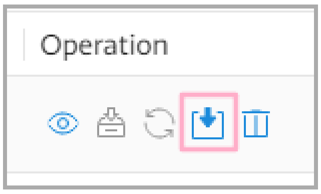


Add

- Repo: HOFSPUB
- Name: HVPNService
- Version: 1.0.0
- Provider: HUAWEI
- Description:
- Package type: Specific Service plugin
- Service name: HVPNService
- Mapping type: python+template
- Service point: HVPNService

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 SSP Package

5.2.1 Editing the SSP Configuration File

Step 1 Open PyCharm and click **Create New Project**. The **New Project** dialog box is displayed. Expand the **Project Interpreter** area, confirm the configuration, and click **Create**.

Step 2 Decompress the downloaded software pacakge 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 5-1 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	Packet type.	Mandatory
group	Package group information.	Optional
producer	Provider of the software package. For SND and GND packets, 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**.	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-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
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

Parameter	Description	Mandatory/ 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**, **key**, **java-class-name**, **Python-class-name**, **groovy-class-name**, and **template** fields must be specified. 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

The following is an example.

```
{
   "name": "HVPNService",
   "version": "1.0.0",
   "description": "5G",
   "package-type": "ssp",
   "producer": "HUAWEI",
   "service-name":"HVPNService",
   "nce-min-versions":[
       "2.0.0"
   ],
   "hooks": [
     {
       "type": "mapping",
       "key": "HVPNService",
       "Python-class-name": "HVPNService.HVPNService.AocNcs_servicepoint"
     }
   ]
}
```

----End

5.2.2 Compiling a Service YANG Model

In the navigation pane of the IDE, double-click the **HVPNService.yang** file in the yang directory to compile a YANG model. The YANG file is named based on the value of **service-name** in the **pkg.json** file.

EasyMap and Discover

The EasyMap and Discover functions are used to compile a YANG model based on the northbound input of the service model. Each module and node in the YANG model generates a northbound UI for users to set parameters.

In the following example, the name, MTU, and DES of the sub-interface along with the VPN name need to be defined in the YANG model to enable users to set them on UIs.

```
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"{
        list hvpnService {
             /*app:application-definition defines a servicepoint that can be identified by the system.
            The value of servicepoint must be the same as the value of the key field in the hooks parameter in
the pkg.json file. */
            app:application-definition "HVPNService";
            key "instanceName";
            leaf instanceName {
                type string;
            }
            list deviceList {
                key "deviceName";
                leaf deviceName {
                   type leafref {
                       path "/device:nes/device:ne/device:operate-name";
                   }
                   mandatory true;
                }
            }
            leaf name {
                type string;
            }

            leaf des {
                type string;
            }

            leaf mtu {
                type int32 {
                   range "46..9600";
               }
            }

            leaf vpn_instance_name {
                type string;
            }
        }
    }
}
```

RPC

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

```
    rpc service-rpc
    {
```

```
    description "get devices by device group id RPC";

    input
    {
      leaf ne-id
      {
  description "device ne id.";
        type string;
      }
    }

    output
    {
    list aoc-ecs-output{
      leaf result {
        type string;
      }
      list output-attribute-map {
         leaf name{
            type string;
          }
         leaf value{
            type string;
          }
        }
      }
    }
  }
```

Customized 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 Mapping Code

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

EasyMap

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.

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

# AocNcs_servicepoint inherits NcsService and overrides the ncs_map method.
class AocNcs_servicepoint(NcsService):
    # Override the ncs_map method to obtain the ifNumb and ParentName parameters required for creating an interface.
    def ncs_map(self, request, aoccontext=None, template=None):
        self.getIfNumb(request)
        self.getParentName(request)

        """
        The render function is used to map services at the network layer to the NE layer. request.xmldictnode is the encapsulated northbound data,
        and HVPNService/servicepoint.j2 is the NE-layer template.
        """
        return self.render('HVPNService/servicepoint.j2', request.xmldictnode)


    # Define the getIfNumb function to obtain ifNumb from the created sub-interface and update it to the parameter dictionary.
    def getIfNumb(self, request):
        # To obtain node data in the YANG model, you can use the x.y.z method to operate the data.
        sub_if_name = request.xmldictnode.hvpnService.name
        pointIndex = sub_if_name.find('.')
        ifNumb = sub_if_name[pointIndex + 1:]
        request.xmldictnode.update({"ifNumber": ifNumb})


    # Define the getParentName function to obtain ParentName from the created sub-interface and update ParentName to the dictionary.
    def getParentName(self, request):
        sub_if_name = request.xmldictnode.hvpnService.name
        pointIndex = sub_if_name.find('.')
        parentName = sub_if_name[0:pointIndex]
        request.xmldictnode.update({"parentName": parentName})
```

RPC

When a user customizes 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 case, 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.
from aoc import NcsService

class AocNcsexample(NcsService):
    def ncs_rpc(self, asrequest, arg1, arg2):
        ...
        return result
```

Discover

To implement the Discover function, derived classes need to inherit the NcsService base class and overwrite 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 aoc.ncs.ncsservice import NcsService
from aoc.ncs.ncs_model_pb2.discover_pb2 import DiscoverOutput
from aoc.sys import datastore

class AocNcsAaaService(NcsService):
    def ncs_map(self, request, aoccontext=None, template=None):
        self.logger.info(request.xml)
        result = self.render('template_Aaa.j2', request.xmldictnode)
        self.logger.info(result)
        return result


    def discover(self, discoverInput, aoc_context):
        self.logger.info(discoverInput)
        self.logger.info(aoc_context)

# Obtain device information from the input.
        deviceId = discoverInput.deviceInfo[0].deviceId
        deviceName = discoverInput.deviceInfo[0].deviceName

# Construct env.
        env = dict()
        env['device'] = deviceName

# Read the device RDB.
        path = '/huawei-ac-nes:inventory-cfg/nes/ne/' + deviceId + '/huawei-aaa:aaa/lam/users'
        users = datastore.read_datastore_rdb(aoc_context, path)
        root = self.xmltodictnode(users)
        result = DiscoverOutput()

# Parse the RDB result.
        for user in root.users.user:
            self.parse_rdb_user(env, user, result)
        self.logger.info(result)
        return result


    def parse_rdb_user(self, env, user, result):
        aaamin = result.serviceConfig.add()
        aaamin.servicePath = '/huawei-ac-applications:applications/aaamini:aaamini/' + user.userName
        env['username'] = user.userName
        env['password'] = user.password
        aaamin.serviceData = self.render('aaamin-discover.j2', env)
```

5.2.4 Developing a Jinja2 Template

There are two approaches to developing a Jinja2 template:

- If a device has been managed, obtain NETCONF packets delivered to the device through the NE management function and then deduce the Jinja2 template of corresponding services.
- Create a Jinja2 template based on the YANG model. This approach is suitable for experienced developers.

The following is an example of deducing the Jinja2 template of a service through NETCONF packets.

1. On the **NE Management** page, create a sub-interface and a VPN instance on a specified device.
2. Then, check the NETCONF packets to be delivered through the dry-run capability provided by the open programmability system.
3. Deduce a Jinja2 template or even YANG model of the service based on the obtained NETCONF packets.

Perform the following operations.

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 content of the NETCONF packets to be delivered to the device.

Step 5 Paste the copied NETCONF packet content to the **servicepoint.j2** file in the template directory of the SSP 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 The following is an example of the developed Jinja2 template.

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
  <nes>
    {% for neName in hvpnService.deviceList %}
    <ne>
      <neid>{{neName.deviceName| to_ne_id}}</neid>
      <ifm xmlns="http://www.huawei.com/netconf/vrp/huawei-ifm">
       <interfaces>
        <interface>
         <ifName>{{hvpnService.name}}</ifName>
         <ifNumber>{{ifNumber}}</ifNumber>
         <ifMtu>{{hvpnService.mtu}}</ifMtu>
         <ifParentIfName>{{parentName}}</ifParentIfName>
         <ifClass>subInterface</ifClass>
         <ifDescr>{{hvpnService.des}}</ifDescr>
        </interface>
       </interfaces>
      </ifm>
```

```
            {%- if hvpnService.vpn_instance_name %}
            <l3vpn xmlns="http://www.huawei.com/netconf/vrp/huawei-l3vpn">
              <l3vpncomm>
                <l3vpnInstances>
                  <l3vpnInstance>
                    <vrfName>{{hvpnService.vpn_instance_name}}</vrfName>
                    <l3vpnIfs>
                      <l3vpnIf>
                        <ifName>{{hvpnService.name}}</ifName>
                      </l3vpnIf>
                    </l3vpnIfs>
                  </l3vpnInstance>
                </l3vpnInstances>
              </l3vpncomm>
            </l3vpn>
            {%- endif %}
        </ne>
        {% endfor %}
    </nes>
</inventory-cfg>
```

----End

5.3 Verifying the SSP Package

5.3.1 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

5.3.2 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>**.

```
   <hvpnService xmlns="http://example.com/HVPNService">
        <deviceList>
```

```
            <deviceName>NE1</deviceName>
        </deviceList>
        <deviceList>
            <deviceName>NE2</deviceName>
        </deviceList>
        <name>GigabitEthernet3/0/0.801</name>
        <des>des</des>
        <mtu>1500</mtu>
        <vpn_instance_name>LTE_OMC</vpn_instance_name>
    </hvpnService>
```

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.
    <hvpnService xmlns="http://example.com/HVPNService">
        <deviceList>
            <deviceName>NE1</deviceName>
        </deviceList>
        <deviceList>
            <deviceName>NE2</deviceName>
        </deviceList>
        <name>GigabitEthernet3/0/0.801</name>
        <des>des</des>
        <mtu>1500</mtu>
        <vpn_instance_name>LTE_OMC</vpn_instance_name>
    </hvpnService>
    '''



    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 Jinija2 template correspond to those in the service YANG model.

----End

5.4 Developing an SSP 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

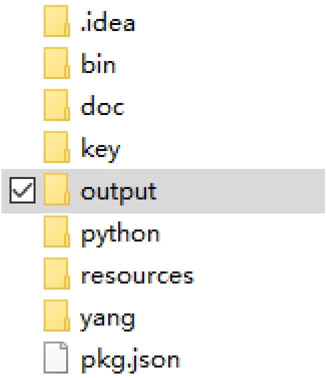
Step 3 Run the **makeFile.bat** file in the **bin** directory in the software package path to develop the software package.

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.



.idea
bin
doc
key
☑ output
python
resources
yang
pkg.json

```
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.

6 Importing and Installing Software Packages

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

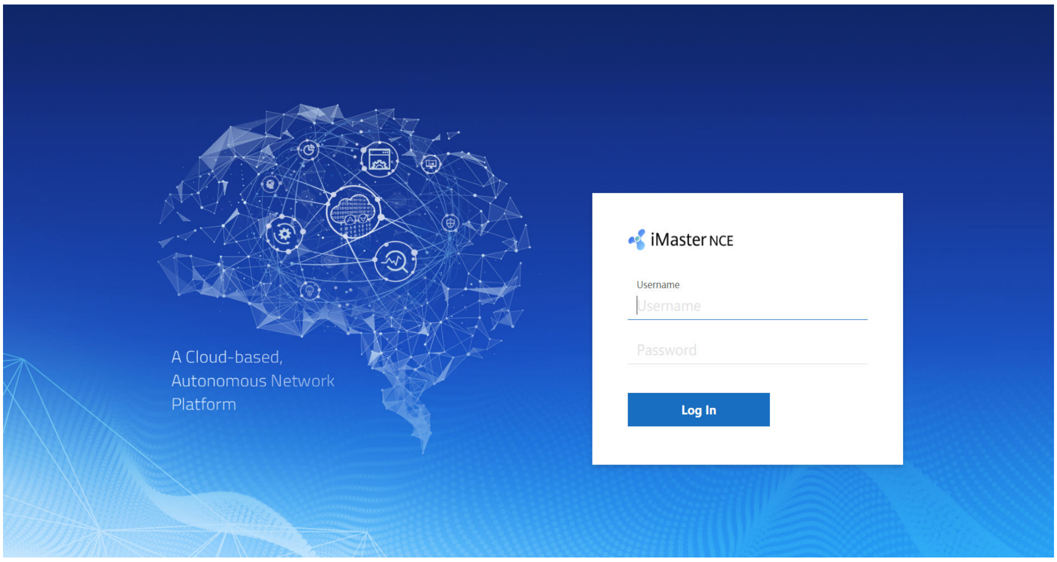
6.1 System Login

The open programmability system (OPS) provides two deployment modes: One is that the system is integrated into iMaster NCE-IP and released as an app (service open programmability). The app is deployed with iMaster NCE-IP. The other one is that the system is released as an independent software package. The open programmability mini software package is installed independently. This section describes how to log in to the system using a browser.

Logging In to the OPS 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 6-1 Logging in to the NCE O&M plane



A Cloud-based, Autonomous Network Platform

iMaster NCE

Username

Username

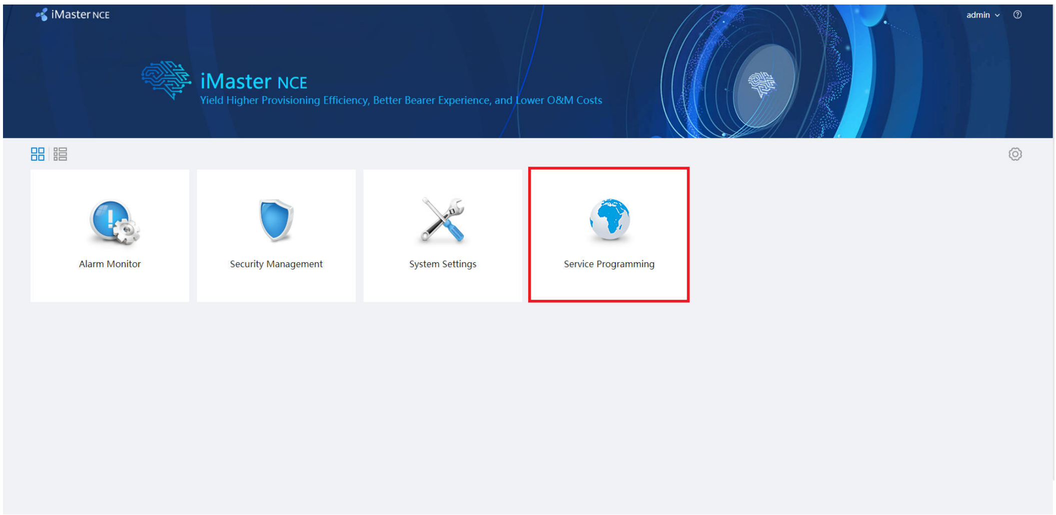
Password

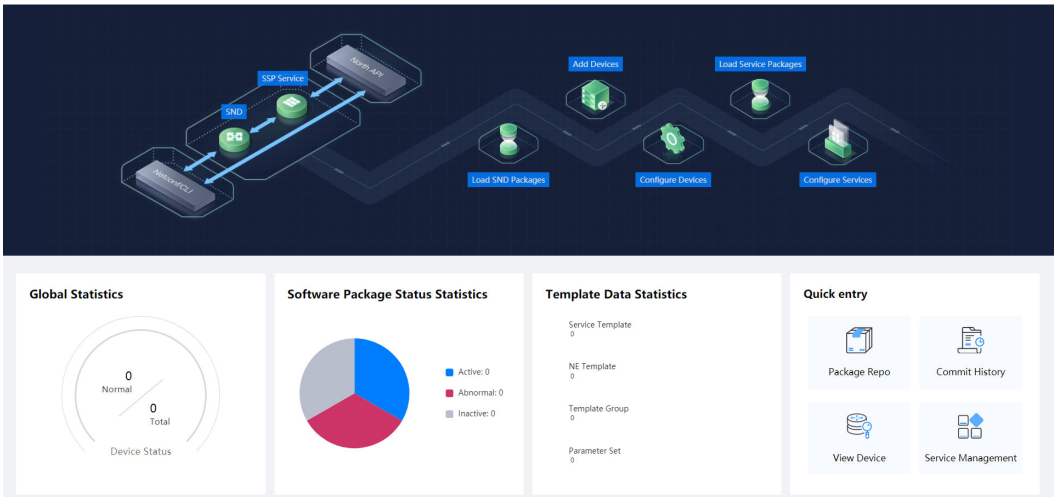
Log In

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 After logging in to the system, click **Service Programming** on the homepage to access the OPS.





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

Logging In to the open programmability Mini System

Step 1 Log in to the developer community and download the open programmability mini software package (**AOCmini_V100R020C00.zip**) on the resource download tab page.

Step 2 Decompress **AOCmini_V100R020C00.zip** and double-click **start.bat** to start the AOC mini service. The window is displayed.

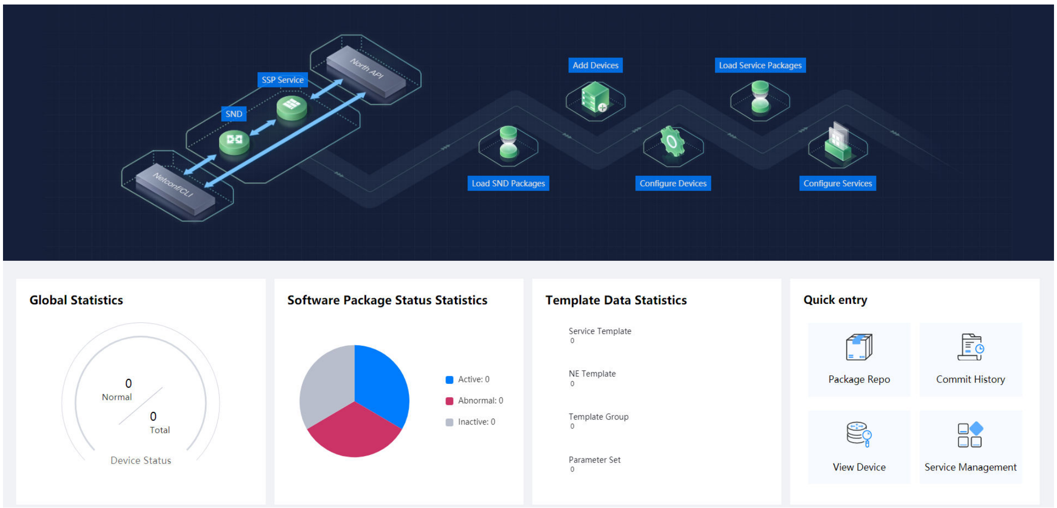
```
C:\Users\swx944510\Desktop\AOCmini\envs\Product-AOCService\controller>..\..\..\rtsp\tomcat\bin
\catalina.bat start
Using CATALINA_BASE:   "C:\Users\swx944510\Desktop\AOCmini/envs/Product-AOCService/"
Using CATALINA_HOME:   "C:\Users\swx944510\Desktop\AOCmini/rtsp/tomcat"
Using CATALINA_TMPDIR: "C:\Users\swx944510\Desktop\AOCmini/envs/Product-AOCService/\temp"
Using JRE_HOME:        "C:\Users\swx944510\Desktop\AOCmini/rtsp/jdk/"
Using CLASSPATH:       "C:\Users\swx944510\Desktop\AOCmini/rtsp/tomcat\bin\bootstrap.jar;C:\Users
\swx944510\Desktop\AOCmini/rtsp/tomcat\bin\tomcat-juli.jar"

==========================================
AOCmini is starting, please wait a moment.
```

Step 3 Wait for about three minutes until the system is started.

```
2020-09-10 16:10:28 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:33 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:39 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:43 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:48 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:53 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:58 Console message: AOCmini is starting, progress: 97.68%
2020-09-10 16:11:03 Console message: AOCmini is starting, progress: 97.68%
2020-09-10 16:11:08 Console message: AOCmini is starting, progress: 97.68%
2020-09-10 16:11:13 Console message: AOCmini started successfully, please visit https://127.0.0.1:32018/
aocwebsite/ in browser.
```

Step 4 Log in to the AOC mini system at https://127.0.0.1:32018/aocwebsite.

Figure 6-2 Home page of the AOC mini system



Global Statistics

0 Normal

0 Total

Device Status

Software Package Status Statistics

- Active: 0
- Abnormal: 0
- Inactive: 0

Template Data Statistics

Service Template 0

NE Template 0

Template Group 0

Parameter Set 0

Quick entry

Package Repo | Commit History | View Device | Service Management

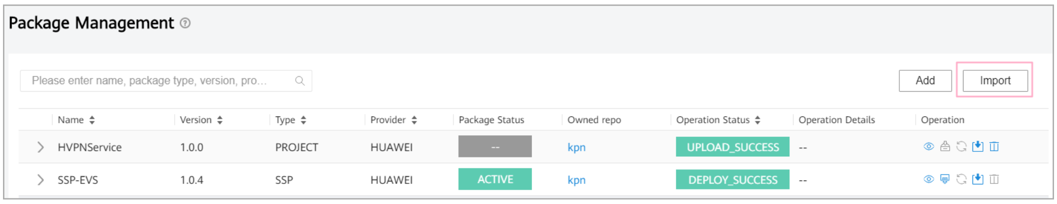
Step 5 On the home page, click the corresponding shortcut entry or click any shortcut entry based on the actual application scenario to access the main menu.

----End

6.2 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 6-3 Importing a software package


Package Management

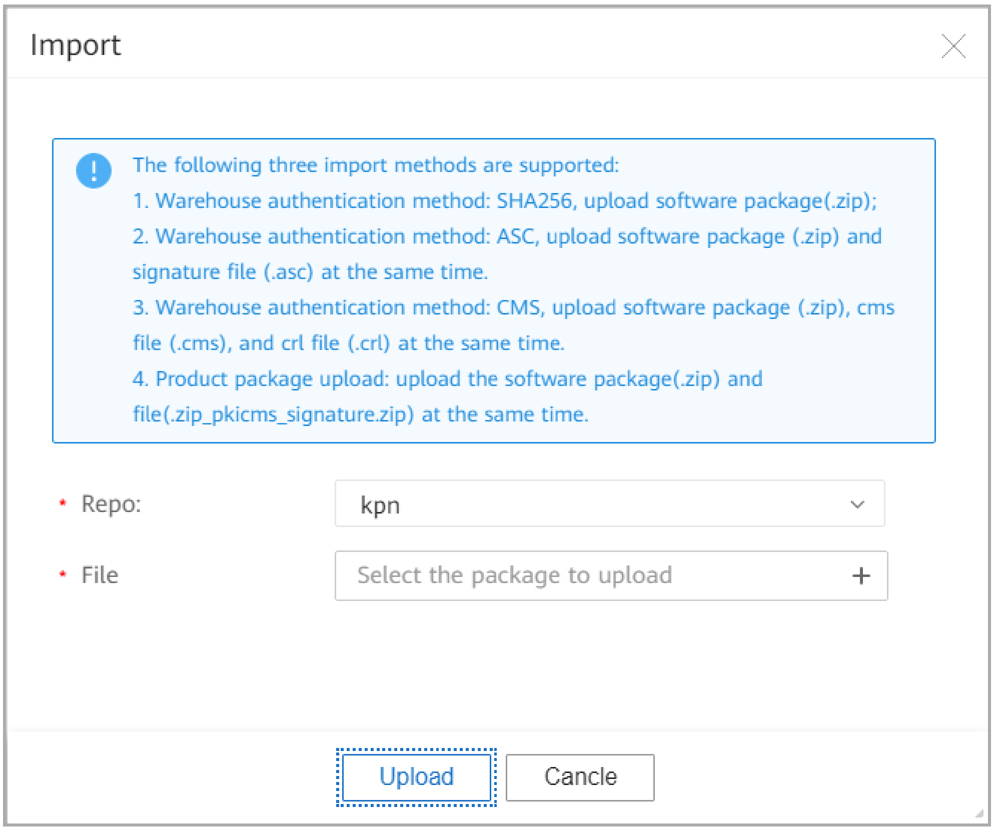
Please enter name, package type, version, pro...

Add | Import

Name	Version	Type	Provider	Package Status	Owned repo	Operation Status	Operation Details	Operation
HVPNService	1.0.0	PROJECT	HUAWEI	--	kpn	UPLOAD_SUCCESS	--	
SSP-EVS	1.0.4	SSP	HUAWEI	ACTIVE	kpn	DEPLOY_SUCCESS	--	

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

Figure 6-4 Selecting the software package to be imported



The following three import methods are supported:

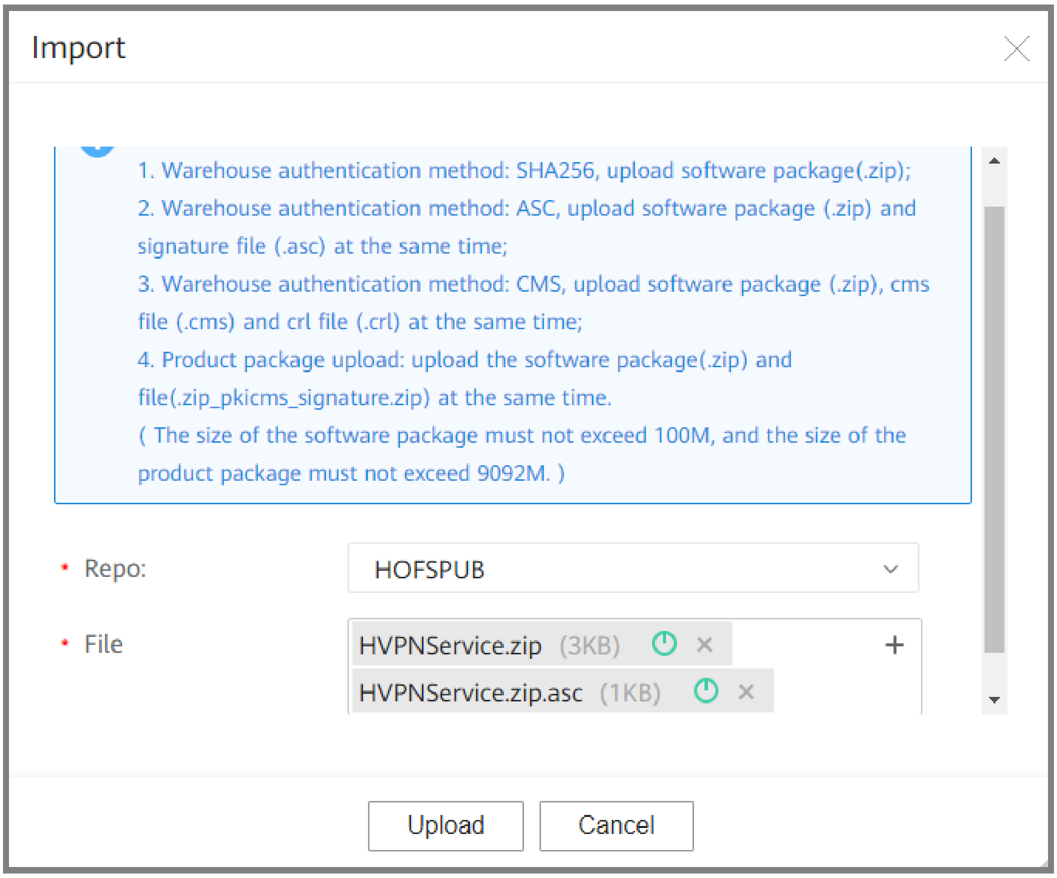
1. Warehouse authentication method: SHA256, upload software package(.zip);
2. Warehouse authentication method: ASC, upload software package (.zip) and signature file (.asc) at the same time.
3. Warehouse authentication method: CMS, upload software package (.zip), cms file (.cms), and crl file (.crl) at the same time.
4. Product package upload: upload the software package(.zip) and file(.zip_pkicms_signature.zip) at the same time.

* Repo: kpn

* File: Select the package to upload

Step 3 Click **Upload**.

Figure 6-5 Clicking **Upload**



Import

1. Warehouse authentication method: SHA256, upload software package(.zip);
2. Warehouse authentication method: ASC, upload software package (.zip) and signature file (.asc) at the same time;
3. Warehouse authentication method: CMS, upload software package (.zip), cms file (.cms) and crl file (.crl) at the same time;
4. Product package upload: upload the software package(.zip) and file(.zip_pkicms_signature.zip) at the same time.

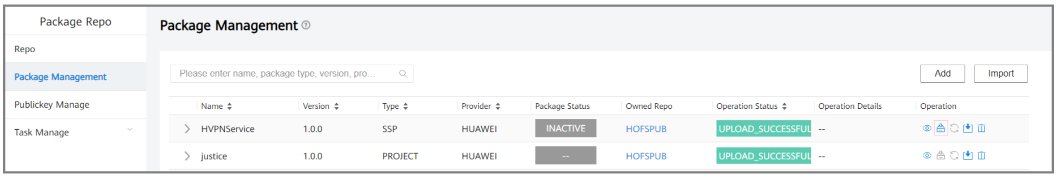
(The size of the software package must not exceed 100M, and the size of the product package must not exceed 9092M.)

* Repo: HOFSPUB

* File HVPNService.zip (3KB) HVPNService.zip.asc (1KB)

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

Figure 6-6 Software package import success



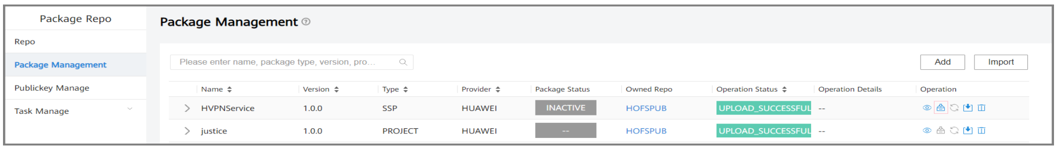
Package Management

Please enter name, package type, version, pro...

Name	Version	Type	Provider	Package Status	Owned Repo	Operation Status	Operation Details	Operation
HVPNService	1.0.0	SSP	HUAWEI	INACTIVE	HOFSPUB	UPLOAD_SUCCESSFUL	--	
justice	1.0.0	PROJECT	HUAWEI	--	HOFSPUB	UPLOAD_SUCCESSFUL	--	

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

Figure 6-7 Installing a software package



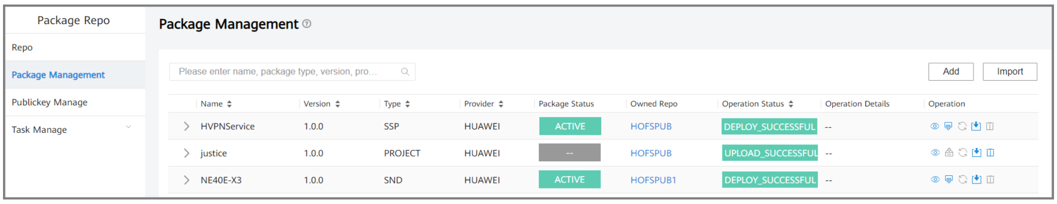
Package Management

Please enter name, package type, version, pro...

Name	Version	Type	Provider	Package Status	Owned Repo	Operation Status	Operation Details	Operation
HVPNService	1.0.0	SSP	HUAWEI	INACTIVE	HOFSPUB	UPLOAD_SUCCESSFUL	--	
justice	1.0.0	PROJECT	HUAWEI	--	HOFSPUB	UPLOAD_SUCCESSFUL	--	

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 6-8 Software package import success



Name	Version	Type	Provider	Package Status	Owned Repo	Operation Status	Operation Details	Operation
HVPNService	1.0.0	SSP	HUAWEI	ACTIVE	HOFSPUB	DEPLOY_SUCCESSFUL	--	
justice	1.0.0	PROJECT	HUAWEI	--	HOFSPUB	UPLOAD_SUCCESSFUL	--	
NE40E-X3	1.0.0	SND	HUAWEI	ACTIVE	HOFSPUB1	DEPLOY_SUCCESSFUL	--	

----End