JUNIPER NETWORKS | Engineering Simplicity

Juniper Extension Toolkit Developer Guide

Published
2020-06-22

JUNOS

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

Juniper Extension Toolkit Developer Guide

The information in this document is current as of the date on the title page.

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement ("EULA") posted at https://support.juniper.net/support/eula/. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

About the Documentation

IN THIS SECTION

Use this guide to develop, deploy, use, and debug JET applications that are developed on Junos OS and third party applications. For information about JET APIs, see the *Juniper Extension Toolkit API Guide*.

Documentation and Release Notes

To obtain the most current version of all Juniper Networks® technical documentation, see the product documentation page on the Juniper Networks website at https://www.juniper.net/documentation/.

If the information in the latest release notes differs from the information in the documentation, follow the product Release Notes.

Juniper Networks Books publishes books by Juniper Networks engineers and subject matter experts. These books go beyond the technical documentation to explore the nuances of network architecture, deployment, and administration. The current list can be viewed at https://www.juniper.net/books.

Using the Examples in This Manual

If you want to use the examples in this manual, you can use the **load merge** or the **load merge relative** command. These commands cause the software to merge the incoming configuration into the current candidate configuration. The example does not become active until you commit the candidate configuration.

If the example configuration contains the top level of the hierarchy (or multiple hierarchies), the example is a *full example*. In this case, use the **load merge** command.

If the example configuration does not start at the top level of the hierarchy, the example is a *snippet*. In this case, use the **load merge relative** command. These procedures are described in the following sections.

Merging a Full Example

To merge a full example, follow these steps:

1. From the HTML or PDF version of the manual, copy a configuration example into a text file, save the file with a name, and copy the file to a directory on your routing platform.

   For example, copy the following configuration to a file and name the file **ex-script.conf**. Copy the **ex-script.conf** file to the **/var/tmp** directory on your routing platform.

   ```
   system {
       scripts {
           commit {
               file ex-script.xsl;
           }
       }
   }
   interfaces {
       fxp0 {
           disable;
           unit 0 {
               family inet {
                   address 10.0.0.1/24;
               }
           }
       }
   }
   ```

2. Merge the contents of the file into your routing platform configuration by issuing the **load merge** configuration mode command:

   ```
   [edit]
   user@host# load merge /var/tmp/ex-script.conf
   load complete
   ```

Merging a Snippet

To merge a snippet, follow these steps:

1. From the HTML or PDF version of the manual, copy a configuration snippet into a text file, save the file with a name, and copy the file to a directory on your routing platform.

   For example, copy the following snippet to a file and name the file **ex-script-snippet.conf**. Copy the **ex-script-snippet.conf** file to the **/var/tmp** directory on your routing platform.

   ```
   commit {
     file ex-script-snippet.xsl; }
   ```

2. Move to the hierarchy level that is relevant for this snippet by issuing the following configuration mode command:

   ```
   [edit]
   user@host# edit system scripts
   [edit system scripts]
   ```

3. Merge the contents of the file into your routing platform configuration by issuing the **load merge relative** configuration mode command:

   ```
   [edit system scripts]
   user@host# load merge relative /var/tmp/ex-script-snippet.conf
   load complete
   ```

For more information about the **load** command, see CLI Explorer.

Documentation Conventions

Table 1 on page ix defines notice icons used in this guide.

Table 1: Notice Icons

Icon	Meaning	Description
	Informational note	Indicates important features or instructions.
	Caution	Indicates a situation that might result in loss of data or hardware damage.
	Warning	Alerts you to the risk of personal injury or death.
	Laser warning	Alerts you to the risk of personal injury from a laser.
	Tip	Indicates helpful information.
	Best practice	Alerts you to a recommended use or implementation.

Table 2 on page ix defines the text and syntax conventions used in this guide.

Table 2: Text and Syntax Conventions

Convention	Description	Examples
Bold text like this	Represents text that you type.	To enter configuration mode, type the **configure** command: user@host> **configure**
`Fixed-width text like this`	Represents output that appears on the terminal screen.	`user@host>` **show chassis alarms** `No alarms currently active`
Italic text like this	• Introduces or emphasizes important new terms. • Identifies guide names. • Identifies RFC and Internet draft titles.	• A policy *term* is a named structure that defines match conditions and actions. • *Junos OS CLI User Guide* • RFC 1997, *BGP Communities Attribute*

Table 2: Text and Syntax Conventions *(continued)*

Convention	Description	Examples
Italic text like this	Represents variables (options for which you substitute a value) in commands or configuration statements.	Configure the machine's domain name: [edit] root@# **set system domain-name** *domain-name*
Text like this	Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components.	• To configure a stub area, include the **stub** statement at the **[edit protocols ospf area area-id]** hierarchy level. • The console port is labeled **CONSOLE**.
< > (angle brackets)	Encloses optional keywords or variables.	**stub <default-metric *metric*>;**
| (pipe symbol)	Indicates a choice between the mutually exclusive keywords or variables on either side of the symbol. The set of choices is often enclosed in parentheses for clarity.	**broadcast | multicast** **(*string1* | *string2* | *string3*)**
# (pound sign)	Indicates a comment specified on the same line as the configuration statement to which it applies.	**rsvp { # Required for dynamic MPLS only**
[ ] (square brackets)	Encloses a variable for which you can substitute one or more values.	**community name members [*community-ids*]**

Indention and braces ({ })	Identifies a level in the configuration hierarchy.	[edit]
		routing-options {
		  static {
		  route default {
		  nexthop *address*;
		  retain;
		  }
		  }
		}

; (semicolon)	Identifies a leaf statement at a configuration hierarchy level.	
GUI Conventions		

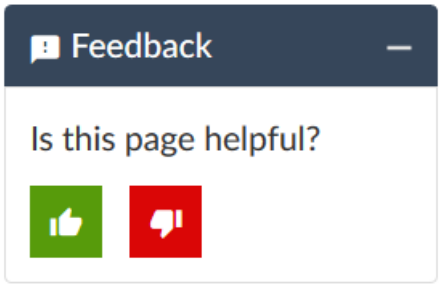
Table 2: Text and Syntax Conventions *(continued)*

Convention	Description	Examples
Bold text like this	Represents graphical user interface (GUI) items you click or select.	• In the Logical Interfaces box, select **All Interfaces**. • To cancel the configuration, click **Cancel**.
> (bold right angle bracket)	Separates levels in a hierarchy of menu selections.	In the configuration editor hierarchy, select **Protocols>Ospf**.

Documentation Feedback

We encourage you to provide feedback so that we can improve our documentation. You can use either of the following methods:

- Online feedback system—Click TechLibrary Feedback, on the lower right of any page on the Juniper Networks TechLibrary site, and do one of the following:



  - Click the thumbs-up icon if the information on the page was helpful to you.
  - Click the thumbs-down icon if the information on the page was not helpful to you or if you have suggestions for improvement, and use the pop-up form to provide feedback.
- E-mail—Send your comments to techpubs-comments@juniper.net. Include the document or topic name, URL or page number, and software version (if applicable).

Requesting Technical Support

Technical product support is available through the Juniper Networks Technical Assistance Center (JTAC). If you are a customer with an active Juniper Care or Partner Support Services support contract, or are

covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

- JTAC policies—For a complete understanding of our JTAC procedures and policies, review the *JTAC User Guide* located at https://www.juniper.net/us/en/local/pdf/resource-guides/7100059-en.pdf.
- Product warranties—For product warranty information, visit https://www.juniper.net/support/warranty/.
- JTAC hours of operation—The JTAC centers have resources available 24 hours a day, 7 days a week, 365 days a year.

Self-Help Online Tools and Resources

For quick and easy problem resolution, Juniper Networks has designed an online self-service portal called the Customer Support Center (CSC) that provides you with the following features:

- Find CSC offerings: https://www.juniper.net/customers/support/
- Search for known bugs: https://prsearch.juniper.net/
- Find product documentation: https://www.juniper.net/documentation/
- Find solutions and answer questions using our Knowledge Base: https://kb.juniper.net/
- Download the latest versions of software and review release notes: https://www.juniper.net/customers/csc/software/
- Search technical bulletins for relevant hardware and software notifications: https://kb.juniper.net/InfoCenter/
- Join and participate in the Juniper Networks Community Forum: https://www.juniper.net/company/communities/
- Create a service request online: https://myjuniper.juniper.net

To verify service entitlement by product serial number, use our Serial Number Entitlement (SNE) Tool: https://entitlementsearch.juniper.net/entitlementsearch/

Creating a Service Request with JTAC

You can create a service request with JTAC on the Web or by telephone.

- Visit https://myjuniper.juniper.net.
- Call 1-888-314-JTAC (1-888-314-5822 toll-free in the USA, Canada, and Mexico).

For international or direct-dial options in countries without toll-free numbers, see https://support.juniper.net/support/requesting-support/.

1

CHAPTER

Getting Started

JET Overview

IN THIS SECTION

Juniper Extension Toolkit (JET), an evolution of the Junos SDK, provides a modern, programmatic interface for developers of third-party applications on Junos devices. It focuses on providing a standards-based interface to the Juniper Networks Junos operating system (Junos OS) for customizing management and control plane functionality.

JET also includes a virtual machine (VM) packaged as a vagrant file, libraries, and other tools to enable developers to write on-device JET applications.

Benefits of JET

- Provides APIs to interact with any device running Junos OS.
- Supports API development in multiple languages.
- Provides tools to develop applications that run on Junos devices.
- Uses an event notification method that enables JET applications to respond to selected system events.

JET Architecture

JET is a framework that enables developers to create applications that extend the functionality of Junos OS. For example, a JET application might extend the Junos CLI by adding a new operational command to show application-specific states. JET applications can run on devices running Junos OS or on another device in your operating environment and connect over the network to a device running Junos OS.

JET applications interact with Junos OS through request-response and notification services over standards based transport channels. Figure 1 on page 15 illustrates the request-response and notification services.

Figure 1: JET Request-Response and Notification Services

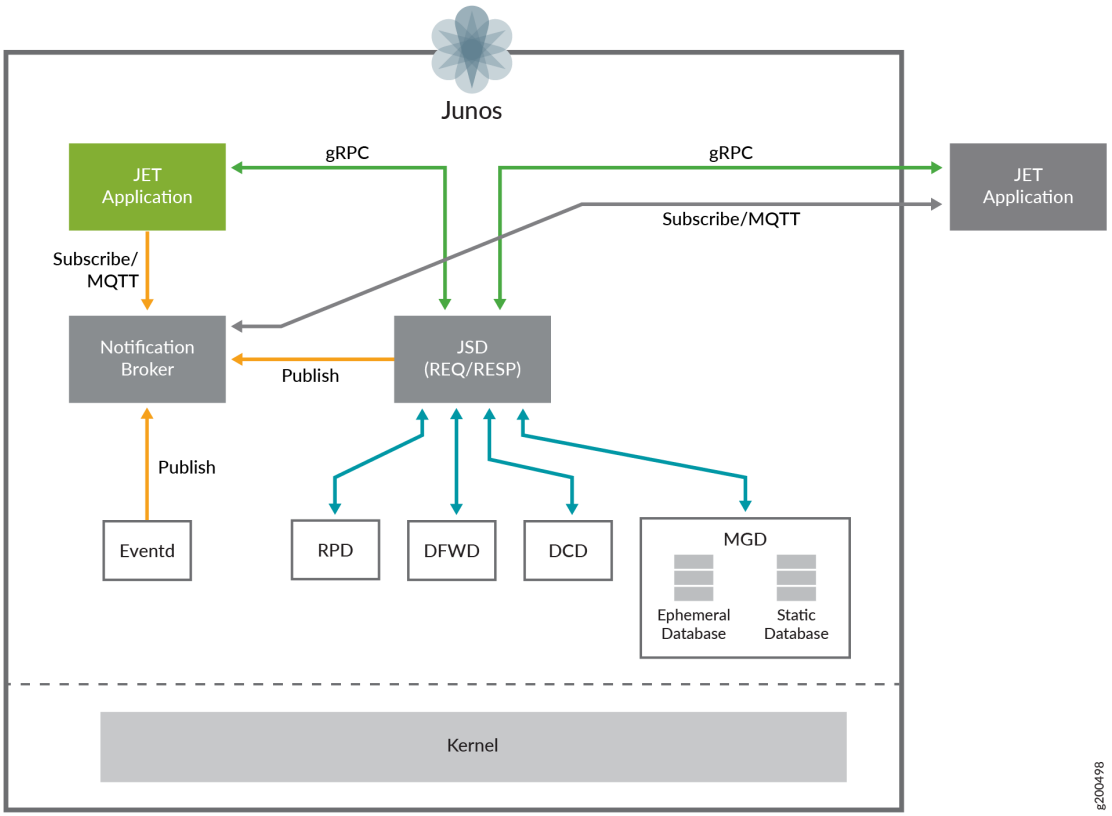


Table 3 on page 15 describes the request-response and notification services.

Table 3: JET Applications Interact with Junos OS Through Services

Service	Description
Request-response—An application can issue a request and wait for the response from Junos OS.	JET services daemon (jsd), which runs on Junos OS, provides the request-response service. When jsd receives a request (by default on TCP port 32767), it creates a new session to service the JET application. The session remains alive as long as the client and server are both up and communicating with each other. Over the lifetime of a session, jsd can execute any number of APIs. jsd can support a maximum of 8 active client sessions and execute APIs from these sessions in parallel. **NOTE:** For secure communications with jsd, use RSA certificates, specifically TLSv1.2 (minimum).

Table 3: JET Applications Interact with Junos OS Through Services *(continued)*

Service	Description
Notifications—An application can receive asynchronous notifications of events happening on Junos OS.	JET provides a publish-subscribe based messaging protocol and a notification broker. JET applications can register with the notification broker and inform the broker about the topics of interest to receive messages. The broker is responsible for distributing messages to the interested clients based on the topic of the message. Junos OS daemons publishing the events (such as eventd) connect to the broker as a publisher and publish the events. JET utilizes Message Queue Telemetry Transport (MQTT) protocol (see https://mqtt.org/) method to implement the notification service.

JET and gRPC

Starting in Junos OS Release 16.2R1, JET supports the gRPC framework for remote procedure calls (RPCs). JET uses gRPC for cross-language services as a mechanism to enable request-response service. gRPC also provides a mechanism to define APIs that are programming language agnostic. For more information, see https://www.grpc.io/.

Release History Table

Release	Description
16.2R1	Starting in Junos OS Release 16.2R1, JET supports the gRPC framework for remote procedure calls (RPCs).

Set Up the JET VM

IN THIS SECTION

Overview

JET provides a development environment that you can download from the Juniper Networks download site. The JET bundle includes a virtual machine (VM) packaged as a vagrant file, the JET toolchain, plug-ins, and other tools and libraries that are required for developing on-device or off-device applications. The JET VM is based on 64-bit Ubuntu 12.04 long-term support release.

If you are developing an application with a dependency on C or C++ modules or developing a signed application, you must use the JET VM for JET application development.

Download the Packages You Need

To use the JET VM, download and install the following packages:

- **Vagrant**

  Vagrant (https://www.vagrantup.com/) is a software that creates and configures virtual development environments. You can think of it as a higher-level wrapper around virtualization software such as VirtualBox (https://www.virtualbox.org/wiki/Downloads). You can use Vagrant to manage the JET development VM.

  To download Vagrant, go to https://www.vagrantup.com/ and download Vagrant for your system's platform (Windows, Mac, or Linux).

- **VirtualBox**

  For the JET VM, Juniper Networks supports only the VirtualBox hypervisor. To download and install VirtualBox:

  1. Go to https://www.virtualbox.org/wiki/Downloads.
  2. Download and install the VirtualBox package for your platform and the VirtualBox extension package.
  3. Enable hardware virtualization support on your machine BIOS if it is not already enabled.

- **JET Files**

  Download the JET VM from the Juniper Networks downloads website in the form of the following packages:

  - JET client IDL library
  - JET sandbox and toolchain file
  - **JET-vagrant.zip** file

Initialize Your Virtual Machine

To start the JET VM:

1. Create a **jet-vagrant** directory.

2. Extract the **JET-vagrant.zip** file you downloaded from the Juniper Networks download site to the **jet-vagrant** directory.

3. Change to the **jet-vagrant** directory where you have extracted the **JET-vagrant.zip** file.

4. Run the **vagrant up** command in the **jet-vagrant** directory.

   > **NOTE:** Before using the JET VM, wait for the installation of all the packages after running the **vagrant up** command.

   If needed, use the following default login credentials:

   ```
   username: vagrant
   password: vagrant
   ```

Set Up Your Virtual Machine Environment

To set the PATH variable and prepare the JET VM:

1. Extract and install the **junos-jet-XX.YRZ.S.tar.gz** package using the **./install** command.

   > **NOTE:** Run the **sudo ./install** command if you are not logged in as root user.

   In the **junos-jet-XX.YRZ.S.tar.gz** package name:

   - XX is the main release number of the product, for example, 18.
   - Y is the minor release number of the product, for example, 3.
   - R is the type of software release, for example, R for FRS or maintenance release.
   - Z is the build number of the product, for example, 1, indicating the FRS rather than a maintenance release.
   - S is the spin number of the product, for example, 13.
2. Open a terminal in the VM.
3. Add the absolute path to the **/junos-jet-XX.YRZ.S.tar.gz/bin** directory to the PATH variable in **.bashrc**.

   ```
   vagrant@jet-vm:~$ echo
   'PATH=$PATH:/usr/local/junos-jet/18.3R1/junos-jet-XX.YRZ.S.tar.gz'>>~/.bashrc
   ```
4. Run the following command to display the JET **XX.YRZ.S.tar.gz** path in the output:

   ```
   vagrant@jet-vm:~$ source ~/.bashrc
   ```
5. Run the **env** command to ensure the PATH variable contains the directory path you just added.

   ```
   vagrant@jet-vm:~$ env

   PATH=/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
    /bin:/usr/games:/usr/local/junos-jet/18.3R1/junos-jet-XX.YRZ.S.tar.gz/bin
   ```

You are ready to develop applications in the JET VM.

WHAT'S NEXT

Develop Off-Device JET Applications | 22

Develop On-Device JET Applications | 34

2

CHAPTER

Application Development

Develop Off-Device JET Applications

IN THIS SECTION

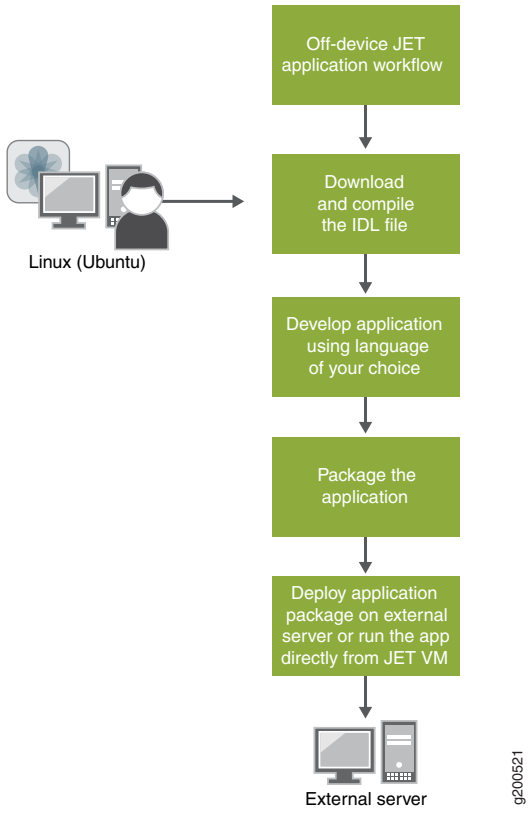
Overview

You can use JET to develop applications that run off-device. This allows you to leverage the benefits of JET on all devices on your network. For ease of development, you can write off-device JET applications in the language of your choice. To develop an off-device application:

1. Download and compile the IDL file.
2. Develop the application using the language of your choice.
3. Package the application.
4. Deploy the application package on an external server or run the application directly from the JET VM.

Figure 2 on page 23 shows the off-device application development workflow.

Figure 2: Off-Device JET Application Workflow



Download and Compile the IDL File

1. Download the IDL file from the Juniper Networks downloads website.

2. Unpack the IDL file.

   For example, if you are using Junos OS Release 18.4R1 or later:

```
ubuntu-16:~ jet$ mkdir proto
ubuntu-16:~ jet$ tar -xzf jet-idl-18.4-20181107.0.tar.gz -C proto/
ubuntu-16:~ jet$ ls proto/

1   2 README
```

```
ubuntu-16:~ jet$ ls proto/2

jnx_authentication_service.proto
jnx_common_addr_types.proto
jnx_common_base_types.proto
jnx_management_service.proto
jnx_registration_service.proto
```

For Junos OS releases prior to 18.4R1, the process is slightly different:

```
ubuntu-16:~ jet$ tar -xzf jet-idl-18.2R1.9.tar.gz
ubuntu-16:~ jet$ ls proto

any.proto
dcd_service.proto
jnx_addr.proto
mpls_api_service.proto
registration_service.proto
authentication_service.proto
firewall_service.proto
jnx_base_types.proto
prpd_common.proto
rib_service.proto
bgp_route_service.proto
gnmi.proto
management_service.proto
prpd_service.proto
routing_interface_service.proto
```

3. Compile Python and gRPC modules for authentication and management service proto files.

   For example, if you are using Junos OS Release 18.4R1 or later:

```
ubuntu-16:~ jet$ python -m grpc_tools.protoc -I./proto/2 --python_out=. --grpc_python_out=.
proto/2/jnx_management_service.proto
ubuntu-16:~ jet$ python -m grpc_tools.protoc -I./proto/2 --python_out=. --grpc_python_out=.
proto/2/jnx_authentication_service.proto
ubuntu-16:~ jet$ python -m grpc_tools.protoc -I./proto/2 --python_out=. --grpc_python_out=.
proto/2/jnx_common_base_types.proto
ubuntu-16:~ jet$ ls -lrt
```

```
total 112
-rw-r--r-- 1 vagrant vagrant 52683 Nov  8 16:47 jet-idl-18.4-20181107.0.tar.gz
drwxr-xr-x 1 vagrant vagrant   170 Nov  8 16:49 proto
-rw-r--r-- 1 vagrant vagrant 40924 Nov  8 16:56 jnx_management_service_pb2.py
-rw-r--r-- 1 vagrant vagrant  4719  Nov  8 16:56
jnx_management_service_pb2_grpc.py
-rw-r--r-- 1 vagrant vagrant  5365  Nov 8  2018 jnx_authentication_service_pb2.py
-rw-r--r-- 1 vagrant vagrant  1898  Nov 8  2018
jnx_authentication_service_pb2_grpc.py
-rw-r--r-- 1 vagrant vagrant  6391  Nov 8  2018 jnx_common_base_types_pb2.py
-rw-r--r-- 1 vagrant vagrant    83    Nov 8  2018
jnx_common_base_types_pb2_grpc.py
```

For Junos OS releases prior to 18.4R1:

```
ubuntu-16:~ jet$ python -m grpc_tools.protoc -I./proto -python_out=. -grpc_python_out=.
proto/management_service.proto
ubuntu-16:~ jet$ python -m grpc_tools.protoc -I./proto -python_out=. -grpc_python_out=.
proto/authentication_service.proto
$ ls -lrt

drwxr-xr-x 1 vagrant vagrant   578 Sep  4 18:17 proto
-rw-r-r- 1 vagrant vagrant 47038 Sep  4 18:18 management_service_pb2.py
-rw-r-r- 1 vagrant vagrant  4418 Sep  4 18:18 management_service_pb2_grpc.py
-rw-r-r- 1 vagrant vagrant  4615 Sep  4 18:18 authentication_service_pb2.py
-rw-r-r- 1 vagrant vagrant  1449 Sep  4 18:18 authentication_service_pb2_grpc.py
```

WHAT'S NEXT

> For details on how to generate code from an IDL file in the language of your choice, see https://www.grpc.io/docs.

Develop and Package Your Application

1. Set up the JET VM.

If you are developing an application with a dependency on C or C++ modules or developing a signed application, you must use the JET VM for JET application development. See "Set Up the JET VM" on page 16 for instructions.

2. You are ready to develop the application using the language of your choice.

   You can write the application using a stub after a client side stub is generated. For more information on generating the gRPC client side stubs, writing the application using the stub, and generating code from an IDL file in the language of your choice, see https://www.grpc.io/docs/.

   > **NOTE:** The Python 2.7 end-of-life and end-of-support date is January 1, 2020. The official upgrade path for Python 2.7 is to Python 3. As support for Python 3 is added to devices running Junos OS for the different types of on-device scripts, we recommend that you migrate supported script types from Python 2 to Python 3, because support for Python 2.7 might be removed from devices running Junos OS in the future.

3. Package the application using JSON. See "Package JET Applications" on page 52 for more information.

SEE ALSO

> *Understanding Python Automation Scripts for Devices Running Junos OS*
>
> *IPv6 Support in Python Automation Scripts*

Prepare to Deploy Your Application

Run your application on an external server or directly from the JET VM. Before you deploy your application on an external server, you need to configure JET interaction with Junos OS.

Configure JET Interaction with Junos OS

To run an off-device application, you need to enable the **request-response** configuration on Junos OS. When using the request-response service, the client application issues a request and synchronously waits for the response from the Junos OS server. Use this section to configure the JET service process (jsd) for the request-response service to run in Secure Sockets Layer (SSL) mode. This provides increased security and enables SSL-based API connections.

NOTE: Currently, JET supports Transport Layer Security (TLS) version 1.2 for certificate exchange and supports multiple encryption algorithms, but does not support mutual authentication. This means that clients can authenticate the server, but the server can not authenticate clients using SSL/TLS certificates. For client authentication, use the LoginCheck() procedure from the authentication service API.

1. Enable jsd to use SSL by adding and configuring the certificate name locally. The certificate must be an RSA certificate. ECDSA and DSA SSL certificates are not supported.

   This method is same as other SSL-based services in Junos OS like xnm-ssl. Keep track of the certificate name entry you specify during certificate generation. You will use it for the **HOST_OVERRIDE** option in the example Python application in the next section. In this example, the certificate name is **router**.

   ```
   ubuntu-16:~ jet$ openssl genrsa -aes256 -out router.key.orig 2048
   ubuntu-16:~ jet$ openssl req -new -key router.key.orig -out router.csr
   ubuntu-16:~ jet$ openssl rsa -in router.key.orig -out router.key
   ubuntu-16:~ jet$ openssl x509 -req -days 365 -in router.csr -signkey router.key
   -out router.crt
   ubuntu-16:~ jet$ cat router.crt router.key > router.pem
   ```

   NOTE: If a certificate is updated with the same identifier, the changes will not be reflected for jsd. You need to either configure the certificate with a new identifier in the jsd hierarchy or perform a jsd restart to reflect the changes made.

2. Copy the SSL certificate **.pem** file to the device.

   ```
   % scp router.pem device-name:/var/tmp
   ```

3. Load the certificate into the keychain on the Junos OS device. For example, if the local name of the SSL certificate is **sslcert**:

   ```
   [edit]
   user@device# set security certificates local sslcert load-key-file
   /var/tmp/router.pem
   ```

4. Enable support for SSL on the gRPC endpoint on the default port of tcp/51051.

```
[edit]
user@device# set system services extension-service request-response grpc ssl
```

5. Specify the maximum number of simultaneous connections for request-response that can be attached to jsd. The higher the number, the higher the impact on the client's performance. The highest maximum number that can be configured is eight.

```
[edit]
user@device# set system services extension-service request-response grpc
max-connections 8
```

You have configured jsd for request-response service to run in SSL mode. You are ready to deploy your JET off-device application.

Example: Python JET Application

IN THIS SECTION

Use this example to develop an off-device JET application written in Python. You can follow the same guidance for other languages that are supported by gRPC. This Python JET application runs the command **get-system-uptime-information** in XML format.

In this example, the **HOST_OVERRIDE** option uses the certificate name that you specified during the certificate generation. See "Prepare to Deploy Your Application" on page 26.

> **NOTE:** Juniper Networks supports both of the following forms for denoting XML opening and closing tags: **<xml-tag/>** and **<xml-tag></xml-tag>**.

Junos OS Release 18.4R1 and Later

Use the example Python application shown in this section as a guide if you are using Junos OS Release 18.4R1 or later.

If you are writing your application using Python 3, include the PASS keyword in the Exception block of the script.

```
    except Exception as tx:
        pass
```

```
#!/usr/bin/env python

# A simple Python client to run XML OP command 'get-system-uptime-information'

# Environment
# Python 2.7.12
# grpcio (1.12.0)
# grpcio-tools (1.12.0)

# Following files should be available in current working directory
# jnx_authentication_service_pb2_grpc.py
# jnx_authentication_service_pb2.py
# jnx_management_service_pb2_grpc.py
# jnx_management_service_pb2.py

import argparse
import grpc
import os
import stat

import jnx_authentication_service_pb2
import jnx_authentication_service_pb2_grpc
import jnx_management_service_pb2
import jnx_management_service_pb2_grpc
import jnx_common_base_types_pb2


_HOST_OVERRIDE = 'router'

def Main():
    try:
        parser = argparse.ArgumentParser()
```

```
        parser.add_argument('-d','--device', help='Input hostname',
            required=True)
        parser.add_argument('-t','--timeout', help='Input time_out value',
            required=True,type=int)
        parser.add_argument('-u', '--user', help='Input username',
            required=True)
        parser.add_argument('-pw', '--password', help='Input password',
            required=True)

        args = parser.parse_args()

        #Establish grpc channel to jet router
        creds = grpc.ssl_channel_credentials(open('/tmp/router.pem').read(),
                                             None, None)
        channel = grpc.secure_channel(args.device + ":32767", creds,
            options=(('grpc.ssl_target_name_override', _HOST_OVERRIDE,),))

        #create stub for authentication services
        stub = jnx_authentication_service_pb2_grpc.AuthenticationStub(channel)
        #Authenticate
        login_request = jnx_authentication_service_pb2.LoginRequest(
            username=args.user, password=args.password, client_id="SampleApp")
        login_response = stub.Login(login_request, args.timeout)
        #Check if authentication is successful
        if login_response.status.code == jnx_common_base_types_pb2.SUCCESS:
            print "[INFO] Connected to gRPC Server"
        else:
            print "[ERROR] gRPC Server Connection failed:"
            print login_response.status.message

        #Create stub for management services
        stub = jnx_management_service_pb2_grpc.ManagementStub(channel)
        print "[INFO] Connected to management service"
        for i in range(1):
            #Provide API request details
            op_xml_command =
"<get-system-uptime-information></get-system-uptime-information>"
            op = jnx_management_service_pb2.OpCommandGetRequest(
                xml_command=op_xml_command, out_format=2)
            # Invoke API
            op_response = stub.OpCommandGet(op, args.timeout)
            # Check API response like status and output
            for resp in op_response:
```

```
                if resp.status.code == jnx_common_base_types_pb2.SUCCESS:
                    print "[INFO] Invoked OpCommandGetRequest succeeded"
                    print "[INFO] Return output in CLI format = "
                    print resp.data
                else:
                    print "[ERROR] Invoked OpCommandGetRequest failed"
                    print "[ERROR] " + resp.status.message


    except Exception as ex:
        print ex


if __name__ == '__main__':
    Main()
```

```
ubuntu-16:~ jet$ python mgd_api_new_doc_example_ssl.py -d JUNOS_DEVICE -t TIMEOUT
-u USER -pw PASSWORD

[INFO] Connected to gRPC Server
[INFO] Connected to management service
[INFO] Invoked OpCommandGetRequest succeeded
[INFO] Return output in CLI format =

Current time: 2018-11-08 09:36:40 PST
Time Source:  NTP CLOCK
System booted: 2018-10-09 17:02:56 PDT (4w1d 17:33 ago)
Protocols started: 2018-10-09 17:05:09 PDT (4w1d 17:31 ago)
Last configured: 2018-11-08 09:30:28 PST (00:06:12 ago) by root
 9:36AM up 29 days, 17:34, 2 users, load averages: 1.05, 0.77, 0.57
```

Before Junos OS Release 18.4R1

Use the example Python application in this section as a guide if you are using Junos OS releases prior to 18.4R1.

```
#!/usr/bin/env python

# A simple Python client to run XML OP command 'get-system-uptime-information'

# Environment
# Python 2.7.12
```

```
# grpcio (1.12.0)
# grpcio-tools (1.12.0)

# Following files should be available in current working directory
# authentication_service_pb2_grpc.py
# authentication_service_pb2.py
# management_service_pb2_grpc.py
# management_service_pb2.py

import argparse
import grpc

import authentication_service_pb2
import authentication_service_pb2_grpc
import management_service_pb2
import management_service_pb2_grpc

_HOST_OVERRIDE = 'router'

def Main():
    try:
        parser = argparse.ArgumentParser()

        parser.add_argument('-d','-device', help='Input hostname',
            required=True)
        parser.add_argument('-t','-timeout', help='Input time_out value',
            required=True,type=int)
        parser.add_argument('-u', '-user', help='Input username',
            required=True)
        parser.add_argument('-pw', '-password', help='Input password',
            required=True)

        args = parser.parse_args()

        #Establish grpc channel to jet router
        creds = grpc.ssl_channel_credentials(open('/tmp/router.pem').read(),
                                             None, None)
        channel = grpc.secure_channel(args.device + ":51051", creds,
            options=(('grpc.ssl_target_name_override', _HOST_OVERRIDE,),))

        #create stub for authentication services
        stub = authentication_service_pb2_grpc.LoginStub(channel)
        #Authenticate
        login_request = authentication_service_pb2.LoginRequest(
```

```
            user_name=args.user, password=args.password, client_id="SampleApp")
        login_response = stub.LoginCheck(login_request, args.timeout)

        #Check if authentication is successful
        if login_response.result == True:
            print "[INFO] Connected to gRPC Server:"
            print login_response.result
        else:
            print "[ERROR] gRPC Server Connection failed!!!"
            print login_response.result

        #Create stub for management services
        stub = management_service_pb2_grpc.ManagementRpcApiStub(channel)
        print "[INFO] Connected to JSD and created handle to mgd services"

        for i in range(1):
            #Provide API request details
            op_xml_command = "<get-system-uptime-information>" \
            "</get-system-uptime-information>"
            op = management_service_pb2.ExecuteOpCommandRequest(
                xml_command=op_xml_command, out_format=2, request_id=1000)
            # Invoke API
            result = stub.ExecuteOpCommand(op, 100)
            # Check API response like status and output
            for i in result:
                print "[INFO] Invoked ExecuteOpCommand API return code = "
                print i.status
                print "[INFO] Return output in CLI format = "
                print i.data
    except Exception as ex:
        print ex

if __name__ == '__main__':
    Main()
```

```
ubuntu-16:~ jet$ python mgd_api_doc_example_ssl.py -d JUNOS_DEVICE -t TIMEOUT_VAL
-u USER -pw PASSWORD

[INFO] Connected to gRPC Server:
True
[INFO] Connected to JSD and created handle to mgd services
[INFO] Invoked ExecuteOpCommand API return code =
0
[INFO] Return output in CLI format =
```

```
Current time: 2018-09-04 11:24:36 PDT
Time Source:  NTP CLOCK
System booted: 2018-08-31 10:58:22 PDT (4d 00:26 ago)
Protocols started: 2018-08-31 11:00:52 PDT (4d 00:23 ago)
Last configured: 2018-08-31 14:21:32 PDT (3d 21:03 ago) by root
11:24AM  up 4 days, 26 mins, 0 users, load averages: 1.20, 1.27, 1.10
```

RELATED DOCUMENTATION

Debug JET Applications | 58

Develop On-Device JET Applications

IN THIS SECTION

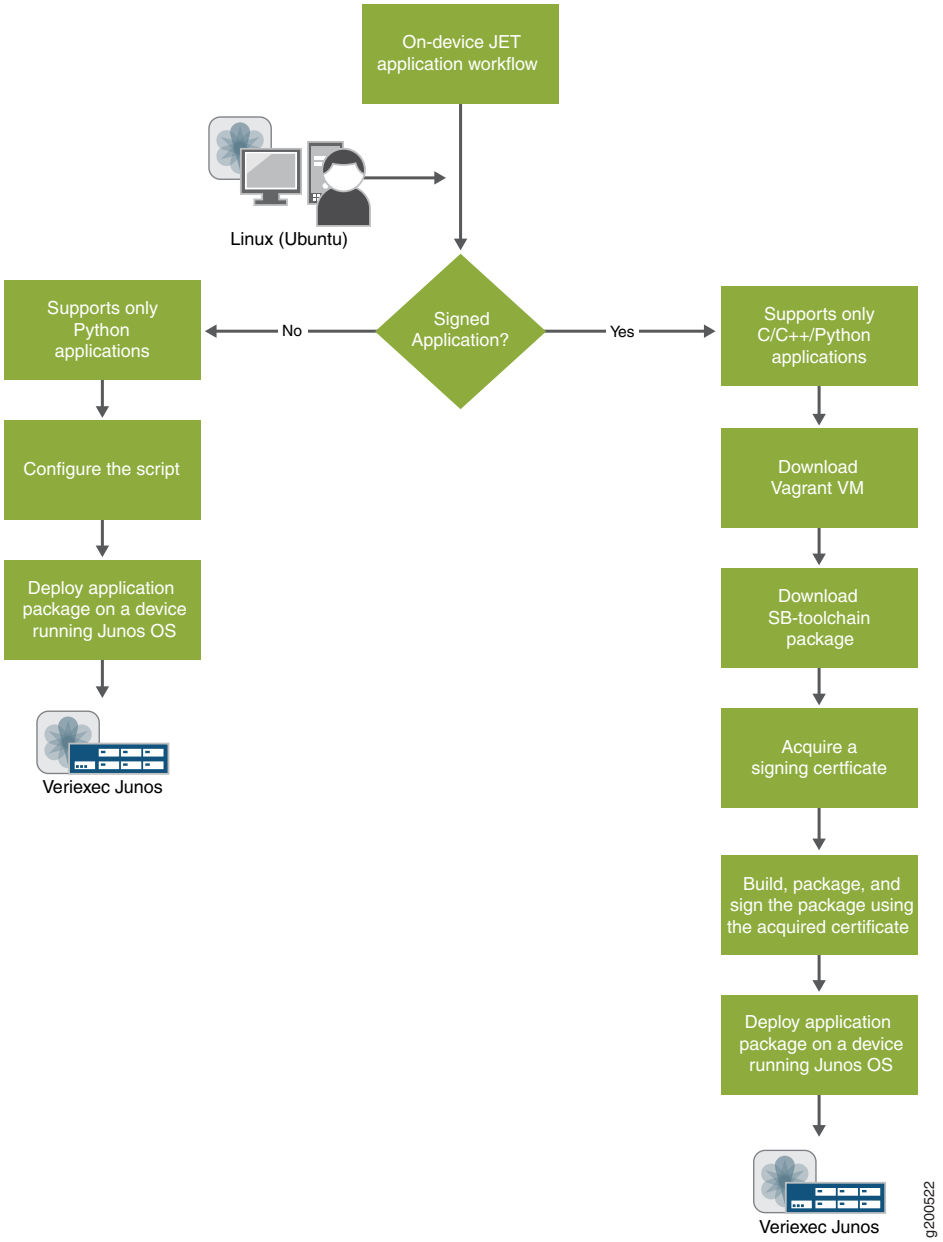
Overview

JET applications written in C, C++, and Python languages can run on-device. You can develop the applications in the downloaded JET VM and then deploy these applications on the device running Junos OS. You can sign on-device JET applications to show that they can be trusted.

NOTE: The Python 2.7 end-of-life and end-of-support date is January 1, 2020. The official upgrade path for Python 2.7 is to Python 3. As support for Python 3 is added to devices running Junos OS for the different types of on-device scripts, we recommend that you migrate supported script types from Python 2 to Python 3, because support for Python 2.7 might be removed from devices running Junos OS in the future.

Figure 3 on page 36 shows the application development workflow for unsigned and signed on-device JET applications.

Figure 3: On-Device JET Application Workflow



SEE ALSO

Understanding Python Automation Scripts for Devices Running Junos OS

IPv6 Support in Python Automation Scripts

Develop Unsigned JET Applications

Unsigned JET applications can only be written in Python.

To develop an unsigned JET application:

1. (Optional) Download and set up the JET VM. See "Set Up the JET VM" on page 16.
2. Develop your application in Python.
3. Package your application. See "Package JET Applications" on page 52.
4. Configure the **language** statement on the Junos OS device. For example, to use Python 3 to run a JET script that supports Python 3:

   ```
   [edit]
   user@device# set system scripts language python3
   ```

   See *Understanding Python Automation Scripts for Devices Running Junos OS* for more information.
5. Run the application on a device running on Junos OS.

Develop Signed JET Applications

You can develop signed applications in C, C++, or Python.

To develop a signed application:

1. Download the packages you need and set up the JET VM. See "Set Up the JET VM" on page 16.
2. Request a signing certificate. See "Request a Signing Certificate" on page 38.
3. Develop the application.
4. Configure the license if your application is written in C or C++. This step is optional for applications written in Python. See *Configuring the JET Application and its License on a Device Running Junos OS* for details.
5. Build the package and sign the package using the acquired certificate.
6. Deploy the application on a device running Junos OS.

Request a Signing Certificate

To develop and distribute JET applications, you must install a package signing certificate on the JET VM by executing the certificate request script. This script assists you in creating a signing key and a certificate request for use with JET.

CAUTION: Never share the signing key with anyone, including Juniper Networks. The key enables anyone to sign applications that your router will trust. Therefore, treat the key with the same level of security as the root password for the routers. Once you obtain your signing key, save it in a file outside of the VM.

The certificate request script asks for the following information:

- City, state, and country.
- Your organization and unit.
- Provider prefix—This is the unique provider name assigned by Juniper to each JET partner.
- User string—This is an additional specification of your choosing. It could be a string specifying the development team or project name. The user string can consist of a lowercase letter followed by one or more lowercase letters or numbers (for example, teamjet2).
- Deployment scope—The deployment scope is the string assigned by Juniper to differentiate multiple certificates for the same JET partner. This defines the validity period for the generated certificate. The scope can be commercial or evaluation. If none is assigned to you, leave it empty.
- Index number—This number is known as a certificate generations number. It will be 1 for your initial certificate. When a certificate expires and when you request a new one, this number will be incremented.
- Email address—The email address for the certificate contact will be embedded into the certificate. We recommend using the email address of a department or unit in your organization. We recommend that you do not use a personal email address.

To create a signed application, request certificates and copy them as explained in the following procedure. This procedure is optional if you want to create an unsigned application.

To manually request a certificate:

1. Create the **/usr/local/junos-jet/certs** directory if it does not already exist on your device.

2. In a VM terminal, run the **jet-certificate-request** command.

3. The script leads you through a series of questions. Answer the questions and press Enter after each answer.

4. Based on your answers, the script generates two files in the **/usr/local/junos-jet/certs** directory: ***certname*_key.pem** and ***certname*_req.pem**. The *certname* is the name of the certificate.

   The certificate name must follow the format ORGANIZATION-USER-TYPE-NUMBER:

   ```
   ERROR: CN has invalid format; regex:
   ^([a-z0-9]+)-([a-z0-9]+)-(commercial|private)-([1-9][0-9]*)$
          Expected format: ORGANIZATION-USER-TYPE-NUMBER
              organization: [a-z0-9]+
                  Must be "juniper" for type is "private"
              user:   [a-z0-9]+
              type:   commercial|private
              number: [1-9][0-9]*
   ```

   The type must be commercial or private. Private certificates are only assigned when the organization is Juniper.

5. Save the ***certname*_key.pem** file outside the VM.

   This is your signing key. Ensure that no one outside of your development organization has access to it. Never share the signing key with anyone, including Juniper Networks.

   The key enables anyone to sign applications that your router will trust. Therefore, treat the key with the same level of security as the root password for the routers.

6. Send the ***certname*_req.pem** file to JET Certificate Processing at jet-cert@juniper.net. This file contains your certificate request. JET Certificate Processing will immediately send your certificate to you.

7. When you receive your certificate, save it as *certname* and copy it to the **/usr/local/junos-jet/certs** directory.

8. Verify the certificate and the signing key are available in the **/usr/local/junos-jet/certs** directory.

Example: Develop a Signed C Package

After you have set up the JET VM and acquired a signing certification, you are ready to create the development sandbox in the VM and start developing your signed, on-device application. Use this example to create C applications echoclient and echoserver.

1. Check out the sandbox. A sandbox is a build tree with a little environment file called **.sandbox-env** at the top that is used by a wrapper script **mk** to ensure the build environment is properly conditioned.

```
vagrant@jet-vm:~$ mksb -n capp echoclient /home/vagrant/capp_server/src/echoclient.json
```

> **NOTE:** The echo client is a demo application. In the **bin/** directory, all the necessary configuration and build related files are available within the sandbox along with source file for the echo client.

2. Build an echo client package.

```
vagrant@jet-vm:/home/vagrant/capp/src$ mk-i386,bsdx echoclient
```

> **NOTE:** Starting in Junos OS Release 20.2R1, if you will be running your JET application on a ACX710 device, you can use the Clang toolchain for ARM-based compilation of JET applications written in C, Python, or Ruby. Use the command **mk-arm,bsdx** instead of **mk-i386,bsdx** to use the Clang toolchain to compile your application.

3. Copy the echo client package onto the device running on Junos OS.

```
vagrant@jet-vm:/home/vagrant/capp/src$ scp
/home/vagrant/capp/junos-jet-sb-obj//ship/echoclient-x86-32-20180829.065039_vagrant.tgz
root@device:/var/tmp
```

4. Enter configuration mode on the Junos OS device.

```
root@device:~ # cli
```

5. Install the echo client package.

```
root@device> request system software add /var/tmp/echoclient-x86-32-20180829.065039_vagrant.tgz
```

```
Verified echoclient-x86-32-20180829.065039_vagrant signed by
junosmanageability-dev-beta-1 method RSA2048+SHA1
```

Confirm it was installed successfully.

```
root@device> show version

Hostname: device
Model: mx480
Junos: 18.4-20180627_dev_common.1
JUNOS OS Kernel 32-bit  [20180621.191151_fbsd-builder_stable_11]
…
…
JET app echoclient [20180829.065039_vagrant]
```

6. Check out the echo server sandbox.

```
vagrant@jet-vm:/home/vagrant$ mksb -n capp_server echoserver
/home/vagrant/capp_server/src/echoclient.json
```

7. Build the echo server package.

```
vagrant@jet-vm:/home/vagrant/capp_server/src$ mk-i386,bsdx echoserver
```

8. Copy the echo server package to the Junos VM.

```
vagrant@jet-vm:/home/vagrant/capp_server/src$ scp
/home/vagrant/capp_server/junos-jet-sb-obj//ship/echoserver-x86-32-20180829.065703_vagrant.tgz
root@device:/var/tmp/
```

9. Add the echo server package to the Junos OS device.

```
root@device> request system software add /var/tmp/echoserver-x86-32-20180829.065703_vagrant.tgz

Verified echoserver-x86-32-20180829.065703_vagrant signed by
junosmanageability-dev-beta-1 method RSA2048+SHA1
```

10. Check that the packages were added successfully.

```
root@device> show version

Hostname: device
Model: mx480
Junos: 18.4-20180627_dev_common.1
JUNOS OS Kernel 32-bit  [20180621.191151_fbsd-builder_stable_11]
JUNOS OS libs [20180621.191151_fbsd-builder_stable_11]
……….
……
JET app echoserver [20180829.065703_vagrant]
JET app echoclient [20180829.065039_vagrant]
```

11. Configure the echo server's provider's ID, license type, and deployment scope on the Junos OS device. Use the same provider license that you used to package it.

```
root@device# set system extensions providers 12345 license-type juniper
deployment-scope commercial
```

For more information, see *Configuring the JET Application and its License on a Device Running Junos OS*.

12. Configure the echo server on the Junos OS device.

```
root@device# set system extensions extension-service application file echoserverd
[edit]
root@device# commit
commit complete
[edit]
root@device# exit
```

```
root@device> request extension-service start echoserverd

Extension-service application 'echoserverd' started with pid: 12345
```

13. Configure the echo client's provider's ID, license type, and deployment scope on the Junos OS device. Use the same provider license that you used to package it.

```
root@device# set system extensions providers 56789 license-type juniper
deployment-scope commercial
```

14. Configure the echo client application on the Junos OS device.

```
root@device# set system extensions extension-service application file echoclientd arguments "127.0.0.1
 Testmessage"
[edit]
root@device# commit
commit complete
[edit]
root@device# exit
```

15. Run the echo client application.

```
root@device> request extension-service start echoclientd

Extension-service application 'echoclientd' started with pid: 56789
-- server reply:Testmessage
-- Testmessage
```

Example: Develop a Signed Python Package Without C Dependencies

After you have set up the JET VM and acquired a signing certification, you are ready to create the development sandbox in the VM and start developing your signed, on-device application. Use this example to develop a signed Python package without C dependencies.

1. In the VM, go to the **/home/vagrant** directory.

2. Create a sandbox by using the **mksb** command, where **SamplePyApp** is the name of the sandbox. A sandbox is a build tree with a little environment file called **.sandbox-env** at the top that is used by a wrapper script **mk** to ensure the build environment is properly conditioned.

```
vagrant@jet-vm:~$ mksb -n SamplePyApp
% mksb -n SamplePyApp
```

3. Create subdirectories in the sandbox.

   First, use the **workon** command to go into your sandbox. The **workon** command takes you directly to the **$SB/src** directory and sets the sandbox correctly.

```
vagrant@jet-vm:/home/vagrant$ workon SamplePyApp
```

Alternatively, you can **cd** to the **src** directory of your sandbox.

Next, create subdirectories for application code in **$SB/src/python**, **$SB/src/lib**, or **$SB/src/bin**, based on whether you need Python, library, or bin (executable) files.

```
vagrant@jet-vm:/home/vagrant/pyapp/src/python$ mkdir SamplePyApp
```

4. Develop the code.

```
vagrant@jet-vm:$ /home/vagrant/pyapp/src/python/SamplePyApp$ ls cmdline_args.py .
```

If you are writing your application using Python 3, include the PASS keyword in the Exception block of the script.

```
    except Exception as tx:
        pass
```

5. Write an application JSON file to package the application.

```
SamplePyApp.json
{
"app-name" : "SamplePyApp",
"app-path" : "python/SamplePyApp",
"language" : "python",
"main-scripts" : ["cmdline_args.py"],
"app-type" : "standalone",
"sign" : "yes",
"os-type" : "bsd11",
"target-arch" : "i386",
"description" : "Simple Python App"
}
```

See"Package JET Applications" on page 52 for more information.

6. Add the path to **jet-pkg-gen.py** to your **.bashrc** file.

```
vagrant@jet-vm:/home/vagrant$ echo
'PATH=$PATH:/usr/local/junos-jet/18.2R1.9/junos-jet-sb/src/junos/host-utils/scripts' >> ~/.bashrc
vagrant@jet-vm:/home/vagrant/pyapp/src$ source ~/.bashrc
```

7. Autogenerate the appropriate makefiles by running the **jet-pkg-gen.py** command. The **jet-pkg-gen.py** command takes two options:

   - The **-i** option is followed by the path and filename of the JSON file.
   - The **-p** option is followed by the path to the **src** directory of the sandbox.

   For example, if the sandbox name is SamplePyApp:

   ```
   vagrant@jet-vm:~/SamplePyApp/src$ jet-pkg-gen.py -I /home/vagrant/pyapp/src/SamplePyApp.json
   -p /home/vagrant/pyapp/src
   ```

   NOTE: The autogenerated application makefile will be correct in most cases. If there are any external library dependencies, adjust the makefile accordingly.

8. Build the entire package.

   First, return to the **src** directory (**$SB/src**). Next, run the **mk-i386** ***SamplePyApp*** command, where ***SamplePyApp*** is the **"app-name"** from the JSON file in Step 5.

   ```
   vagrant@jet-vm:/home/vagrant/pyapp/src$ mk-i386,bsdx SamplePyApp
   ```

   NOTE: Starting in Junos OS Release 20.2R1, if you will be running your JET application on a ACX710 device, you can use the Clang toolchain for ARM-based compilation of JET applications written in C, Python, or Ruby. Use the command **mk-arm,bsdx** instead of **mk-i386,bsdx** to use the Clang toolchain to compile your application.

9. Copy the package onto a device running Junos OS.

   ```
   vagrant@jet-vm:/home/vagrant/pyapp/src$
   scp../junos-jet-sb-obj/ship/SamplePyApp-x86-32-20180828.231545_vagrant.tgz root@device:/var/tmp/
   ```

Now configure the Junos OS device and install the package.

1. Enter configuration mode.

```
root@device> configure
Entering configuration mode
[edit]
root@device#
```

2. Configure the application's provider's ID, license type, and deployment scope on the Junos OS device, if necessary. Use the same provider license that you used to package it.

```
root@device# set system extensions providers 12345 license-type juniper
deployment-scope commercial
```

   For more information, see *Configuring the JET Application and its License on a Device Running Junos OS*.

3. Exit to operational mode and install the copied package on the Junos OS device.

```
root@device# exit
root@device> request system software add
/var/tmp/SamplePyApp-x86-32-20180828.231545_vagrant.tgz
```

   NOTE: This step will fail if **providers** is not configured.

4. Verify the package was installed successfully.

```
root@device> show version

Hostname: device
Model: mx480
…
…
JET app SamplePyApp [20180828.231545_vagrant]
```

5. Enter configuration mode.

```
root@device> configure
Entering configuration mode
[edit]
root@device#
```

6. Configure the command-line arguments through the Junos OS CLI. If a Python JET script is available in the **/var/db/scripts/jet/** directory on a device running Junos OS, you can configure command-line arguments for the file and supply the arguments from the Junos CLI.

   Here are the arguments in the application.

```
import argparse

def main():
    parser = argparse.ArgumentParser(description='This is a demo script.')

    parser.add_argument('-arg1', required=True)
    parser.add_argument('-arg2', required=True)

    args = parser.parse_args()

    print args.arg1
    print args.arg2

if __name__ == '__main__':
    main()
```

   Configure the command-line arguments in the CLI. In this example, the script filename is **cmdline_args.py**.

```
root@device# set system extensions extension-service application file cmdline_args.py arguments "-arg1
jet -arg2 application"
```

7. Commit the configuration and exit to operational mode.

```
root@device# commit
root@device# exit
```

8. Run the application.

```
root@device> request extension-service start cmdline_args.py

Extension-service application 'cmdline_args.py' started with PID: 12345
jet
application
```

Example: Develop a Signed Python Package With C Dependencies

After you have set up the JET VM and acquired a signing certification, you are ready to create the development sandbox in the VM and start developing your signed, on-device application. Use this example to develop a signed Python package with C dependencies.

1. Check out the sandbox.

```
vagrant@jet-vm:/vagrant/jet-trial-apps/signed-python-with-c-dep$ mksb -n PyAppC
```

2. Create an application directory in the Python subdirectory.

```
vagrant@jet-vm:/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/src/python$ mkdir pyappc
```

3. A bitarray is an example of a relatively simple Python module with a C dependency. Download and extract the bitarray from https://pypi.org/project/bitarray/ into the Python application directory.

```
vagrant@jet-vm:/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/src/python/pyappc$ ls
bitarray

_bitarray.c  __init__.py
```

This is an example of a simple Python application that uses a bitarray module:

```
vagrant@jet-vm:/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/src/python/pyappc$
 cat bitarray_app.py
from bitarray import bitarray

a = bitarray()
a.append(True)
```

```
a.extend([False, True, False])
print a
```

If you are writing your application using Python 3, include the PASS keyword in the Exception block of the script.

```
    except Exception as tx:
        pass
```

4. Create the JSON configuration file that references the external source files. See "Package JET Applications" on page 52 for more information.

```
vagrant@jet-vm:/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/src$ cat pyappc.json

{
 "app-name" : "PyAppC",
 "app-path" : "python/pyappc",
 "language" : "python",
 "main-scripts" : ["bitarray_app.py"],
 "app-type" : "standalone",
 "sign" : "yes",
 "os-type" : "bsd11",
 "target-arch" : "i386",
 "description" : "Simple Python App with C dependencies",

 "srcs" : {
            "python/pyappc/bitarray" : ["__init__.py"]
 },
 "extn-srcs" : {
            "python/pyappc/bitarray" : ["_bitarray.c"]
 }
}

vagrant@jet-vm:/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/src$ jet-pkg-gen.py -l
/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/src/pyappc.json -p
/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/src
```

5. Run the following command to create the necessary makefiles and the manifest file that locates the files on the Junos OS device when the package is installed.

```
/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/src/pkgs/PyAppC/contents.manifest

vagrant@jet-vm:/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/src$ cat
./pkgs/PyAppC/contents.manifest.orig

/set package_id=31 role=Provider_Daemon
%TOPDIR%/python/pyappc/bitarray_app.py store=%INSTALLDIR%/bitarray_app.py mode=555
 program_id=1
%TOPDIR%/python/pyappc/bitarray/__init__.py
store=%PYTHON_MOD_INSTALLDIR%/PyAppC/bitarray/__init__.py mode=555 program_id=1
%TOPDIR%/python/pyappc/_bitarray.so
store=%PYTHON_MOD_INSTALLDIR%/PyAppC/_bitarray.so mode=555 program_id=1
```

6. To locate the bitarray module on a Junos OS device, add the **/PyAppC/** path for the **__init__.py** file and the **bitarray/** directory path for the **_bitarray.so** file.

```
vagrant@jet-vm:/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/src$ cat
./pkgs/PyAppC/contents.manifest

/set package_id=31 role=Provider_Daemon
%TOPDIR%/python/pyappc/bitarray_app.py store=%INSTALLDIR%/bitarray_app.py mode=555
 program_id=1
%TOPDIR%/python/pyappc/bitarray/__init__.py
store=%PYTHON_MOD_INSTALLDIR%/bitarray/__init__.py mode=555 program_id=1
%TOPDIR%/python/pyappc/_bitarray.so
store=%PYTHON_MOD_INSTALLDIR%/bitarray/_bitarray.so mode=555 program_id=1
```

7. Build and package the application.

```
vagrant@jet-vm:/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/src$ mk-i386,bsdx PyAppC
```

> **NOTE:** Starting in Junos OS Release 20.2R1, if you will be running your JET application on a ACX710 device, you can use the Clang toolchain for ARM-based compilation of JET applications written in C, Python, or Ruby. Use the command **mk-arm,bsdx** instead of **mk-i386,bsdx** to use the Clang toolchain to compile your application.

8. Copy the built package onto the device running Junos OS.

```
vagrant@jet-vm:/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/src$ scp
/vagrant/jet-trial-apps/signed-python-with-c-dep/PyAppC/junos-jet-sb-obj/ship/
PyAppC-x86-32-20180829.211252_vagrant.tgz root@device:/var/tmp/
```

9. Configure the application's provider's ID, license type, and deployment scope on the Junos OS device, if necessary. Use the same provider license that you used to package it.

```
root@device# set system extensions providers 12345 license-type juniper deployment-scope commercial
```

For more information, see *Configuring the JET Application and its License on a Device Running Junos OS*.

10. Install the package on the device running Junos OS.

```
root@device> request system software add PyAppC-x86-32-20180830.031354_vagrant.tgz
```

Once the package is installed successfully, the dependent Python module and the C shared library are installed on the device in the standard Python module path as specified in the manifest file.

```
root@device:/opt/lib/python2.7/site-packages # ls bitarray/__init__.py  _bitarray.so
```

11. Add the application in configuration mode.

```
root@device# set system extensions extension-service application file bitarray_app.py
[edit]
root@device# commit
commit complete
```

12. Run the application

```
root@device> request extension-service start bitarray_app.py

Extension-service application 'bitarray_app.py' started with PID: 12345
bitarray('1010')
```

Package JET Applications

IN THIS SECTION

After application development is complete, write the JavaScript Object Notation (JSON) file describing the content to build and package the application before deploying it on the device. JSON is a lightweight data-interchange format. It is easy for humans to read and write, and also easy for machines to parse and generate. For more details, see https://www.json.org.

JSON files consist of a collection of attributes are included inside a set of curly braces. Attributes use two structures:

- A collection of key-value pairs.
- An ordered list of values.

Read further to learn about each of the attributes contained in the JSON format for application packaging.

Main Section Attributes

IN THIS SECTION

The top block of the JSON file is the main section of the file. It consists of mandatory and optional attributes.

Mandatory Attributes

Table 4 on page 53 describes the mandatory attributes that all JSON files for application packaging must have in the main section. The following is an example of a simple application JSON file containing only the mandatory attributes:

```
{
  "app-name": "testcapp",
  "app-path": "bin/test-c-app",
  "language": "c",
  "app-type": "standalone",
  "sign": "yes",
  "os-type": "bsd10",
  "target-arch": "i386",
  "description": "C Test Application",
  "srcs": {
    "bin/test-c-app": ["test_app.c"]
  }
}
```

Table 4: Mandatory Attributes in the JSON File Main Section

Attribute	Description	Example Values
"app-name"	Specify the name of the application.	**"sample_pyapp"**
"app-path"	Specify the path to the application's implementation directory. All paths should be relative to sandbox **src**.	**"python/sample_pyapp"**
"language"	Specify the language used for developing the application.	**"python"**, **"c"**, **"c++"**
"main-scripts"	This is a list attributes. Specify the filename or filenames of the main script or scripts that run on the device (do not specify the module here). The main script files will be deployed under the **/var/db/scripts/jet** path on the device.	**["foo.py", "bar.py"]**
"app-type"	Specify whether an application is to be a standalone program or a daemon.	**"standalone"** or **"daemon"**
"sign"	Indicate whether the application is to be signed or unsigned.	**"yes"** or **"no"**
"os-type"	Specify whether the application is to be deployed on legacy Junos OS (bsd6) or Junos OS with upgraded FreeBSD (bsd10).	**"bsd6"**, **bsd10**, or **"bsd11"**

Table 4: Mandatory Attributes in the JSON File Main Section *(continued)*

Attribute	Description	Example Values
"target-arch"	Specify the target architecture on which the application is to be deployed.	**"i386"**, **"powerpc"**, **"octeon"**, **"xlr"**, or **"arm"**
"description"	Write a brief (one-line) description about the application. This will be displayed in the **show version** operational command output.	**"Simple Python test app"**

Optional Attributes

Table 5 on page 54 describes the optional attributes you can include in the main section of the JSON file for application packaging. The following is an example main section with mandatory and optional attributes:

```
{
  "app-name": "sample_pyapp",
  "app-path": "python/sample_pyapp",
  "language": "python",
  "main-scripts": ["foo.py", "bar.py"],
  "app-type": "standalone",
  "sign": "no",
  "os-type": "bsd6",
  "target-arch": "i386",
  "description": "Simple Python test app",
  "c-compiler-flags": "-DFOO  -DBAR",
  "c++-compiler-flags": "-DAPP_CHECK  –DSOMETHING_ON",
  "linker-flags": "-lstdc++ -lfoo"  }
```

Table 5: Optional Attributes in the JSON File Main Section

Attribute	Description	Example Values
"c-compiler-flags"	Specify the list of C compiler flags, if any. Compilation flags can be defined for the main section, dependent libraries (dep-libs), or dependent Python modules (dep-py-modules).	***"flag1 flag2 flag3"***

Table 5: Optional Attributes in the JSON File Main Section *(continued)*

Attribute	Description	Example Values
"c++-compiler-flags"	Specify the list of C++ compiler flags, if any. Compilation flags can be defined for the main section, dependent libraries (dep-libs), or dependent Python modules (dep-py-modules).	**"*flag1 flag2 flag3*"**
"linker-flags"	Specify the list of linker flags, if any. Use these flags to specify additional libraries to link to or additional link-specific flags that are required during linking, You can define linker-specific flags either in the main section or in the dep-py-modules section.	**"*flag1 flag2 flag3*"**

Source Attributes

Table 6 on page 56 shows two source attributes you can use to specify source files for the application package. The following is an example Python application with additional module files to be deployed, along with the main script file:

```
{
  "app-name": "sample_pyapp",
  "app-path": "python/sample_pyapp",
  "language": "python",
  "main-scripts": ["foo.py", "bar.py"],
  "app-type": "standalone",
  "sign": "no",
  "os-type": "bsd6",
  "target-arch": "i386",
  "description": "Simple Python test app",

  "srcs": {
    "python/sample_pyapp": ["a.py", "b.py"],
    "python/sample_pyapp/temp": ["temp1.py", "temp2.py"]
  },

  "extn-srcs": {
    "python/sample_pyapp": ["foo.c", "bar.c"],
    "python/sample_pyapp/temp": ["1.cpp", "2.cpp"]
  }
}
```

Table 6: Source Attributes You Can Use in a JSON File

Attribute	Description	Example Values

"srcs"	Specify the list of additional source files. For Python applications, these source files are the additional module files. For C or C++ applications, these source files are the source files to be compiled to generate **lib/binary**. Each entry should be a key-value pair, where the key is the path of the source files and the value is an array of source filenames.	"srcs": {
		"python/sample_pyapp": ["a.py", "b.py"],
		"python/sample_pyapp/temp": ["temp1.py",
		"temp2.py"]
		}

"extn-srcs"	This section is applicable only for Python. Specify the list of C or C++ module files to be compiled. Each entry should be a key-value pair, where the key is the path of the source files and the value is an array of source filenames.	"extn-srcs": {
		"python/sample_pyapp": ["foo.c", "bar.c"],
		"python/sample_pyapp/temp": ["1.cpp", "2.cpp"]
		}

Dependent Libraries

You must compile any dependent libraries available in the dependent libraries (**dep-libs**) section. The library generated from this JSON code is packaged with the application. The **dep-libs** section is an array of multiple library dependencies, each composed of the following key-name pairs:

- **"lib-name"** is the name of the library.
- **"lib-path"** is the path of the library source code in the development sandbox.
- **"srcs"** is a key-value pair in which the path is the key and its value is a list of source files.

The following is an example of a **dep-libs** attribute:

```
"dep-libs": [
  {
    "lib-name": "xyz",
    "lib-path": "lib/xyz",
```

```
        "srcs": {
          "lib/xyz": ["foo.c", "bar.c"]
        }
      }
      ]
```

Dependent Python Modules

The dependent Python modules (**dep-py-modules**) attribute is used only for Python applications. This attribute contains any dependent Python modules that need to be compiled and packaged with the application. The **dep-py-modules** attribute is an array in which you can specify multiple Python module dependencies. Each dependency is composed of the following objects:

- **"py-module-name"** is the name of the Python module.
- **"py-module-path"** is the path of the Python module source code in the development sandbox.
- **"srcs"** is a key-value pair in which the path is the key and its value is a list of source files.
- **"extn-srcs"** is a key-value pair in which the path is the key and its value is a list of Python extension source files.

The following is an example of a **dep-py-modules** attribute:

```
"dep-py-modules": [
      {
        "py-module-name": "module_a",
        "py-module-path": "python/module_a",
        "srcs": {
          "python/module_a": ["foo.py", "bar.py"]
        },
        "extn-srcs": {
          "python/module_a": ["foo.c", "bar.c"],
          "python/module_a/sub_mod": ["lmn.cpp"]
        }
      }
      ]
```

RELATED DOCUMENTATION

Develop Off-Device JET Applications | 22
Develop On-Device JET Applications | 34

Debug JET Applications

IN THIS SECTION

Use this topic to debug JET applications.

Debugging Tips

- For debugging applications on a device running Junos OS, you can configure the trace file option with the **edit system services extension-service** **traceoptions** statement. You need to enable this statement on the Junos OS device before writing the sample applications.
- The Junos service process (jsd) is supported only on the Routing Engine running in master mode. It is not supported on the backup Routing Engine.
- To eliminate any firewall issues, use an on-device application to test.
- For notification applications, verify that your client IP source address (the address from which the connection is established) is added to the list of allowed clients in the jsd notification configuration.
- Ensure that the maximum number of notification connections does not exceed the number configured on the device. Use the following command to see the clients:

```
netstat -a | grep 1883
```

How to Invoke the Debugger During Install

For non-daemonized applications that run on the router, you can invoke the debugger at the same time that you install the application. To load your application along with the debugger:

1. Use the Junos OS CLI to invoke the debugger and install the application at the same time.

```
user@device> request extension-service start invoke-debugger cli application-name.py

Extension-service application 'application-name.py' started with pid: 12345
```

2. Enter **help** to display a list of the supported commands.

```
(Pdb) help

Documented commands (type help <topic>):
========================================
EOF    bt         cont      enable  jump  pp       run      unt
a      c          continue  exit    l     q        s        until
alias  cl         d         h       list  quit     step     up
args   clear      debug     help    n     r        tbreak   w
b      commands   disable   ignore  next  restart  u        whatis
break  condition  down      j       p     return   unalias  where
Miscellaneous help topics:
==========================
exec  pdb
Undocumented commands:
======================
retval  rv
```

3. Use the debugger commands as needed by typing **help <topic>**.

Issue: Cannot Connect to jsd

Use this procedure if your application cannot connect to jsd.

1. Check whether jsd is up and running on the Junos OS device using the following command:

```
ps aux | grep jsd
```

2. If jsd is not up, restart jsd. Choose from the following options:
   - gracefully—Gracefully restart the process.
   - immediately—Immediately restart (SIGKILL) the process.
   - soft—Soft reset (SIGHUP) the process.
   - |—Pipe through a command.

```
user@device# restart jsd <gracefully | immediately | soft>
```

3. If jsd is up, verify the configuration is present on the device using the following command:

```
user@device# show system services extension-service
```

   You should see the configuration in the output. If you do not, redo the configuration.

4. If the configuration is present, verify jsd is listening on configured port 51051:

```
nestat -a | grep 51051
```

   You should see a matching entry.

5. If you do not see a matching entry, restart jsd.

```
user@device# restart jsd <gracefully | immediately | soft>
```

RELATED DOCUMENTATION

3

CHAPTER

Additional Resources

Additional Resources

- Getting Started with JET: Video Tutorials
- Expert Advice: Junos Extension Toolkit (JET)
- FAQ: Learning About JET Part 1—Python on Junos OS
- FAQ: Learning About JET Part 2—JavaScript Object Notation (JSON)
- FAQ: Learning About JET Part 3—JET APIs
- FAQ: Learning About JET Part 4—Fast Programmatic Configuration
- An Intro to Juniper's JET Automation framework and how to use it
- Junos OS Has the Toolkit to Make You an Automation Wizard
- The Modern, Autonomous Enterprise—Four Essential Network Solutions

4

CHAPTER

Configuration Statements

application (Extensions)

Syntax

```
application {
    file script-name {
        arguments arguments;
        checksum hash-algorithm hash-value;
        daemonize;
        refresh;
        refresh-from;
        respawn-on-normal-exit;
        routing-instance;
        source;
        traceoptions {
            file <filename> <files number> <match regex> <size size> <world-readable | no-world-readable>;
            flag flag;
            no-remote-trace;
        }
        username username;
    }
    max-datasize max-datasize;
    traceoptions {
        file <filename> <files number> <size size> <world-readable | no-world-readable>;
        flag flag;
        no-remote-trace;
    }
}
```

Hierarchy Level

[edit system extensions extension-service]

Release Information

Statement introduced in Junos OS Release 16.1.

Description

Configure the Junos OS extension service application.

NOTE: Global traceoptions for daemonized applications do not take effect if the daemonized application and global traceoptions are committed separately.

The remaining statements are explained separately. See CLI Explorer.

Required Privilege Level
maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

RELATED DOCUMENTATION

extension-service (System Extensions) | 71

extensions

Syntax

```
extensions {
    extension-service {
        application {
            file script-name {
                arguments arguments;
                checksum hash-algorithm hash-value;
                daemonize;
                refresh;
                refresh-from;
                routing-instance;
                source;
                traceoptions {
                    file <filename> <files number> <match regex> <size size> <world-readable | no-world-readable>;
                    flag flag;
                    no-remote-trace;
                }
                username username;
            }
            max-datasize max-datasize;
            traceoptions {
                file <filename> <files number> <size size> <world-readable | no-world-readable>;
                flag flag;
                no-remote-trace;
            }
        }
    }
    providers {
        provider-id {
            license-type license deployment-scope [ deployments ];
        }
    }
    resource-limits {
        package package-name {
            resources {
                cpu {
                    priority number;
                    time seconds;
                }
                file {
                    core-size bytes;
```

```
            open number;
            size bytes;
          }
          memory {
            data-size bytes;
            locked-in bytes;
            resident-set-size bytes;
            socket-buffers bytes;
            stack-size bytes;
          }
        }
      }
      process process-ui-name {
        resources {
          cpu {
            priority number;
            time seconds;
          }
          file {
            core-size bytes;
            open number;
            size bytes;
          }
          memory {
            data-size bytes;
            locked-in bytes;
            resident-set-size bytes;
            socket-buffers bytes;
            stack-size bytes;
          }
        }
      }
    }
}
```

Hierarchy Level

```
[edit system]
```

Release Information

Statement introduced in Junos OS Release 9.0.

extension-service option introduced in Junos OS Release 16.1.

Description

Configure extensions to Junos OS.

You must configure the **providers** statement to enable application packages developed using the Junos SDK to be deployed and run on the router.

You must configure the **extension-service** statement to enable application packages developed using the Juniper Extension Toolkit (JET) to be deployed and run on the device.

The remaining statements are explained separately. See CLI Explorer.

> **NOTE:** This configuration is optional in Junos OS Evolved. You can run JET applications directly using a Python interpreter instead of configuring and invoking them in the CLI.

Required Privilege Level

admin—To view this statement in the configuration.
admin-control—To add this statement to the configuration.

extension-service (System Extensions)

Syntax

```
extension-service {
    application {
        file script-name {
            arguments arguments;
            checksum (md5 | sha-256 | sha1) hash;
            daemonize;
            refresh;
            refresh-from;
            respawn-on-normal-exit;
            routing-instance
            source;
            traceoptions {
                file <filename> <files number> <size size> <world-readable | no-world-readable>;
                flag flag;
                no-remote-trace;
            }
            username username;
        }
        max-datasize max-datasize;
        traceoptions {
            file <filename> <files number> <size size> <world-readable | no-world-readable>;
            flag flag;
            no-remote-trace;
        }
    }
}
```

Hierarchy Level

```
[edit system extensions]
```

Release Information

Statement introduced in Junos OS Release 16.1.

respawn-on-normal-exit option introduced in Junos OS Releases 17.3R3 and 18.1R1.

Description

Enable Junos OS extension services.

The remaining statements are explained separately. See CLI Explorer.

Required Privilege Level

maintenance—To view this statement in the configuration.

maintenance-control—To add this statement to the configuration.

RELATED DOCUMENTATION

extensions | 68

extension-service (System Services gRPC)

List of Syntax

Junos OS Syntax on page 73
Junos OS Evolved Syntax on page 73

Junos OS Syntax

```
extension-service {
    request-response {
        grpc {
            max-connections max-connections;
            routing-instance routing-instance;
            ssl {
                address ip-address;
                local-certificate local-certificate;
                mutual-authentication {
                    certificate-authority certificate-authority-profile-name;
                    client-certificate-request (no-certificate | request-certificate | request-certificate-and-verify |
                        require-certificate | require-certificate-and-verfiy);
                }
                port port;
            }
        }
    }
    notification {
        allow-clients {
            address ip-address;
        }
        broker-socket-send-buffer-size broker-socket-send-buffer-size;
        max-connections max-connections;
        port port;
    }
    traceoptions {
        file <filename> <files number> <match regex> <size size> <world-readable | no-world-readable>;
        flag flag;
        no-remote-trace;
    }
}
```

Junos OS Evolved Syntax

```
extension-service {
```

```
request-response {
    grpc {
        max-connections max-connections;
        routing-instance routing-instance;
        ssl {
            address ip-address;
            local-certificate local-certificate;
            mutual-authentication {
                certificate-authority certificate-authority-profile-name;
                client-certificate-request (no-certificate | request-certificate | request-certificate-and-verify |
                    require-certificate | require-certificate-and-verfiy);
            }
            port port;
        }
    }
}
traceoptions {
    file <filename> <files number> <match regex> <size size> <world-readable | no-world-readable>;
    flag flag;
    no-remote-trace;
}
}
```

Hierarchy Level

```
[edit system services]
```

Release Information

Statement introduced in Junos OS Release 16.1.
grpc option introduced in Junos OS Release 16.2.

Description

Enable Junos OS extension services.

The remaining statements are explained separately. See CLI Explorer.

Required Privilege Level

system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

file (JET)

Syntax

```
file filename {
    arguments arguments;
    checksum hash-algorithm hash-value;
    daemonize;
    refresh;
    refresh-from;
    respawn-on-normal-exit;
    routing-instance;
    source;
    traceoptions {
        file <filename> <files number> <match regex> <size size> <world-readable | no-world-readable>;
        flag flag;
        no-remote-trace;
    }
    username username;
}
```

Hierarchy Level

```
[edit system extensions extension-service application]
```

Release Information

Statement introduced in Junos OS Release 16.1.
refresh, **refresh-from**, **respawn-on-normal-exit**, **routing-instance**, and **source** options added in Junos OS Release 18.1R1.

Description

For files in the **[edit system extensions extension-service application]** hierarchy level, specify the configuration for each file in the extension-service application.

Options

arguments *arguments*—Specify the command-line arguments called by a JET application. A program can take any number of command-line arguments. Enter the arguments in the way the application expects. Developer must supply this information.

daemonize—Specify the file as daemonized. An application runs as a daemonized process in the background. An application configured to run as a daemonized process is automatically triggered upon commit. A non-daemonized application must be triggered manually from the command-line client.

filename—Local filename of the script file.

respawn-on-normal-exit—Automatically restart a daemonized JET application written in Python after you exit the application normally, such as when you upgrade the JET application. If this option is not configured, and if the application normally exits, it will not restart automatically. This option can only be used with Python JET applications that have been configured to run as a daemonized process using the **daemonize** option.

username *username*—Specify the name of the user under whose privileges the extension service will execute. This user name is configured at the **[edit system login]** hierarchy level. If you do not associate a user name with an extension-service application, the application is executed as user **nobody**.

Default: nobody

The remaining statements are explained separately. See CLI Explorer.

Required Privilege Level

maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

grpc

Syntax

```
grpc {
    max-connections max-connections;
    routing-instance routing-instance;
    ssl {
        address ip-address;
        local-certificate local-certificate;
        mutual-authentication {
            certificate-authority certificate-authority-profile-name;
            client-certificate-request (no-certificate | request-certificate | request-certificate-and-verify | require-certificate
                | require-certificate-and-verfiy);
        }
        port port;
    }
}
```

Hierarchy Level

```
[edit system services extension-service request-response]
```

Release Information

Statement introduced in Junos OS Release 16.2.

Description

Configure the type of connections the gRPC service accepts for API applications.

Options

routing-instance *routing-instance*—Name of routing instance for grpc.

The remaining statements are explained separately. See CLI Explorer.

Required Privilege Level

system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

RELATED DOCUMENTATION

request-response | 88

interface-notification (programmable-rpd)

Syntax

```
interface-notification name;
```

Hierarchy Level

```
[edit logical-systems name routing-instances name routing-options programmable-rpd client id],
[edit logical-systems name routing-options programmable-rpd client id],
[edit routing-instances name routing-options programmable-rpd client id],
[edit routing-options programmable-rpd client id]
```

Release Information

Statement introduced in Junos OS Release 17.4R1.

Description

Restrict interface event notifications from the programmable routing protocol process (prpd) to specified JET clients and interfaces. The prpd provides public APIs to program routing systems, making it possible for users to directly access the APIs to customize, create, and modify behavior of their network.

Default

By default, no restrictions are imposed and JET clients are notified of all interfaces.

Options

name—Interface name

Required Privilege Level

routing

RELATED DOCUMENTATION

- show programmable-rpd clients | **104**
- traceoptions (programmable-rpd) | **97**
- purge-timeout (programmable-rpd) | **82**

max-connections

Syntax

```
max-connections max-connections;
```

Hierarchy Level

```
[edit system services extension-service request-response grpc]
```

Release Information

Statement introduced in Junos OS Release 16.1.

Description

Number of simultaneous connections for request-response that can be attached to jsd. The higher the number, the higher the impact on clients performance.

Options

Range: 1 through 8

Default: 8

Required Privilege Level

system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

notification (System Services)

Syntax

```
notification {
    allow-clients {
        address ip-address;
    }
    broker-socket-send-buffer-size broker-socket-send-buffer-size;
    max-connections max-connections;
    port port;
}
```

Hierarchy Level

```
[edit system services extension-service]
```

Release Information

Statement introduced in Junos OS Release 16.1.

Description

Enable notification services for applications running on devices running Junos OS.

Options

allow-clients address *ip-address*—Specify IPv4 or IPv6 addresses (prefix length optional) or host names from which notifications are allowed. You can specify a set of values using square brackets ([ ]).

broker-socket-send-buffer-size *broker-socket-send-buffer-size*—Socket send buffer size for the broker to publish the messages

max-connections *max-connections*—Specify the maximum number of connections.

Range: 1 through 20

Default: 20

port *port*—Specify the number of the port to accept incoming connections.

Range: 1 through 65535

Default: 1883

Required Privilege Level

system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

RELATED DOCUMENTATION

grpc | 77

providers

Syntax

```
providers {
   provider-id {
      license-type license deployment-scope [ deployments ];
   }
}
```

Hierarchy Level

[edit system extensions]

Release Information

Statement introduced in Junos OS Release 9.0.
license-type option introduced in Junos OS Release 10.2.

Description

Activate the certificate of the provider of the application and enable the PIC for loading of the application.

Options

provider-id—Provider ID for the application package. The provider ID identifies the provider of the application to the system. The provider ID must be activated on the device to allow the application to be deployed on the device and run.

license-type—Configure the license type and the scope of application deployment.

license—Type of license. Obtain correct value from the application's provider.

deployment—Scope of application deployment. You can configure a set of deployments. Obtain correct value from the application's provider.

Required Privilege Level

admin—To view this statement in the configuration.
admin-control—To add this statement to the configuration.

purge-timeout (programmable-rpd)

Syntax

```
purge-timeout {
    never;
    timeout <seconds>
}
```

Hierarchy Level

```
[edit routing-options programmable-rpd]
```

Release Information
Statement introduced in Junos OS Release 16.2.
never option introduced in Junos OS Release 18.4.

Description

Set the time, in seconds, after which a disconnected client times-out. Upon disconnect, the client state remains available but no operations occur. If the disconnected client reconnects before the set time has elapsed, the states are restored on the router. If it does not, all client operations are reverted and the programmable routing protocol process (prpd) server notifies any other modules of the disconnect so they can purge any other client operations.

When the purge-timeout is set to **never**, the prpd-client-added routes are not deleted when the client disconnects and does not reconnect back. The routes are deleted only when the client explicitly deletes the routes. If routing is restarted, then the client-added routes are lost.

The prpd provides public APIs to program routing systems, making it possible for users to directly access the APIs to customize, create and modify behavior of their network.

Options

Values:

never—When this option is configured purge timeout is infinite (that is, client added routes never time out) for the BGP route service.

timeout *seconds* —(Optional) Set the time, in seconds, after which disconnected clients time-out on the prpd server and the routes added by the client are purged.

Default: 120

Range: 1 through 604,800

NOTE: Starting in Junos OS Releases 18.4R1, the maximum **purge-timeout** value is 604,800 seconds (7 days). Prior to this release, the maximum value was 1000 seconds.

Required Privilege Level

routing and trace—To view this statement in the configuration.

routing-control and trace-control—To add this statement to the configuration.

RELATED DOCUMENTATION

traceoptions | **97**

show programmable-rpd clients | **104**

show route

refresh (JET)

Syntax

```
refresh;
```

Hierarchy Level

```
[edit system extensions extension-service application file filename]
```

Release Information
Statement introduced in Junos OS Release 18.1R1.

Description
Overwrite the local copy of all enabled commit scripts or a single enabled commit script with the copy located at the source URL, as specified in the **source** statement at the same hierarchy level. If the **load-scripts-from-flash** statement is configured, the device refreshes the scripts on the flash drive instead of the hard disk.

The update operation occurs as soon as you issue the **set refresh** configuration mode command. Issuing the **set refresh** command does not add the **refresh** statement to the configuration. Thus the command behaves like an operational mode command by executing an operation, instead of adding a statement to the configuration.

> **NOTE:** On the QFabric system, commit scripts are stored in the **/pbdata/mgd_shared/*partition-ip*/var/db/scripts/commit/** directory on the Director device.

As of Junos OS Release 18.1R1, you can specify which routing instance the update is done through. To specify the routing instance to use for updating commit scripts, configure the routing instance in two places in the CLI:

```
user@host# set system routing-instances routing-instance-name description description

user@host# set system extensions extension-service application file filename
routing-instance routing-instance-name
```

If you enable the non-default managment instance and use **mgmt_junos** for ***routing-instance-name***, you can configure scripts to update using the dedicated management instance mgmt_junos.

Required Privilege Level

maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

RELATED DOCUMENTATION

refresh-from (JET) | 86

Configuring and Using a Master Source Location for a Script

Example: Configuring and Refreshing from the Master Source for a Script

Management Interface in a Nondefault Instance

refresh-from (JET)

Syntax

```
refresh-from url;
```

Hierarchy Level

```
[edit system extensions extension-service application file filename]
```

Release Information

Statement introduced in Junos OS Release 18.1R1.

Description

Overwrite the local copy of all enabled commit scripts or a single enabled commit script with the copy located at the specified URL. If the **load-scripts-from-flash** statement is configured, the device refreshes the scripts on the flash drive instead of the hard disk.

The update operation occurs as soon as you issue the **set refresh-from *url*** configuration mode command. Issuing the **set refresh-from** command does not add the **refresh-from** statement to the configuration. Thus the command behaves like an operational mode command by executing an operation, instead of adding a statement to the configuration.

NOTE: This statement is not supported on the QFabric system.

As of Junos OS Release 18.1R1, you can specify which routing instance the update is done through. To specify the routing instance to use for updating op scripts, configure the routing instance in two places in the CLI:

```
user@host# set system routing-instances routing-instance-name description description
user@host# set system extensions extension-service application file filename routing-instance
    routing-instance-name
```

If you enable the non-default managment instance and use **mgmt_junos** for ***routing-instance-name***, you can configure scripts to update using the dedicated management instance mgmt_junos.

Options

url—The source specified as a Hypertext Transfer Protocol (HTTP) URL, FTP URL, or secure copy (scp)-style remote file specification.

Required Privilege Level

maintenance—To view this statement in the configuration.

maintenance-control—To add this statement to the configuration.

RELATED DOCUMENTATION

Using an Alternate Source Location for a Script

refresh (JET) | **84**

source (JET Scripts) | **92**

routing-instance (JET Scripts) | **91**

request-response

Syntax

```
request-response {
    grpc {
        max-connections max-connections;
        routing-instance routing-instance;
        ssl {
            address ip-address;
            local-certificate local-certificate;
            mutual-authentication {
                certificate-authority certificate-authority-profile-name;
                client-certificate-request (no-certificate | request-certificate | request-certificate-and-verify |
                    require-certificate | require-certificate-and-verfiy);
            }
            port port;
        }
    }
}
```

Hierarchy Level

```
[edit system services extension-service]
```

Release Information

Statement introduced in Junos OS Release 16.1.
grpc option introduced in Junos OS Release 16.2.

Description

Allow request-response API execution.

Statements are explained separately.

Required Privilege Level

system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

rib-service (programmable-rpd)

Syntax

```
rib-service {
    dynamic-next-hop-interface (enable | disable);
}
```

Hierarchy Level

```
[edit routing-options programmable-rpd]
```

Release Information

Statement introduced in Junos OS Release 20.2R1.

Description

Configure programmable routing protocol process (prpd) options that apply only to RIB service APIs.

Options

dynamic-next-hop-interface (enable | disable)—Disable or enable dynamic next-hop interface binding. When enabled, programmed RIB routes react to Up, Down, Add, and Delete events for direct next-hop interfaces. When all next-hop interfaces are unconfigured or down, the route is hidden and becomes inactive. When a next-hop interface is configured or comes up, the route becomes visible and active. This prevents dropped traffic and keeps inactive routes from being propagated through the network.

This feature applies to all routes programmed using the rib_service JET API where an interface is configured as a direct next-hop, including interfaces that are part of a flexible tunnel. It also applies to tunnels configured with the flexible_tunnel_service JET API. Indirect next-hops are resolved by the RPD resolver normally.

Changes to the configuration of this statement only affect routes programmed using Junos OS Release 20.2R1 or later.

Default: This option is enabled by default starting in Junos OS Release 20.2R1. In earlier releases, dynamic next-hop interface binding is disabled by default.

Required Privilege Level

routing

RELATED DOCUMENTATION

Understanding Programmable Flexible VXLAN Tunnels

routing-instance (JET Scripts)

Syntax

```
routing-instance routing-instance-name;
```

Hierarchy Level

```
[edit system extensions extension-service application file filename]
```

Release Information

Statement introduced in Junos OS Release 18.1R1.

Description

Configure the routing instance you want to use to update Automation scripts. To use a management instance, configure the **management-instance** statement along with the **routing-instance** statement, thus enabling JET scripts to use the non-default management routing instance mgmt_junos when refreshing the scripts.

Options

routing-instance-name—Name of the routing instance. For the management instance, use **mgmt_junos**. Otherwise, you can specify any routing instance name.

NOTE: You must also define the routing instance under the **[edit routing-instances]** hierarchy level.

Required Privilege Level

system—To view this statement in the configuration.
system-control—To add this statement to the configuration.

RELATED DOCUMENTATION

management-instance

Management Interface in a Nondefault Instance

source (JET Scripts)

Syntax

```
source url;
```

Hierarchy Level

```
[edit system extensions extension-service application file filename]
```

Release Information

Statement introduced in Junos OS Release 18.1R1.

Description

Specify the location of the master source file for a JET script. When you issue the **set refresh** configuration mode command at the same hierarchy level, the local copy of the script is overwritten by the version stored at the specified URL. If the **load-scripts-from-flash** statement is configured, the device refreshes the scripts on the flash drive instead of the hard disk.

NOTE: JET scripts are stored in the **/var/db/scripts/jet** directory.

Options

url—Master source file for a JET script specified as an HTTP URL, FTP URL, or scp-style remote file specification.

Required Privilege Level

maintenance—To view this statement in the configuration.
maintenance-control—To add this statement to the configuration.

RELATED DOCUMENTATION

Configuring and Using a Master Source Location for a Script

Example: Configuring and Refreshing from the Master Source for a Script

traceoptions (Extensions)

Syntax

```
traceoptions {
    file <filename> <files number> <match regex> <size size> <world-readable | no-world-readable>;
    flag flag;
    no-remote-trace;
}
```

Hierarchy Level

```
[edit system extensions extension-service application],
[edit system extensions extension-service application file script-name]
```

Release Information

Statement introduced in Junos OS Release 16.1 for MX80, MX104, MX240, MX480, MX960, MX2010, MX2020, vMX Series.
Statement introduced in Junos OS Release 19.1R1 for the **[edit system extensions extension-service application file *script-name*** hierarchy level.

Description

Trace options for extension-service applications. You can set the **traceoptions** statement for the **application** level or for the **file *script-name*** level or for both. If the **traceoptions** statement is set for both the **application** and **file *script-name*** levels, the latter will have a higher priority.

> **NOTE:** Global traceoptions for daemonized applications do not take effect if the daemonized application and global traceoptions are committed separately

Options

file—Indicate trace file information.

filename—Name of the file to receive the tracing operation output. Enclose the name in quotation marks. Traceoption output files are located in the **/var/log/** directory.

files *number*—(Optional) Specify maximum number of trace files.

Range: 2 through 1000

Default: 3

size *size*—(Optional) Specify the maximum size of each trace file. When a trace file named ***trace-file*** reaches its maximum size, it is renamed ***trace-file*.0**. The traceoption output continues in a second trace file named ***trace-file*.1**. When ***trace-file*.1** reaches its maximum size, output continues in a third file named ***trace-file*.2**, and so on. When the maximum number of trace files is reached, the oldest trace file is overwritten.

Range: 10240 through 1073741824

Default: 128k

world-readable | no-world-readable—(Optional). Grant all users permission to read log files, or restrict the permission only to the root user and users who have Junos OS maintenance permission.

flag *flag*—Specify the tracing operation to perform. To specify more than one tracing operation, include multiple **flag** statements:

all—Trace all operations.

config—Trace important events.

general—Trace script input data.

grpc—Trace grpc server events.

notification—Trace notification events.

routing-socket—Trace routing socket calls.

timeouts—Trace timeouts.

timer—Trace internal timer events.

no-remote-trace—Disable remote tracing. This option is valid only when [system tracing] is configured.

Required Privilege Level
trace—To view this statement in the configuration.
trace-control—To add this statement to the configuration.

RELATED DOCUMENTATION

application | 66

traceoptions (Services)

Syntax

```
traceoptions {
    file <filename> <files number> <match regex> <size size> <world-readable | no-world-readable>;
    flag flag;
    no-remote-trace;
}
```

Hierarchy Level

```
[edit system services extension-service]
```

Release Information

Statement introduced in Junos OS Release 16.1.

Trace flag options **libgrpc-debug, libgrpc-errors,** and **libgrpc-info** are supported in Junos OS Release 19.3R1 only.

Description

Define tracing operations for the JET service process (jsd).

Options

file—Indicate trace file information.

filename—Name of the file to receive the tracing operation output. Enclose the name in quotation marks. Traceoption output files are located in the **/var/log/** directory.

files *number*—(Optional) Specify the maximum number of trace files.

Range: 2 through 1000

Default: 10

match *regex*—Specify the regular expression for lines to be logged.

size *size*—(Optional) Specify the maximum size of each trace file. When a trace file named ***trace-file*** reaches its maximum size, it is renamed ***trace-file*.0**. The traceoption output continues in a second trace file named ***trace-file*.1**. When ***trace-file*.1** reaches its maximum size, output continues in a third file named ***trace-file*.2**, and so on. When the maximum number of trace files is reached, the oldest trace file is overwritten.

Range: 10,240 through 1,073,741,824 bytes

Default: 1000k

world-readable | no-world-readable—(Optional). Grant all users permission to read log files, or restrict the permission only to the root user and users who have Junos OS maintenance permission.

flag *flag*—Specify the tracing operation to perform. To specify more than one tracing operation, include multiple **flag** statements:

- **all**—Trace everything.
- **config**—Trace configuration events.
- **general**—Trace general events.
- **grpc**—Trace grpc server events.
- **libgrpc-debug**—(Junos OS Release 19.3R1 only) Trace all lib grpc-related events.
- **libgrpc-errors**—(Junos OS Release 19.3R1 only) Trace lib grpc errors.
- **libgrpc-info**—(Junos OS Release 19.3R1 only) Trace lib grpc info and errors.
- **notification**—Trace notification events.
- **routing-socket**—Trace routing socket calls
- **timeouts**—Trace timeouts.
- **timer**—Trace internal timer events.

NOTE: The libgrpc traceoptions are only supported in Junos OS Release 19.3R1. They allow you to see events from the grpc libraries in the jsd trace log. Within the grpc core there are multiple libraries (for example, iomgr, compression, and profiling).

no-remote-trace—Disable remote tracing.

Required Privilege Level
trace—To view this statement in the configuration.
trace-control—To add this statement to the configuration.

traceoptions (programmable-rpd)

Syntax

```
traceoptions {
    file filename <files number> <size size> <world-readable | no-world-readable>;
    flag flag <disable>;
}
```

Hierarchy Level

```
[edit routing-options programmable-rpd]
flag <flags>
file <filename> <size>
```

Release Information

Statement introduced in Junos OS Release 16.2.

Description

Starts logging traces related to the programmable routing protocol process (prpd). The prpd provides public APIs to program routing systems, making it possible for users to directly access the APIs to customize, create and modify behavior of their network.

Use the **traceoptions** command, along with related show commands, to help debug client-server interactions, identify the flow of control, and detect errors, get client-level information and statistics.

You can filter traces according to the flag(s) you have enabled.

Default

If you do not include this statement, no tracing operations are performed.

Options

Values:

file *filename*—Name of the file to receive the output of the tracing operation. Enclose the name within quotation marks. All files are placed in the directory **/var/log**.

files *number*—(Optional) Maximum number of trace files. When a trace file named ***trace-file*** reaches its maximum size, it is renamed ***trace-file*.0**, then ***trace-file*.1**, and so on, until the maximum number of trace files is reached. Then, the oldest trace file is overwritten. Note that if you specify a maximum number of files, you also must specify a maximum file size with the **size** option.

Range: 2 through 1000 files

Default: 10 files

no-world-readable—(Optional) Prevent any user from reading the log file.

size *size*—(Optional) Maximum size of each trace file, in kilobytes (KB), megabytes (MB), or gigabytes (GB). When a trace file named ***trace-file*** reaches this size, it is renamed ***trace-file*.0**. When the ***trace-file*** again reaches its maximum size, ***trace-file*.0** is renamed ***trace-file*.1** and ***trace-file*** is renamed ***trace-file*.0**. This renaming scheme continues until the maximum number of trace files is reached. Then, the oldest trace file is overwritten. Note that if you specify a maximum file size, you also must specify a maximum number of trace files with the **files** option.

Syntax: ***x*k** to specify KB, ***x*m** to specify MB, or ***x*g** to specify GB

Range: 1024 to 4,294,967,295 bytes

Default: 128 KB

world-readable—(Optional) Allow any user to read the log file.

flag *flag*—Specifies the tracing operation to perform. To specify more than one tracing operation, include multiple **flag** statements. The options are:

- **all**—All tracing operations.
- **client**—Client events.
- **general**—All normal operations and routing table changes (a combination of the **normal** and **route** trace operations).
- **normal**—All normal operations.
- **policy**—Routing policy operations and actions.
- **route**—Routing table changes.
- **state**—State transitions.
- **task**—Interface transactions and processing.
- **timer**—Timer usage.

Required Privilege Level

routing and trace—To view this statement in the configuration.
routing-control and trace-control—To add this statement to the configuration.

RELATED DOCUMENTATION

purge-timeout | 82

show programmable-rpd clients | 104

show route

5
CHAPTER

Operational Commands

request extension-service (start | stop)

Syntax

```
request extension-service (start | stop) application-name
<invoke-debugger cli>
```

Release Information

Command introduced in Junos OS Release 16.1 for MX80, MX104, MX240, MX480, MX960, MX2010, MX2020, vMX Series.

Command introduced in Junos OS Evolved Release 20.1R1 for PTX10003, PTX10008, PTX10016, and QFX5220.

Description

Start or stop a JET application running on a device running Junos OS.

Options

application-name—Name of application to be started or stopped.

invoke-debugger cli—(Optional) Starts the extension service process in debugger mode.

Required Privilege Level

maintenance

Output Fields

When you enter this command, you are provided feedback on the status of your request.

Sample Output

`user@device>` **request extension-service start cmdline_args.py**

```
Extension-service application 'cmdline_args.py' started with pid: 99418
```

show extension-service status

Syntax

```
show extension-service status (application-name | all)
```

Release Information

Command introduced in Junos OS Release 16.1 for MX80, MX104, MX240, MX480, MX960, MX2010, MX2020, vMX Series.

Command introduced in Junos OS Evolved Release 20.1R1 for PTX10003, PTX10008, PTX10016, and QFX5220.

Description

Display the status of all JET applications.

> **NOTE:** The **show extension-service status** operational command is limited to use with Python applications only.

Options

application-name—Display information for a single application.

all—Display information for all JET applications running on the system.

Required Privilege Level

view

List of Sample Output

show extension-service status on page 102
show extension-service status all on page 102
show extension-service status all (when no applications are active) on page 102

Output Fields

Table 7 on page 101 lists the output fields for the **show extension-service status** command.

Table 7: show extension-service status Output Fields

Field Name	Field Description
Name	Name of the application.
Arguments	Arguments passed to the application.

Table 7: show extension-service status Output Fields *(continued)*

Field Name	Field Description
Process-id	Process ID.
Stack-Segment-Size	Size of the stack segment memory.
Data-Segment-Size	Size of the data segment memory.

Sample Output

show extension-service status

```
user@host> show extension-service status application-one
```

```
Extension service application details:
Name : application-one
Arguments: -arg1 foo -arg2 goo
Process-id: 52592
Stack-Segment-Size: 16777216B
Data-Segment-Size: 134217728B
```

show extension-service status all

```
user@host> show extension-service status all
```

```
Extension service application details:
Name : application-name1
Arguments: -arg1 foo -arg2 goo
Process-id: 54834
Stack-Segment-Size: 16777216B
Data-Segment-Size: 134217728B
Name : application-name2
Arguments: -arg1 foo -arg2 goo
Process-id: 55011
Stack-Segment-Size: 16777216B
Data-Segment-Size: 134217728B
```

show extension-service status all (when no applications are active)

```
user@host> show extension-service status all
```

```
warning: No active extension-services
```

show programmable-rpd clients

Syntax

```
show programmable-rpd clients
```

Release Information
Command introduced in Junos OS Release 16.2.

Description
Lists clients connected to the programmable routing protocol process (prpd) server. The prpd provides public APIs to program routing systems, making it possible for users to directly access the APIs to customize, create and modify behavior of their network.

The command output shows client specific details and statistics such as client ID, protocol and corresponding gateway handle, purge timer, the client up/down status, and, if the client is disconnected, the time remaining before the client state is purged. If the client has not registered any protocol, the gateway handle is 0.

Required Privilege Level
view

RELATED DOCUMENTATION

purge-timeout | 82

show route

List of Sample Output
show programmable-rpd clients on page 104

Output Fields

Sample Output

show programmable-rpd clients

`user@host>` **show programmable-rpd clients**

```
RPD global purge timeout: 120
RPD Server connected client details:
ClientIdentifier   PurgeTimer  Status   Timeout  Protocol      Gateway
```

```
3                   150         Up                BGP-Static      578
2                    75         Up                NoGwProtocol      0
1                   120         Down     117      BGP-Static       577
```