

Juniper Extension Toolkit Developer Guide

Published
2021-12-14

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

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

Juniper Extension Toolkit Developer Guide
Copyright © 2021 Juniper Networks, Inc. All rights reserved.

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 This Guide | vi

1

Getting Started

JET Overview | 2

Set Up the JET VM | 5

Overview | 5

Download the Packages You Need | 5

Initialize Your Virtual Machine | 6

Set Up Your Virtual Machine Environment | 7

2

Application Development

Develop Off-Device JET Applications | 10

Overview | 10

Download and Compile the IDL File | 11

Develop and Package Your Application | 13

Prepare to Deploy Your Application | 14

Example: Python JET Application | 16

Develop On-Device JET Applications | 22

Overview | 23

Develop Unsigned JET Applications | 25

Develop Signed JET Applications | 25

Compile 64-Bit Applications | 28

Example: Develop a Signed C Package | 28

Example: Develop a Signed Python Package Without C Dependencies | 32

Example: Develop a Signed Python Package With C Dependencies | 37

Package JET Applications | 41

Debug JET Applications | 47

Debugging Tips | 47

How to Invoke the Debugger During Install | 48

Issue: Cannot Connect to jsd | 49

3

Additional Resources

Additional Resources | 52

4

Configuration Statements

application (Extensions) | 55

extensions | 57

extension-service (System Extensions) | 60

extension-service (System Services gRPC) | 63

file (JET) | 65

grpc | 68

interface-notification (programmable-rpd) | 70

max-connections | 72

notification (System Services) | 73

providers | 75

purge-timeout (programmable-rpd) | 77

refresh (JET) | 79

refresh-from (JET) | 81

request-response | 83

rib-service (programmable-rpd) | 85

routing-instance (JET Scripts) | 87

source (JET Scripts) | 89

traceoptions (Extensions) | 91

traceoptions (Services) | 94

traceoptions (programmable-rpd) | 97

5

Operational Commands

request extension-service (start | stop) | 102

show extension-service request-response clients | 103

show extension-service request-response servers | 107

show extension-service status | 109

show programmable-rpd clients | 112

About This Guide

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](#).

1

CHAPTER

Getting Started

[JET Overview | 2](#)

[Set Up the JET VM | 5](#)

JET Overview

IN THIS SECTION

- [Benefits of JET | 2](#)
- [JET Architecture | 2](#)
- [JET and gRPC | 4](#)

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

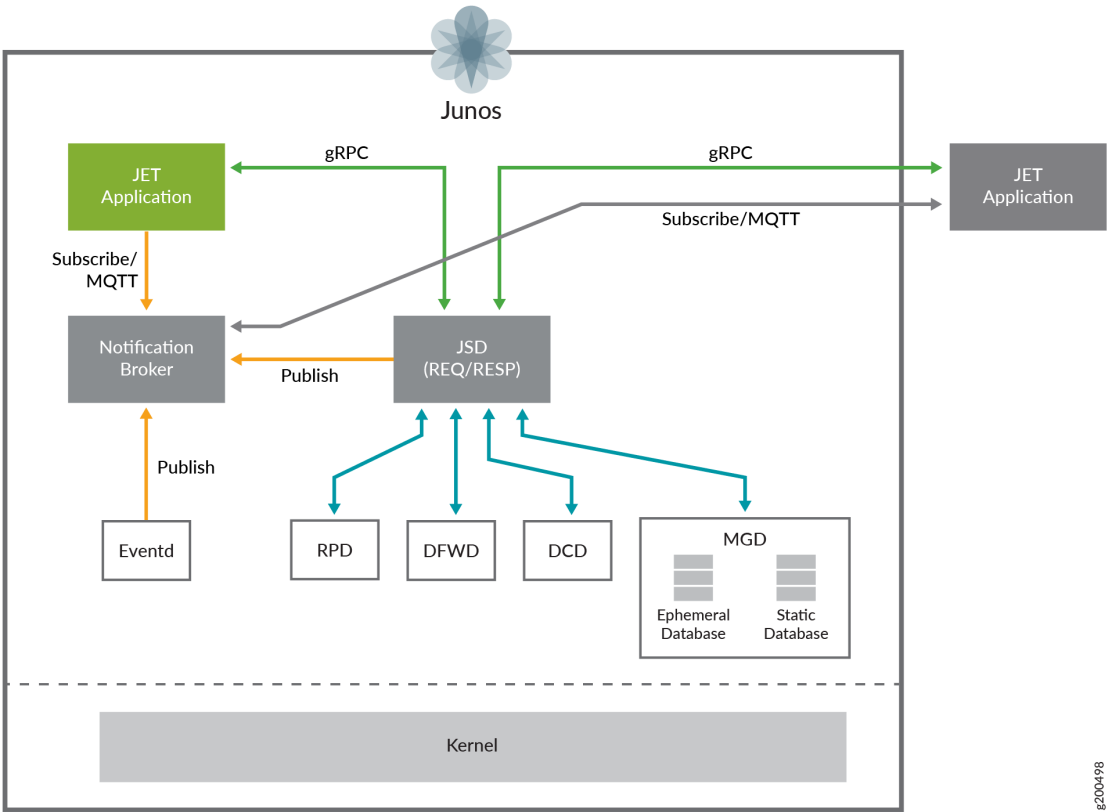
- Provides APIs to interact with any device running Junos OS.
- Supports API development in multiple languages.
- Provides tools to develop applications that run on Junos devices.
- Uses an event notification method that enables JET applications to respond to selected system events.

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 3](#) illustrates the request-response and notification services.

Figure 1: JET Request-Response and Notification Services



[Table 1 on page 4](#) describes the request-response and notification services.

Table 1: 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. | <p>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.</p> <p>NOTE: For secure communications with jsd, use RSA certificates, specifically TLSv1.2 (minimum).</p> |
| Notifications—An application can receive asynchronous notifications of events happening on Junos OS. | <p>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.</p> <p>JET utilizes Message Queue Telemetry Transport (MQTT) protocol (see https://mqtt.org/) method to implement the notification service.</p> |

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 | 5
- Download the Packages You Need | 5
- Initialize Your Virtual Machine | 6
- Set Up Your Virtual Machine Environment | 7

WHAT'S NEXT

"Develop Off-Device JET Applications" | 10

"Develop On-Device JET Applications" | 22

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.

2

CHAPTER

Application Development

Develop Off-Device JET Applications | 10

Develop On-Device JET Applications | 22

Package JET Applications | 41

Debug JET Applications | 47

Develop Off-Device JET Applications

IN THIS SECTION

- [Overview | 10](#)
- [Download and Compile the IDL File | 11](#)
- [Develop and Package Your Application | 13](#)
- [Prepare to Deploy Your Application | 14](#)
- [Example: Python JET Application | 16](#)

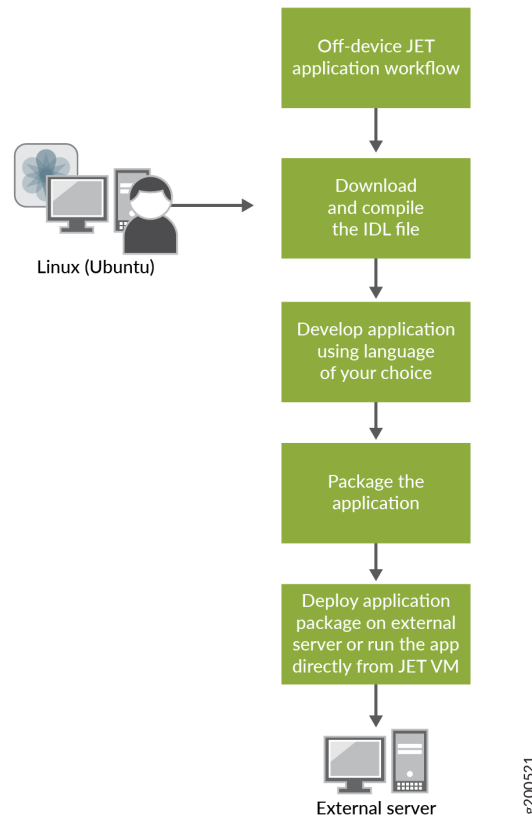
Overview

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 11 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
```

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 5](#) 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 41](#) 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

IN THIS SECTION

- [Configure JET Interaction with Junos OS | 14](#)

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

- [Junos OS Release 18.4R1 and Later | 17](#)
- [Before Junos OS Release 18.4R1 | 19](#)

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 14](#).

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](#) | 47

Develop On-Device JET Applications

IN THIS SECTION

- [Overview](#) | 23
- [Develop Unsigned JET Applications](#) | 25
- [Develop Signed JET Applications](#) | 25
- [Compile 64-Bit Applications](#) | 28
- [Example: Develop a Signed C Package](#) | 28
- [Example: Develop a Signed Python Package Without C Dependencies](#) | 32

- [Example: Develop a Signed Python Package With C Dependencies](#) | 37

Overview

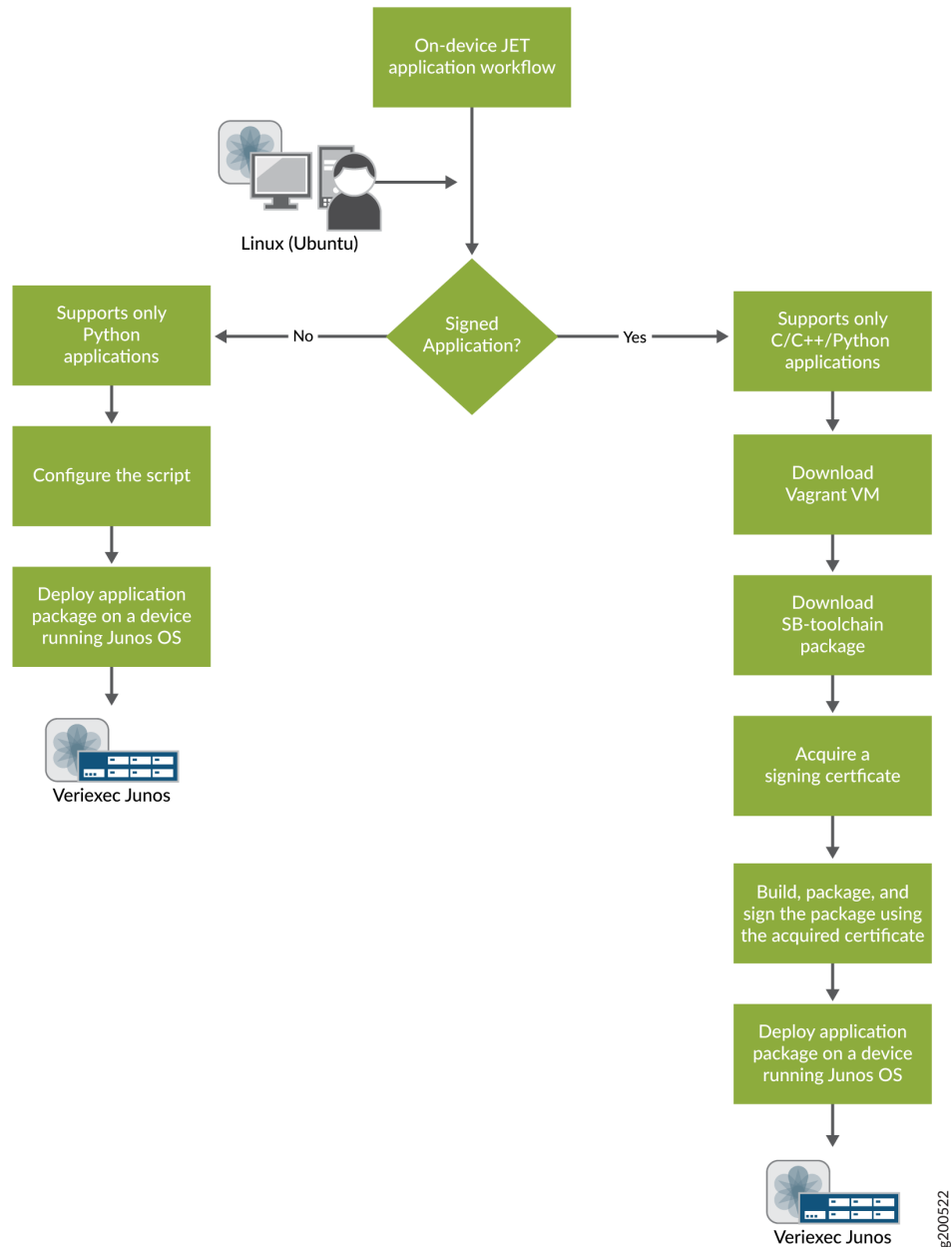
JET applications written in C, C++, and Python languages can run on-device. You can develop the applications in the downloaded JET VM and then deploy these applications on the device running Junos OS. You can sign on-device JET applications to show that they can be trusted.

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.

Starting in Junos OS Release 21.1R1, Python 2.7 is no longer supported and the `set system scripts language python` statement is deprecated. Use the `set system scripts language python3` statement instead. Junos OS Evolved Release 21.1R1 still supports Python 2.7.

Figure 3 on page 24 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 5](#).
2. Develop your application in Python.
3. Package your application. See ["Package JET Applications" on page 41](#).
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

IN THIS SECTION

- [Request a Signing Certificate | 26](#)

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 5](#).
2. Request a signing certificate. See ["Request a Signing Certificate" on page 26](#).
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. The organization should not be vague. There cannot be any hyphens ("-").
- Certificate type: Specify whether the certificate is commercial or private. Non-Juniper entities must use commercial. Private certificates are only assigned when the organization is Juniper.
- 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. See the requirements for each answer listed above.
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. All four parameters are mandatory or else you will see the following error:

```
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]*
```

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 <mailto:jet-cert@juniper.net>. This file contains your certificate request. The requestor should be authorized to request signing certificates

on behalf of their organization. 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.

Compile 64-Bit Applications

JET supports 64-bit applications for Junos OS with FreeBSD and Junos OS with upgraded FreeBSD. When you are ready to package your application, use the following commands to compile 64-bit applications for use with the AMD64 or ARM64 64-bit processor architecture.

To compile the application for use with AMD64 and Junos OS with FreeBSD:

1. Check you have the GCC toolchain, which should be included in the JET package.
2. Use the command `mk-amd64 application name` to build the application package.

To compile the application for use with AMD64 and Junos OS with upgraded FreeBSD:

1. Check you have the Clang toolchain, which should be included in the JET package.
2. Use the command `mk-amd64,bsdxc application name` to build the application package.

To compile the application for use with ARM64 and Junos OS with upgraded FreeBSD:

1. Check you have the Clang toolchain, which should be included in the JET package.
2. Use the command `mk-arm64,bsdxc application name` to build the application package.

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,bsdxc 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,bsdxc` instead of `mk-i386,bsdxc` 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,bsd 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 41](#) 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,bsd 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,bsd` instead of `mk-i386,bsd` 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 41](#) 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 -
I /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,bsd
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,bsd` instead of `mk-i386,bsd` 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')
```

Release History Table

| Release | Description |
|---------|---|
| 20.2R1 | 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. |

Package JET Applications

IN THIS SECTION

- Main Section Attributes | 41
- Source Attributes | 44
- Dependent Libraries | 45
- Dependent Python Modules | 46

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

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

Mandatory Attributes

Table 2 on page 42 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 2: 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" |

Table 2: Mandatory Attributes in the JSON File Main Section (Continued)

| Attribute | Description | Example Values |
|---------------|--|--|
| "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" |
| "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 3 on page 44](#) 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 3: 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). | " <i>flag1 flag2 flag3</i> " |
| "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). | " <i>flag1 flag2 flag3</i> " |
| "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. | " <i>flag1 flag2 flag3</i> " |

Source Attributes

Table 4 on page 45 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 4: 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. | <pre> "srcs": { "python/sample_pyapp": ["a.py", "b.py"], "python/sample_pyapp/temp": ["temp1.py", "temp2.py"] } </pre> |
| "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. | <pre> "extn-srcs": { "python/sample_pyapp": ["foo.c", "bar.c"], "python/sample_pyapp/temp": ["1.cpp", "2.cpp"] } </pre> |

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 | 10](#)

[Develop On-Device JET Applications | 22](#)

Debug JET Applications

IN THIS SECTION

- [Debugging Tips | 47](#)
- [How to Invoke the Debugger During Install | 48](#)
- [Issue: Cannot Connect to jsd | 49](#)

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 primary 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:

```
netstat -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

[Develop Off-Device JET Applications](#) | 10

[Develop On-Device JET Applications](#) | 22

3

CHAPTER

Additional Resources

[Additional Resources](#) | 52

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 \(Extensions\)](#) | 55

[extensions](#) | 57

[extension-service \(System Extensions\)](#) | 60

[extension-service \(System Services gRPC\)](#) | 63

[file \(JET\)](#) | 65

[grpc](#) | 68

[interface-notification \(programmable-rpd\)](#) | 70

[max-connections](#) | 72

[notification \(System Services\)](#) | 73

[providers](#) | 75

[purge-timeout \(programmable-rpd\)](#) | 77

[refresh \(JET\)](#) | 79

[refresh-from \(JET\)](#) | 81

[request-response](#) | 83

[rib-service \(programmable-rpd\)](#) | 85

[routing-instance \(JET Scripts\)](#) | 87

[source \(JET Scripts\)](#) | 89

[traceoptions \(Extensions\)](#) | 91

[traceoptions \(Services\)](#) | 94

[traceoptions \(programmable-rpd\)](#) | 97

application (Extensions)

IN THIS SECTION

- [Syntax | 55](#)
- [Hierarchy Level | 56](#)
- [Description | 56](#)
- [Required Privilege Level | 56](#)
- [Release Information | 56](#)

Syntax

```

application {
    file script-name {
        arguments arguments;
        checksum hash-algorithm hash-value;
        daemonize;
        interpreter (bash | python | python3);
        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]
```

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.

Release Information

Statement introduced in Junos OS Release 16.1.

RELATED DOCUMENTATION

[extension-service \(System Extensions\)](#) | 60

extensions

IN THIS SECTION

- [Syntax](#) | 57
- [Hierarchy Level](#) | 59
- [Description](#) | 59
- [Required Privilege Level](#) | 60
- [Release Information](#) | 60

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]
```

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.

Release Information

Statement introduced in Junos OS Release 9.0.

extension-service option introduced in Junos OS Release 16.1.

extension-service (System Extensions)

IN THIS SECTION

- [Syntax | 61](#)
- [Hierarchy Level | 62](#)
- [Description | 62](#)
- [Required Privilege Level | 62](#)
- [Release Information | 62](#)

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]
```

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.

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.

RELATED DOCUMENTATION

[extensions](#) | 57

extension-service (System Services gRPC)

IN THIS SECTION

- [Junos OS Syntax | 63](#)
- [Junos OS Evolved Syntax | 64](#)
- [Hierarchy Level | 65](#)
- [Description | 65](#)
- [Required Privilege Level | 65](#)
- [Release Information | 65](#)

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-verify);  
        }  
        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;
        level <error>;
        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]
```

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.

Release Information

Statement introduced in Junos OS Release 16.1.

grpc option introduced in Junos OS Release 16.2.

file (JET)

IN THIS SECTION

- [Syntax | 66](#)
- [Hierarchy Level | 66](#)
- [Description | 67](#)

- Options | 67
- Required Privilege Level | 67
- Release Information | 68

Syntax

```
file filename {  
    arguments arguments;  
    checksum hash-algorithm hash-value;  
    daemonize;  
    interpreter (bash | python | python3);  
    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]
```


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

| | |
|--|---|
| <code>arguments</code> <i>arguments</i> | 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. |
| <code>daemonize</code> | 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. |
| <code>filename</code> | Local filename of the script file. |
| <code>interpreter</code> (<code>bash</code> <code>python</code> <code>python3</code>) | Specify whether a device running Junos OS Evolved should run a daemonized on-device JET application using Bash, Python, or Python 3. |
| <code>respawn-on-normal-exit</code> | 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 <code>daemonize</code> option. |
| <code>username</code> <i>username</i> | Specify the name of the user under whose privileges the extension service will execute. This user name is configured at the <code>[edit system login]</code> hierarchy level. If you do not associate a user name with an extension-service application, the application is executed as user <code>nobody</code> . <ul style="list-style-type: none"> • Default: <code>nobody</code> |

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.

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.

interpreter option added in Junos OS Evolved Release 20.3R1.

grpc

IN THIS SECTION

- [Syntax | 68](#)
- [Hierarchy Level | 69](#)
- [Description | 69](#)
- [Options | 69](#)
- [Required Privilege Level | 69](#)
- [Release Information | 69](#)

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]
```

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.

Release Information

Statement introduced in Junos OS Release 16.2.

RELATED DOCUMENTATION

| [request-response](#) | 83

interface-notification (programmable-rpd)

IN THIS SECTION

- [Syntax](#) | 70
- [Hierarchy Level](#) | 70
- [Description](#) | 71
- [Default](#) | 71
- [Options](#) | 71
- [Required Privilege Level](#) | 71
- [Release Information](#) | 71

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]
```


max-connections

IN THIS SECTION

- [Syntax | 72](#)
- [Hierarchy Level | 72](#)
- [Description | 72](#)
- [Options | 73](#)
- [Required Privilege Level | 73](#)
- [Release Information | 73](#)

Syntax

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

Hierarchy Level

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

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.

Release Information

Statement introduced in Junos OS Release 16.1.

notification (System Services)

IN THIS SECTION

- [Syntax | 74](#)
- [Hierarchy Level | 74](#)
- [Description | 74](#)
- [Options | 74](#)
- [Required Privilege Level | 75](#)
- [Release Information | 75](#)

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]
```

Description

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

Options

| | |
|---|--|
| <code>allow-clients address <i>ip-address</i></code> | 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 ([]). |
| <code>broker-socket-send-buffer-size <i>broker-socket-send-buffer-size</i></code> | Socket send buffer size for the broker to publish the messages |
| <code>max-connections <i>max-connections</i></code> | Specify the maximum number of connections. <ul style="list-style-type: none"> • 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.

Release Information

Statement introduced in Junos OS Release 16.1.

RELATED DOCUMENTATION

| [grpc](#) | [68](#)

providers

IN THIS SECTION

- [Syntax](#) | [76](#)
- [Hierarchy Level](#) | [76](#)
- [Description](#) | [76](#)
- [Options](#) | [76](#)
- [Required Privilege Level](#) | [77](#)
- [Release Information](#) | [77](#)

Syntax

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

Hierarchy Level

[edit system [extensions](#)]

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.

Release Information

Statement introduced in Junos OS Release 9.0.

license-type option introduced in Junos OS Release 10.2.

purge-timeout (programmable-rpd)

IN THIS SECTION

- [Syntax | 77](#)
- [Hierarchy Level | 78](#)
- [Description | 78](#)
- [Options | 78](#)
- [Required Privilege Level | 79](#)
- [Release Information | 79](#)

Syntax

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

Hierarchy Level

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

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.

Release Information

Statement introduced in Junos OS Release 16.2.

never option introduced in Junos OS Release 18.4.

RELATED DOCUMENTATION

[traceoptions \(programmable-rpd\) | 97](#)

[show programmable-rpd clients | 112](#)

show route

refresh (JET)

IN THIS SECTION

- [Syntax | 80](#)
- [Hierarchy Level | 80](#)
- [Description | 80](#)
- [Required Privilege Level | 81](#)
- [Release Information | 81](#)

Syntax

```
refresh;
```

Hierarchy Level

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

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 management 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.

Release Information

Statement introduced in Junos OS Release 18.1R1.

RELATED DOCUMENTATION

[refresh-from \(JET\) | 81](#)

[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 Non-Default Instance](#)

refresh-from (JET)

IN THIS SECTION

- [Syntax | 82](#)
- [Hierarchy Level | 82](#)
- [Description | 82](#)
- [Options | 83](#)
- [Required Privilege Level | 83](#)
- [Release Information | 83](#)

Syntax

```
refresh-from url;
```

Hierarchy Level

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

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 management 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.

Release Information

Statement introduced in Junos OS Release 18.1R1.

RELATED DOCUMENTATION

[Using an Alternate Source Location for a Script](#)

[refresh \(JET\) | 79](#)

[source \(JET Scripts\) | 89](#)

[routing-instance \(JET Scripts\) | 87](#)

request-response

IN THIS SECTION

- [Syntax | 84](#)
- [Hierarchy Level | 84](#)
- [Description | 84](#)
- [Required Privilege Level | 85](#)

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-verify);  
      }  
      port port;  
    }  
  }  
}
```

Hierarchy Level

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

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.

Release Information

Statement introduced in Junos OS Release 16.1.

grpc option introduced in Junos OS Release 16.2.

rib-service (programmable-rpd)

IN THIS SECTION

- [Syntax | 85](#)
- [Hierarchy Level | 86](#)
- [Description | 86](#)
- [Options | 86](#)
- [Required Privilege Level | 86](#)
- [Release Information | 87](#)

Syntax

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

Hierarchy Level

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

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

Release Information

Statement introduced in Junos OS Release 20.2R1.

RELATED DOCUMENTATION

Understanding Programmable Flexible VXLAN Tunnels

[show programmable-rpd clients](#) | 112

[traceoptions \(programmable-rpd\)](#) | 97

routing-instance (JET Scripts)

IN THIS SECTION

- [Syntax](#) | 87
- [Hierarchy Level](#) | 88
- [Description](#) | 88
- [Options](#) | 88
- [Required Privilege Level](#) | 88
- [Release Information](#) | 88

Syntax

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

Hierarchy Level

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

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.

Release Information

Statement introduced in Junos OS Release 18.1R1.

RELATED DOCUMENTATION

[management-instance](#)

[Management Interface in a Non-Default Instance](#)

source (JET Scripts)

IN THIS SECTION

- [Syntax | 89](#)
- [Hierarchy Level | 89](#)
- [Description | 90](#)
- [Options | 90](#)
- [Required Privilege Level | 90](#)
- [Release Information | 90](#)

Syntax

```
source url;
```

Hierarchy Level

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

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.

Release Information

Statement introduced in Junos OS Release 18.1R1.

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)

IN THIS SECTION

- [Syntax | 91](#)
- [Hierarchy Level | 91](#)
- [Description | 92](#)
- [Options | 92](#)
- [Required Privilege Level | 93](#)
- [Release Information | 93](#)

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]
```

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. |
| <i>filename</i> | Name of the file to receive the tracing operation output. Enclose the name in quotation marks. Traceoption output files are located in the <code>/var/log/</code> directory. |
| files <i>number</i> | (Optional) Specify maximum number of trace files. <ul style="list-style-type: none">• Range: 2 through 1000• Default: 3 |
| size <i>size</i> | (Optional) Specify the maximum size of each trace file. When a trace file named <i>trace-file</i> reaches its maximum size, it is renamed <i>trace-file.0</i> . The traceoption output continues in a second trace file named <i>trace-file.1</i> . When <i>trace-file.1</i> reaches its maximum size, output continues in a third file named <i>trace-file.2</i> , and so on. When the maximum number of trace files is reached, the oldest trace file is overwritten. <ul style="list-style-type: none">• 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 <i>flag</i> | Specify the tracing operation to perform. To specify more than one tracing operation, include multiple <code>flag</code> statements: |

| | |
|-----------------------------|------------------------------|
| <code>all</code> | Trace all operations. |
| <code>config</code> | Trace important events. |
| <code>general</code> | Trace script input data. |
| <code>grpc</code> | Trace grpc server events. |
| <code>notification</code> | Trace notification events. |
| <code>routing-socket</code> | Trace routing socket calls. |
| <code>timeouts</code> | Trace timeouts. |
| <code>timer</code> | 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.

Release Information

Statement introduced in Junos OS Release 16.1.

RELATED DOCUMENTATION

[application \(Extensions\)](#) | 55

traceoptions (Services)

IN THIS SECTION

- [Syntax | 94](#)
- [Hierarchy Level | 94](#)
- [Description | 94](#)
- [Options | 95](#)
- [Required Privilege Level | 96](#)
- [Release Information | 96](#)

Syntax

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

Hierarchy Level

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

Description

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

Options

| | |
|--|---|
| file | Indicate trace file information. |
| <i>filename</i> | 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 <i>number</i> | (Optional) Specify the maximum number of trace files. <ul style="list-style-type: none"> • Range: 2 through 1000 • Default: 10 |
| match <i>regex</i> | Specify the regular expression for lines to be logged. |
| size <i>size</i> | (Optional) Specify the maximum size of each trace file. When a trace file named <i>trace-file</i> reaches its maximum size, it is renamed <i>trace-file.0</i> . The traceoption output continues in a second trace file named <i>trace-file.1</i> . When <i>trace-file.1</i> reaches its maximum size, output continues in a third file named <i>trace-file.2</i> , and so on. When the maximum number of trace files is reached, the oldest trace file is overwritten. <ul style="list-style-type: none"> • 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 <i>flag</i> | Specify the tracing operation to perform. To specify more than one tracing operation, include multiple flag statements: <ul style="list-style-type: none"> • 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 trace flags 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).

level Set the trace log verbosity. Use the `error` option to only show error events.

NOTE: This `error` option does not apply to the libgrpc trace flags.

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.

Release Information

Statement introduced in Junos OS Release 16.1.

`level` option introduced in Junos OS Release 20.2R1.

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

traceoptions (programmable-rpd)

IN THIS SECTION

- [Syntax | 97](#)
- [Hierarchy Level | 97](#)
- [Description | 98](#)
- [Default | 98](#)
- [Options | 98](#)
- [Required Privilege Level | 99](#)
- [Release Information | 99](#)

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>
```

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:** *kb* to specify KB, *mb* to specify MB, or *gb* 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.

Release Information

Statement introduced in Junos OS Release 16.2.

RELATED DOCUMENTATION

[purge-timeout \(programmable-rpd\) | 77](#)

[show programmable-rpd clients | 112](#)

show route

5

CHAPTER

Operational Commands

`request extension-service (start | stop)` | 102

`show extension-service request-response clients` | 103

`show extension-service request-response servers` | 107

`show extension-service status` | 109

`show programmable-rpd clients` | 112

request extension-service (start | stop)

IN THIS SECTION

- [Syntax | 102](#)
- [Description | 102](#)
- [Options | 102](#)
- [Required Privilege Level | 103](#)
- [Output Fields | 103](#)
- [Sample Output | 103](#)
- [Release Information | 103](#)

Syntax

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

Description

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

Options

- | | |
|----------------------------|---|
| <i>application-name</i> | Name of application to be started or stopped. |
| <i>invoke-debugger cli</i> | (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

command-name

```
user@device> request extension-service start cmdline_args.py
Extension-service application 'cmdline_args.py' started with pid: 99418
```

Release Information

Command introduced in Junos OS Release 16.1.

show extension-service request-response clients

IN THIS SECTION

- [Syntax | 104](#)
- [Description | 104](#)
- [Options | 104](#)
- [Required Privilege Level | 104](#)
- [Output Fields | 104](#)

- [Sample Output | 105](#)
- [Release Information | 106](#)

Syntax

```
show extension-service request-response clients (detail | brief)
```

Description

Display the status of the request-response clients connected to the device.

Options

| | |
|------------------|---|
| <i>client-id</i> | The client identifier. |
| brief | (Default) Display a summary of the information. |
| detail | Display detailed information. |

Required Privilege Level

view

Output Fields

[Table 5 on page 105](#) lists the output fields for the `show extension-service request-response clients` command.

Table 5: show extension-service request-response clients Output Fields

| Field Name | Field Description |
|-------------------|---|
| Client ID | Client identifier. |
| Socket Address | Address of the socket. |
| Client Type | Type of the client. |
| Client Login Time | The most recent login time of the remote procedure call (gRPC) client. This is when the authentication request was received for the channel. |
| Channel Count | The number of channels. |
| User Name | The user name for which the session was authenticated in a gRPC session. If authentication is not required, this field displays as "No User." This helps you identify which users have requested programmable operations. |

Sample Output

show extension-service request-response clients

```
user@device> show extension-service request-response clients
```

```

Client ID          Socket Address          Client Type Client Login Time
Channel Count
ipv6:::ffff:10.209.0.224:45888 ipv6:::ffff:10.209.0.224:45888 gRPC      2020-02-24 04:08:11
UTC      1

unix:::20          unix:::20                gRPC      2020-02-23 15:23:47
UTC      1

```

show extension-service request-response clients detail

```
user@device> show extension-service request-response clients detail
```

```
Channel information:
```

```
Client ID: ipv6::ffff:10.209.0.224:45888
Socket Address: ipv6::ffff:10.209.0.224:45888
Client Type: gRPC
Client Login Time : 2020-02-24 04:08:11 UTC
Channel Count: 1
```

```
Channel target: unix:/var/run/japi_mgd-api
Channel status: GRPC_CHANNEL_READY
User Name: root
```

```
Channel information:
```

```
Client ID: unix::20
Socket Address: unix::20
Client Type: gRPC
Client Login Time : 2020-02-23 15:23:47 UTC
Channel Count: 1
```

```
Channel target: unix:/var/run/japi_na-grpcd
Channel status: GRPC_CHANNEL_READY
User Name: No User
```

Release Information

Command introduced in Junos OS.

Client Login Time and User Name output fields introduced in Junos OS Release 20.4R1 for PTX5000.

show extension-service request-response servers

IN THIS SECTION

- [Syntax | 107](#)
- [Description | 107](#)
- [Required Privilege Level | 107](#)
- [Output Fields | 107](#)
- [Sample Output | 108](#)
- [Release Information | 108](#)

Syntax

```
show extension-service request-response servers
```

Description

Display the status of the request-response servers connected to the device.

Required Privilege Level

view

Output Fields

[Table 6 on page 108](#) lists the output fields for the `show extension-service request-response servers` command.

Table 6: show extension-service request-response servers Output Fields

| Field Name | Field Description |
|-----------------|--|
| Max connections | The maximum number of simultaneous connections for request-response that can be attached to jsd. |
| Address | The address of the server. |
| Status | The status of the server. |

Sample Output

show extension-service request-response servers

```
user@device> show extension-service request-response servers
gRPC server information:
  Max connections: 5, Skip-authentication: Disabled

  Address: unix:/var/run/japi_jsd
  Status: Up, Type: Clear-text
```

Release Information

Command introduced in Junos OS.

show extension-service status

IN THIS SECTION

- [Syntax | 109](#)
- [Description | 109](#)
- [Options | 109](#)
- [Required Privilege Level | 110](#)
- [Output Fields | 110](#)
- [Sample Output | 110](#)
- [Release Information | 111](#)

Syntax

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

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

Output Fields

[Table 7 on page 110](#) 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. |
| 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
```

Release Information

Command introduced in Junos OS Release 16.1.

show programmable-rpd clients

IN THIS SECTION

- [Syntax | 112](#)
- [Description | 112](#)
- [Required Privilege Level | 112](#)
- [Output Fields | 113](#)
- [Sample Output | 113](#)
- [Release Information | 113](#)

Syntax

```
show programmable-rpd clients
```

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

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

Release Information

Command introduced in Junos OS Release 16.2.

RELATED DOCUMENTATION

[purge-timeout \(programmable-rpd\)](#) | 77

show route