# Network Configuration Example

## Configuring Zero Touch Deployment for ACX Routers

Modified: 2018-01-23

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



*Network Configuration Example Configuring Zero Touch Deployment for ACX Routers*


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 http://www.juniper.net/support/eula/. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

# Table of Contents

CHAPTER 1

# Overview



## About This Network Configuration Example

This network configuration example (NCE) illustrates how to deploy large numbers of ACX Series routers using a process referred to as zero touch deployment (ZTD) or rapid deployment. With the help of scripts and configuration templates (the "Pull" method), a powered-on ACX router can be autoconfigured to a production-ready state with little to no user intervention. For environments running Junos Space, the platform's device management capabilities can also be leveraged and integrated into the ZTD process (the "Push" method.) Configuration examples are included for both the ZTD Push method and ZTD Pull method.

## Customer Use Cases for ACX Series Routers

When deploying a large numbers of devices, the tasks involved—from initial unboxing through fully-configured—can come at a significant cost. If the devices are staged in a warehouse, should a base configuration be added to them before they are distributed to the field? Who creates and adds the base configuration? How do they add the base configuration?

To avoid these costs, should a customer use just-in-time shipping to have the devices arrive at the installation site with the installer? This option can provide a potential savings, but it can also add more complexity for the installer, who must spend more time installing and bringing up the devices onsite. More complexity can lead to configuration errors and deployment delays, which ultimately result in extra cost to the customer.

The range of deployment scenarios for ACX routers is quite large. They can be deployed in service provider networks where the entire network is privately owned; they can be installed as customer premises equipment (CPE) devices in telco networks; they can be installed as part of a mobile service provider (MSP) network at a remote cell site, communicating back to the MSP securely over the Internet.

Given the wide variety of installation scenarios, Juniper Networks has enabled ACX routers to be rapidly deployed and automatically configured simply by unpacking them and powering them on. These automatic configuration capabilities help users reduce cost, speed deployment, increase configuration accuracy, and make the deployment process scalable.

There are multiple use cases where ACX routers can be deployed in the access and aggregation segments of the service provider network. The network configuration examples illustrated in this document cover following use cases:

- Mobile backhaul (MBH)
- Residential aggregation
- Metro Ethernet for business access services
- Metro Ethernet for wholesale MBH services
- Field area network

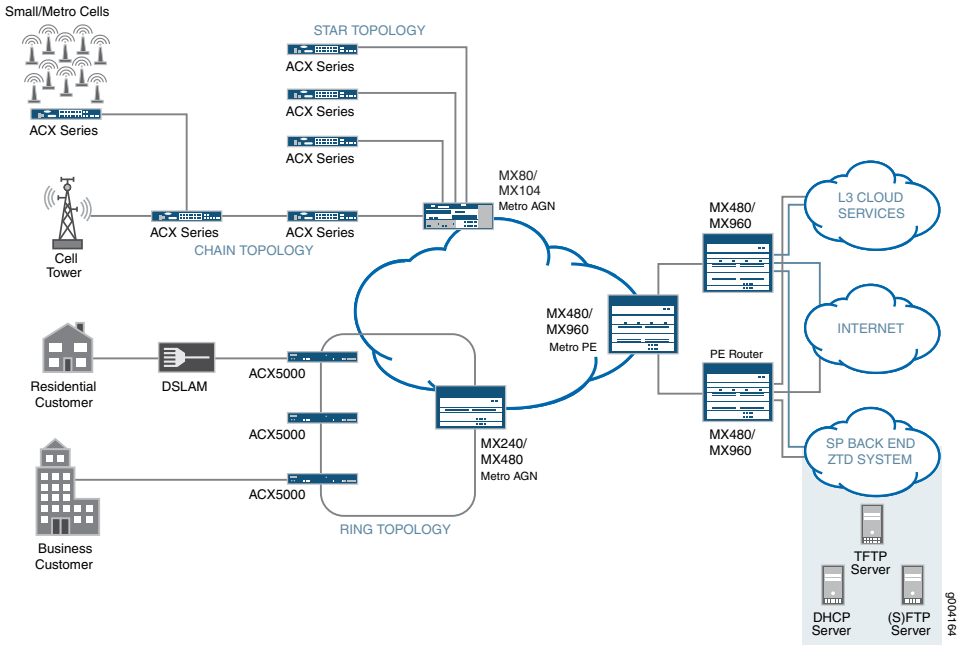
This NCE illustrates how zero touch deployment (ZTD) can be enabled in a service provider's metro network built with ACX routers. The use case can vary based on service provider preferences, as well as type of the technology used in the network to enable service delivery.

**IEEE 802.3 Ethernet bridging**

in this scenario, ACX routers are arranged in star, ring or chain topologies and connected to the metro aggregation router with Gigabit Ethernet or 10-Gigabit Ethernet optical links. The ACX routers are part of the metro infrastructure, which belongs to metro operators and serves to connect either end terminal units such as cell towers (NodeB, eNodeB, BTS, etc.) and customer CPEs, or other access nodes, such as DSLAMs, GPONs, etc. Metro Ethernet services are essentially enabled by means of 802.1ad (Q-in-Q) trunks.

Figure 1 on page 7 shows some of the locations where an ACX router can be deployed in a mobile service provider (MSP) or metro Ethernet environment.

Figure 1: ACX Router Deployment Locations




### IP/MPLS

Replacing IEEE 802.3 with IP/MPLS as the foundation technology in the access segment of the network leads to multiple benefits for the service provider when it comes to the service provisioning. On the other hand, this scenario is known to be rather complex during initial network deployment. ZTD with MPLS autoprovisioning helps to overcome these difficulties. While MPLS does not have the same plug-and-play type capabilities as Ethernet, and general automation of IP/MPLS networks can be difficult, the Junos OS includes support for a variety of automation tools as well as MPLS over IP-enabled unnumbered interfaces to help make this scenario possible.

### Wholesale/leased line

In this scenario, a mobile service provider (MSP) leases connectivity services over third-party service provider's Layer 2 metro Ethernet networks, or over Layer 3 private or public networks usually seen as part of the macro or small cell deployments.

With a small/femto cell use case, the ACX router acts as a gateway for the overall cell site infrastructure, which includes small/femto cells and LAN switches, to the mobile core network and to the OAM/NM systems of the MSP. A third-party, wholesale MBH network provides Layer 2 or Layer 3 connectivity between MSP cell sites and a centrally located mobile packet core. With regards to ZTD, there are two distinctive factors to be aware of:

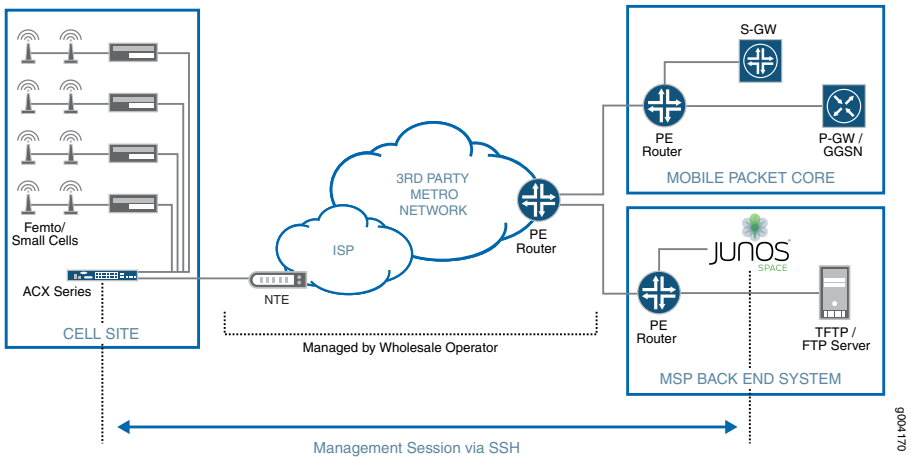
- The cell site typically cannot make an initial connection with the ZTD back-end system of the MSP, thus the typical autoinstallation process cannot be used. In this case, a basic configuration must be loaded onto the ACX router using a USB device, also known

as the one touch deployment method. With the configuration committed on the ACX router, the device can then connect to the ZTD back-end system.

- An untrusted, third-party infrastructure raises more security concerns than when the MSP owns the MBH infrastructure.

Figure 2 on page 8 shows a typical location where an ACX router can be deployed in a wholesale environment.

Figure 2: ACX Router Deployment Locations



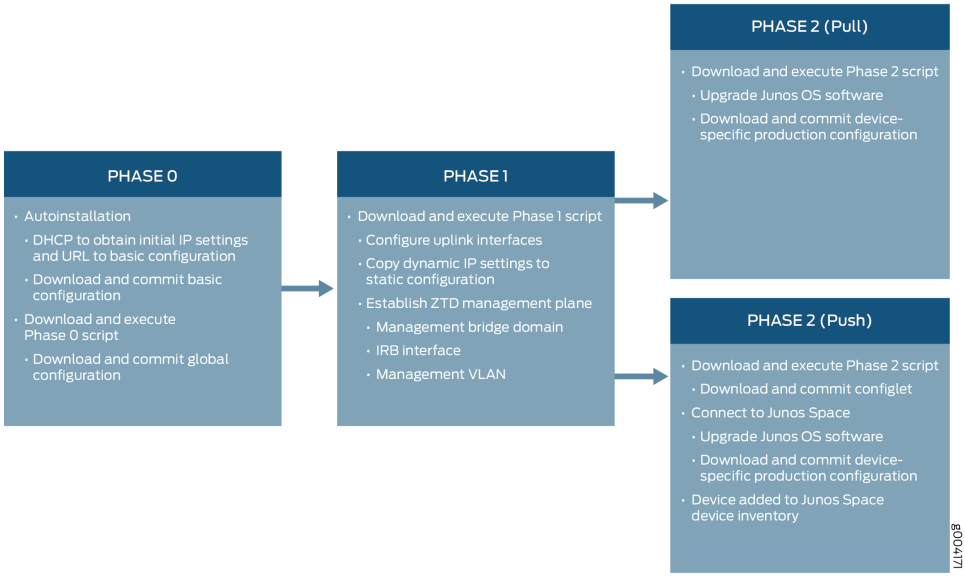

**Related Documentation**

- ACX Autoinstallation Overview on page 8
- Deployment Methods for ACX Routers on page 12
- Example: Deploying ACX Routers Using the ZTD Pull Method on page 21
- Example: Deploying ACX Routers Using the ZTD Push Method on page 49

## ACX Autoinstallation Overview

The zero touch deployment (ZTD) process includes three general phases, as shown in Figure 3 on page 9:

Figure 3: ZTD Phases - Overview




Phase 0, and the overall ZTD process, begins with the autoinstallation function. This essential building block of the overall solution enables automatic configuration of initial network settings on a new ACX router when it is first connected to a network and powered on. The autoinstallation feature is part of the factory default configuration, as shown here:

```
root# show
  version 15.1X54-D20.2;
  system {
      autoinstallation {
          delete-upon-commit; ## Deletes [system autoinstallation] upon
  change/commit
      }
      syslog {
          user * {
              any emergency;
          }
          file messages {
              any error;
              authorization info;
          }
          file interactive-commands {
              interactive-commands any;
          }
      }
      ## Warning: missing mandatory statement(s): 'root-authentication'
```

NOTE: If no network connectivity is established for 5 minutes, a system reboot is required to reinitiate the autoinstallation process.

### Obtaining an IP Address

When an ACX router is powered on for the first time, it sends out DHCP, BOOTP, or RARP requests on all connected interfaces (except the management port) to obtain an IP address.

The ACX router sends a DHCPDISCOVER packet with the following options:

- Vendor class identifier (option 60)—Identifies the type of the platform: ACX500, ACX1100, ACX2100, ACX4000, ACX5048, or ACX5096
- DHCP request (option 53)

The DHCP server responds with the DHCPOFFER with the following details:

- IP-Address (client)
- Router-address (option 3)
- DHCP-Server ip-address (option 54)
- tftp-server ip-address (option 150)
- Boot File Name (option 67; relative path)—The IP address, location, and name of a basic configuration file to download

The ACX router sends a DHCPREQUEST message to take ownership of the offered IP address, and the DHCP Server responds with a DHCPACK, completing the DHCP message exchange.

**Interface Support for Autoinstallation**

Each ACX platform supports autoinstallation on particular interfaces. As of this writing, autoinstallation interface support is as shown in Table 1 on page 10:

### Table 1: Interfaces Supporting Autoinstallation

| Platform | Interfaces Supporting Autoinstallation |
|---|---|
| ACX500 | • ge-0/0/0 through ge-0/0/1<br>• ge-0/1/0 through ge-0/1/3 |
| ACX1000 | • ge-0/1/0 through ge-0/1/7<br>• ge-0/2/0 through ge-0/2/3 |
| ACX1100 | • ge-0/0/0 through ge-0/0/7<br>• ge-0/1/0 through ge-0/1/3 |
| ACX2000 | • ge-0/1/0 through ge-0/1/7<br>• ge-0/2/0 through ge-0/2/1<br>• xe-0/3/0 through xe-0/3/1 |
| ACX2100 | • ge-1/0/0 through ge-1/0/3<br>• ge-1/1/0 through ge-1/1/3<br>• ge-1/2/0 through ge-1/2/1<br>• xe-1/3/0 through xe-1/3/1 |

Table 1: Interfaces Supporting Autoinstallation *(continued)*

| Platform | Interfaces Supporting Autoinstallation |
|---|---|
| ACX2200 | • ge-0/0/0 through ge-0/0/3<br>• ge-0/1/0 through ge-0/1/3<br>• ge-0/2/0 through ge-0/2/1<br>• xe-0/3/0 through xe-0/3/1 |
| ACX4000 | • ge-0/0/0 through ge-0/0/7<br>• ge-0/1/0 through ge-0/1/1<br>• ge-1/0/0 through ge-1/0/5<br>• ge-1/1/0 through ge-1/1/5<br>• xe-0/2/0 through xe-0/2/1 |
| ACX5048 | • xe-0/0/0 through xe-0/0/47 |
| ACX5096 | • xe-0/0/0 through xe-0/0/95 |

**Related Documentation**

- Customer Use Cases for ACX Series Routers on page 5
- Deployment Methods for ACX Routers on page 12
- Example: Deploying ACX Routers Using the ZTD Pull Method on page 21
- Example: Deploying ACX Routers Using the ZTD Push Method on page 49
- ACX Deployment Scripts on page 77

## Deployment Methods for ACX Routers

The zero touch deployment (ZTD) process includes two deployment methods: the Push method and the Pull method. These deployment methods can be used to deploy ACX routers across a wide variety of deployment scenarios.

**Pull Method Overview**

The Pull method involves the ACX router and a centralized configuration server:

- The ACX router uses information obtained during the initial autoinstallation process to connect to the configuration server.
- The configuration server has DHCP, TFTP, and FTP services enabled. It also stores Junos OS software images, configuration files, and scripts that automate the configuration process.
- A basic configuration is loaded onto the ACX router from the configuration server.
- A series of scripts enable the remaining steps of the ZTD process, such as applying a production configuration to the ACX router, upgrading its Junos OS version, and so on.

**Push Method Overview**

The Push method involves the ACX router and a centralized configuration server, with help from the Junos Space Network Management Platform:

- The ACX router uses information obtained during the initial autoinstallation process to connect to the configuration server (which has DHCP, TFTP, and FTP services enabled).
- A basic configuration is loaded onto the ACX router from the configuration server.
  - In scenarios where the ACX router does not have reachability to make the initial connection to the configuration server, the ZTD method is not possible. In these cases, a variation called one touch deployment is available. This method uses a USB drive to load the basic configuration onto the ACX locally. The process then continues as usual.
- The ACX router connects to the Junos Space Platform, which applies a production configuration template, upgrades the Junos OS version, and so on.

## Zero Touch Deployment Using the Pull Method

As noted above, the Pull method of deploying ACX routers does not involve any network management solution or software to manage ACX routers. Most of the intelligence to upgrade the ACX router configuration and Junos OS image is placed on the ACX router itself, with support being provided by the configuration server.

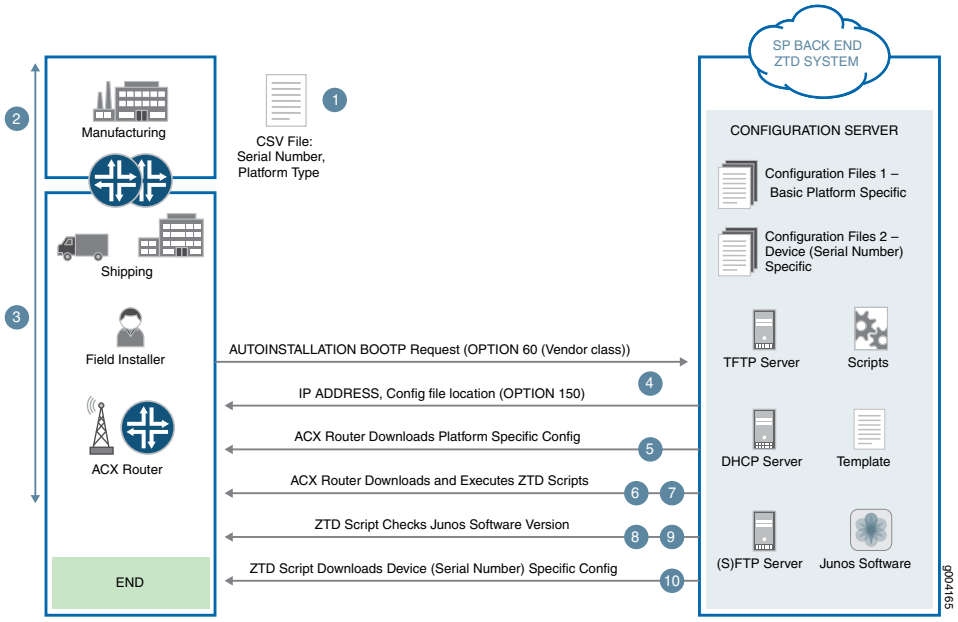
The Pull method is divided into three phases, each with its own dedicated script. The scripts are stored on the configuration server, where the ACX router can download and execute them in sequence.

The three scripts perform the following functions:

- Phase 0—Bootstraps the device and provides a basic configuration, which enables remote management access.
- Phase 1— Establishes a transparent management (OAM) plane by assigning uplink interfaces to a management VLAN. In principle, ZTD can be accomplished more quickly using just a single script (Phase 0) when deploying ACX routers in a star topology. For networks built with a ring or chain topology, this additional phase is required to enable ZTD management by establishing shared VLAN connectivity that spans across the entire access segment. Even before the final configuration is loaded to the node, an established management plane allows the next ACX router in the ring or chain to start its ZTD process.
- Phase 2—Generates a configuration based on inputs and parameters from the global configuration template, and loads a final site-specific configuration, if available

Figure 4 on page 13 describes the steps involved in deploying ACX routers using the Pull method.

### Figure 4: Deploying ACX Routers Using the Pull Method




Zero touch deployment by using the Pull method can be accomplished as follows:

1. The MSP or customer NOC prepares for ZTD deployment by:
   - Making a list of the platforms to be deployed, including serial number and platform type
   - Storing all the necessary elements to the configuration server, including the basic ACX router configuration (by device model, if applicable), global configuration,

production configuration (per ACX device), production version of Junos OS, and the op scripts required to enable ZTD during various stages, at the T/FTP location

2. The manufacturer ships the new device(s) to the customer site(s).
3. The field technician arrives on site, unpacks the ACX router, installs it into a rack, completes the necessary cabling and powers up the device, which triggers the autoinstallation function.
4. As part of the initial DHCP message exchange, the ACX router receives the URL to obtain a basic configuration, which is stored at the TFTP location on the configuration server. By the end of the autoinstallation process, an IP address is assigned and a basic configuration is committed to the ACX router.
5. The basic configuration contains the URL to the ZTD Phase 0 script and the global configuration template, which are stored at the FTP location on the configuration server. Event options in the configuration execute the script and download the global configuration template to the ACX router.
6. When the global configuration is downloaded to the ACX router, the event options for the Phase 0 script are stopped and new event options included in the global configuration take effect.

   The global configuration contains event options and URLs to the ZTD Phase 1 and Phase 2 scripts, as well as a production configuration for the ACX router.
7. Event options in the global configuration template trigger the execution of the ZTD Phase 1 script.

   The Phase 1 script performs following tasks:

   - Configures dedicated uplink interfaces, also known as network-to-network interfaces (NNI) based on the input parameters read from the global configuration template
   - Creates a bridge domain for a management VLAN, to enable a transparent Layer 2 management plane
   - Places logical units of the NNIs inside the management bridge domain
   - Uses DHCP-assigned parameters to create a static IP configuration:
     - IRB management IP address in the same bridge domain to enable management access to the ACX router. The script copies the dynamic IP configuration assigned via the DHCP client configuration and copies it into the IP configuration of the IRB interface and default route. After static IP configuration is completed, the script deletes the configuration for the DHCP client.
     - Default route
     - Unique hostname
     - Creates a configuration for VSTP for the management VLAN (if multiple rings or partially meshed topology is used)
     - Deletes the DHCP client configuration

- Deactivates the event options for ZTD Phase 1
- Activates the event options for ZTD Phase 2

8. ZTD Phase 2 checks the current Junos OS software version on the ACX router and compares it with the version recommended for production. The production version of the Junos OS image is specified in the basic configuration template. If the parameters for a recommended version of Junos OS are not found or set as “inactive”, the ACX router continues with its current version of the Junos OS. If the current Junos OS version is different from the recommended production version, the script downloads the new Junos OS image to the ACX router from the FTP server.
9. The ACX router validates the software image and starts the upgrade procedure. On successful installation, the ACX router reboots.
10. The ACX router connects to the FTP server and checks for a configuration file that matches its serial number (***<serial-number>*.conf**). The router downloads the file and upgrades its configuration to this production configuration.

## Zero Touch Deployment Using the Push Method

As noted earlier, the Push method of deploying ACX routers uses assistance provided by a network management solution. The Junos Space Platform provides rapid deployment support, including device discovery, configuration templates, and OS upgrade capabilities. And once the devices are deployed, Junos Space can then serve as an ongoing central network management solution.

Zero touch deployment using Junos Space can be accomplished using scripts and aggregation routers configured as ZTD helpers to establish connectivity to the configuration server. Figure 5 on page 17 describes the steps involved in deploying ACX routers with the Push method.

Junos Space enables zero touch (and one touch) deployment of ACX routers through the following features:

- Connection profile—A set of connectivity parameters to assign IP addresses to specific NNIs of the ACX routers during the autoconfiguration process. The connection profile can be assigned to the modeled instance so that all device instances use common connectivity parameters to obtain the IP addresses. Junos Space allows you to assign IP addresses through DHCP, PPPoE, PPPoA, and statically (IP addresses are verified with the IP addresses assigned in Junos Space Platform).
- Device configuration deployment using the following features:

- Configuration template—Used to deploy a common production configuration to multiple ACX routers (included in one modeled instance) during the autoconfiguration process. You create a template definition and a configuration template by using the template definition. You can deploy the configuration in the configuration template automatically from Junos Space, or manually through a USB device or a configuration server.
- Quick template—A template created without a template definition can be associated with an instance, and the configuration can be deployed to the ACX router automatically or manually.
- Modify device configuration—Used to create and add an ACX router-specific configuration to the ACX router.
- Review and deploy configuration—Used to approve or reject the configuration changes assigned to the ACX router and deploy the approved configuration.

This NCE uses a configuration template to deploy a configuration to ACX routers.

NOTE: **You must enable the approval-based workflow in the Modify Application Settings page in the Administration workspace to be able to approve or reject the configuration changes on the ACX routers.**

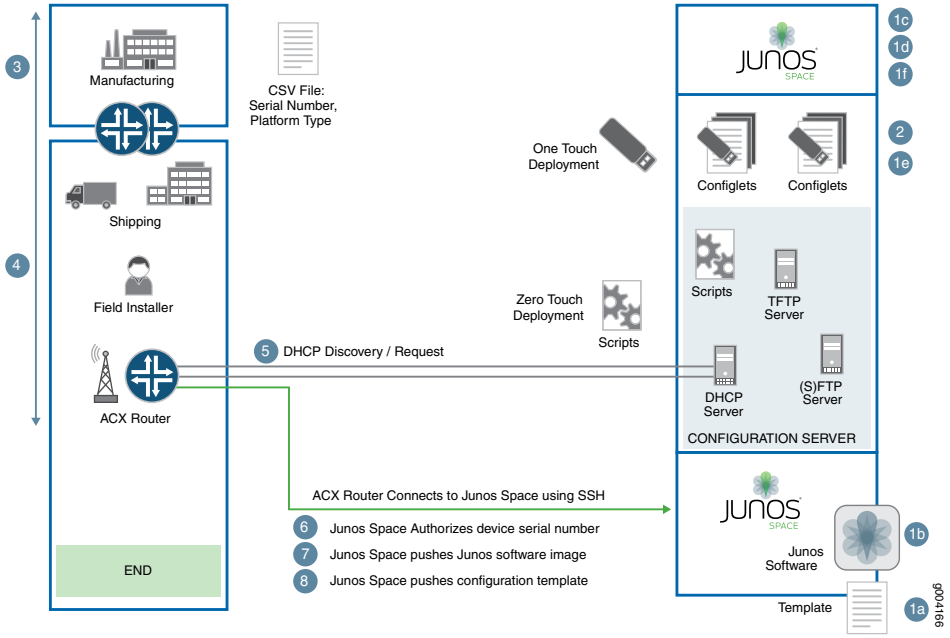
- Junos OS image management—Junos Space can stage and deploy Junos OS images to devices under its management. Using the Images and Scripts workspace, upload multiple Junos OS software images (specific to an ACX router model) to Junos Space. These images are associated to specific device instances when you create a modeled instance, and deployed to the ACX routers during the zero touch deployment process.
- Modeling devices—A collection of device instances (also known as modeled devices) that can be created in the Junos Space database before the ACX routers are discovered. You specify IP addresses (static connection profile), hostnames, Junos OS versions, and serial numbers per device, and assign a connection profile to the modeled instance. You can also assign a device template containing the production configuration to all device instances.
- Device activation—Instances of devices created as part of the modeled instance are activated to associate each device instance to a physical device on your network. This is done through a device-initiated connection by loading a configlet to the device (one touch deployment), or a Junos Space-initiated connection (zero touch deployment). The configlet contains the following details:
  - Outbound SSH configuration to connect to the Junos Space Platform.
  - Authentication details of the ACX router (username and password)
  - IP address assignment details
  - Hostname (if you selected hostname validation)

  Depending on whether the ACX routers must be activated immediately or in the future, modeled devices can be activated when the modeled instance is created or by using the Activate Modeled Device workflow.

- Activating a device at the time of creating a modeled instance—You must load the configlet generated from the modeled instance to the ACX router. On reboot, the ACX router obtains the IP address and connects to Junos Space.
- Activating a device by using the Activate Modeled Device workflow and a device-initiated connection—You must load the configlet generated for the modeled device to the ACX router. On reboot, the ACX router obtains the IP address and connects to Junos Space.
- Activating a device by using the Activate Modeled Device workflow and a Junos Space-initiated connection—You must specify the IP address of the ACX router to initiate a Junos Space-initiated connection to the ACX router. This method is typically used when:
  - The ACX router cannot connect to the Junos Space server after the configlet is loaded to the ACX router and the ACX router has reachability to the Junos Space server.
  - The Junos Space server does not accept inbound SSH connections.

NOTE: During activation, if you selected hostname or serial number validation, the hostname and serial number are validated.

Figure 5: Push Method of Deploying ACX Routers




Zero touch deployment using the Push method and scripts can be accomplished as follows:

1. In Junos Space, perform the following steps:

a. Create device templates containing the production configuration for ACX routers using the Device Templates workspace (**Device Templates > Templates**). We recommend that you validate the configuration on another device before deploying to a production ACX router.

b. Import the desired (recommended) version of the Junos OS using the Images and Scripts workspace.

c. Create a connection profile specifying the mode of IP address assignment to the ACX routers using the Devices workspace (**Devices > Model Devices > Connection Profiles**).

d. Create a modeled instance with the details of purchased and to-be-deployed ACX routers. Modeled instances contain details such as hostname, serial number, ACX Series device model, connection profile, configuration updates (by using the device templates feature), and Junos OS software images. The details such as hostname, serial number, and platform can be uploaded through a CSV file.

e. Generate configlets for the modeled device instance. The contents of the configlet should be copied

- to a configuration file, **space_configlet.conf**, and stored in the FTP server repository, or
- to a USB flash drive to used by the field installer if the one touch deployment model is used.

f. Use one of the following methods to deploy additional configuration to the ACX router instance:

- Create Quick templates and assign the Quick template to the ACX router's instance.
- Modify, review, and deploy the configuration of the ACX router instance.

2. Activate or activate later.

   If you are activating the ACX router through a device-initiated connection, the NOC activates the modeled instance of the ACX router through a device-initiated connection with the Automatic configuration update option selected, and sends the configlet with the initial connection parameters to the field technician to discover the ACX router.

   If you activate the modeled instance of the ACX router, the status moves from the Modeled state to Waiting for Deployment state.

3. The manufacturer ships the new device(s) to the customer site(s).

4. The field technician arrives on site, unpacks the ACX router, installs it into a rack, completes the necessary cabling and powers up the device.

   If using the zero touch deployment (ZTD) method, this triggers the autoinstallation function.

   If using the one touch deployment (OTD) method, the field installer plugs in the USB flash drive containing the Junos Space configlet.

5. For the ZTD method, the ACX router exchanges messages with the DHCP server and receives initial IP settings, as well as the URL to obtain a basic configuration. By the end of the autoinstallation process, an IP address is assigned and a basic configuration is committed to the ACX router. The basic configuration triggers the ZTD Phase 1 script, which downloads and commits a global configuration template. The global configuration then triggers the ZTD Phase 1 script, which configures a management VLAN. The global configuration then triggers the ZTD Phase 2 script, which downloads and commits a configlet. The configlet enables the ACX router to connect to the Junos Space server. Note that for the Push method, the Phase 2 script does not include upgrading the Junos OS or downloading a final configuration to the device. These steps are performed by Junos Space (see next steps).

   For the OTD method, the USB drive provides the configlet, and resulting connectivity to Junos Space.

6. The ACX router connects to the Junos Space server using SSH.

   Once the connection is established, Junos Space authenticates the ACX router credentials (username and password or key-based) and validates the serial number or hostname (as assigned during the modeled instance creation).

   On successful validation, the ACX router is added to the Junos Space database. On the Junos Space **Devices > Device Management** page, the ACX router's state moves from Modeled or Waiting for Deployment to Out of Sync.

7. Junos Space checks the Junos OS version on the ACX router and upgrades or downgrades the image as appropriate (to the version specified in the modeled instance).

8. Junos Space deploys and commits the production configuration (through device templates, quick templates, or modify configuration workflow) to the ACX router.

   When the ACX router reboots and reconnects to Junos Space, the router's state on the **Devices > Device Management** page moves to In Sync if the production configuration is deployed successfully, or to Out of Sync or Connecting or Sync Failed if unsuccessful.

**Related Documentation**

- Customer Use Cases for ACX Series Routers on page 5
- ACX Autoinstallation Overview on page 8
- Example: Deploying ACX Routers Using the ZTD Pull Method on page 21
- Example: Deploying ACX Routers Using the ZTD Push Method on page 49
- ACX Deployment Scripts on page 77

CHAPTER 2

# Configuration Examples



## Example: Deploying ACX Routers Using the ZTD Pull Method

This example demonstrates zero touch deployment (ZTD) for ACX routers using the Pull method.



### Requirements

This example uses the following hardware and software components:

- Three MX240 routers
- Five provisioned ACX2100 routers
- One unboxed ACX2100 router
- Configuration server with DHCP, TFTP, and FTP services enabled
- Junos OS Release 15.1 and later

### Overview

This example describes the tasks required to enable ZTD for new ACX routers in a service provider network while ensuring minimal involvement from the deployment technician and the MSP NOC personnel.

**Preparing Configuration Templates and Scripts for ACX Deployment Phases**

The ZTD process involves a set of configuration files and scripts that are downloaded and executed on provisioned boxes at time of deployment. These scripts and configuration files must be prepared and placed into the TFTP/FTP repository of the configuration server in advance.

The following configuration files are required:

- The **basic configuration** template is downloaded to the bootstrapping node as part of the autoinstallation process. A separate configuration file should be created for each platform—ACX500, ACX1100, ACX2100, ACX4000, ACX5000—and placed into the predefined directory of the TFTP server.

  The basic configuration template includes:

  - System configuration settings, such as administrative access and management protocols
  - Event options to execute the first (Phase 0) op script (stored on the FTP server)
  - URL and name of the global configuration template (stored on the FTP server)
  - URL and name of the image file of the recommended Junos OS version (stored on the FTP server)

- The **global configuration** template contains router configuration common across all access routers, which are sharing the same network roles.

  The global configuration template may include:

  - Minimum settings for administrative access
  - Network services configuration for DNS, FTP, HTTP, NTP, and so on
  - Network management: SNMP, SYSLOG, Netconf, SSHv2
  - Configuration templates for networking protocols
  - Configuration templates for user and core facing interfaces
  - URL and input parameters for the of the ZTD Phase 1 and Phase 2 scripts
  - List of uplink interfaces, also known as network-to-network interfaces (NNIs) that are used to connect the ACX router to the network
  - ID of the management VLAN (referred to in the configuration as the OAM VLAN)
  - Host-name prefix: “csr-“, “an-“ or “agn-“

The following scripts are required:

- **ZTD Phase 0** script execution is triggered by the event option in the basic configuration template. The script downloads and commits the global configuration template to the ACX router.
- **ZTD Phase 1** script execution is triggered by the event option in the global configuration. The script assigns the NNIs to a management VLAN.
- **ZTD Phase 2** script execution is triggered by the event options in the global configuration. The script upgrades the Junos OS version to the recommended version, and downloads and commits the router-specific configuration to the ACX router.

**Aggregation Router Configuration Overview**

New ACX routers generally won't have direct Layer 2 connectivity to the DHCP service of the configuration server. To enable this connectivity, routers in the aggregation segment must be configured prior to starting the ZTD process.

Management access to the access segment is provided by the shared management (OAM) VLAN. At the aggregation routers, the VLAN is terminated into a dedicated virtual-switch instance with a bridge domain.

The VSTP protocol is used to avoid Layer 2 broadcast storms within the management VLAN.

The native VLAN function must be enabled on the access segment-facing interfaces of the aggregation routers, and configured with the VLAN ID of the management VLAN. This configuration allows untagged DHCP frames from the new ACX router to be forwarded to the management bridge domain and on to the configuration server.

An integrated routing and bridging (IRB) interface on each aggregation router provides connectivity between the Layer 2 management bridge domain and the Layer 3 domain of the management network.

Finally a DHCP relay function must be enabled on top of the IRB interface. The IP address assigned to the IRB interface serves as a default gateway for the management network of the ACX routers. The IP address also indicates to the DHCP server which address range will be used to assign management addresses to the ACX router in the given access segments.

#### DHCP Configuration Overview

You can use any standard DHCP server that meets the requirements specified by RFC 2131. The DHCP server must be able to manage address pools and ranges per ACX router platform or class (DHCP option 60).

The DHCP server configuration elements used in this example include the following:

- Classes created to match the values of option 60 (vendor class identifier) sent in the DHCPDISCOVER message from the client
  - This configuration example is applicable to ACX2100, ACX4000, and ACX5096 routers. A fourth class, CSR, is included to match DISCOVERY requests received with option 60 set to CSR-acx.
- URL to the basic configuration file that is loaded during Phase 0 of the ZTD process
- Address pools per access segment
  - This example uses two address pools created for ACCESS SEGMENT 1 and ACCESS SEGMENT 10.
- Address ranges associated with the device classes.

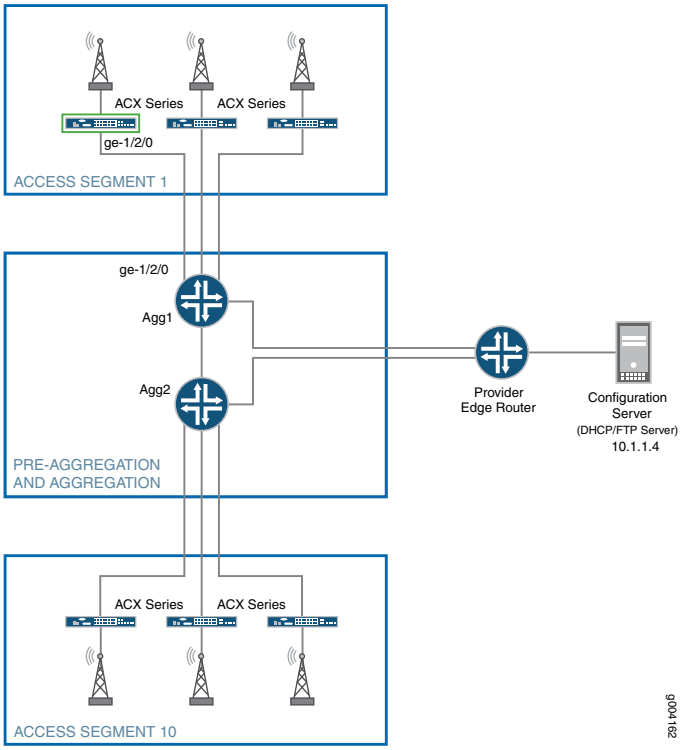
 

- This example includes two address ranges per address pool. Once the ACX router downloads and commits the basic configuration, the device sends a new DHCP request with option 60 set to CSR-acx. A new IP address is then assigned to the ACX router that puts it in the management VLAN.

## Topology

Figure 6 on page 24 depicts a topology for zero touch deployment of an ACX router using the Pull method. The topology has three ACX routers in each access segment. The new ACX router being deployed in ACCESS SEGMENT 1 has its interface ge-1/2/0 connected to interface ge-1/2/0 on router Agg1 in the aggregation segment. Aggregation router Agg1 has an interface is connected to the provider edge router. The configuration server is also connected to the provider edge router. It acts as a DHCP, TFTP, and FTP server, and hosts the scripts, configuration files, and Junos OS images required to deploy the new ACX router.

### Figure 6: Topology Diagram for Zero Touch Deployment Using the Pull Method




NOTE: The configurations in this example include deployment-specific parameters such as root authentication, FTP URLs, server IP address, Junos OS version, and so on. Adjust these variables to suit your network environment.

## Configuration

To deploy ACX routers by using the zero touch deployment Push method, perform these tasks:



### Creating the Basic Configuration File for the New ACX Router

**Step-by-Step Procedure**

To create a basic configuration template for the new ACX 2100 router:

1. Copy the following configuration to a text editor.

```
/*-------------------------------------------*/
/*  THIS IS AN EXAMPLE OF THE BASIC ZTP      */
/*  CONFIGURATION WHICH WOULD BE USED WITH   */
/*  ACX2100 SERIES ROUTERS                   */
/*-------------------------------------------*/
groups {
    global {
        system {
            debugger-on-panic;
            debugger-on-break;
            dump-on-panic;
            authentication-order [ password radius ];
            root-authentication {
                encrypted-password "$ABC123"; ## SECRET-DATA
            }
            login {
                user user {
                    uid 928;
                    class superuser;
                    shell csh;
                    authentication {
                        encrypted-password "$ABC123"; ## SECRET-DATA
                    }
                }
            }
            services {
                finger;
                ftp;
                rlogin;
                rsh;
                ssh;
                telnet;
                xnm-clear-text;
                netconf {
                    ssh;
                }
            }
        }
    }
    GR-ZTP {
```

```
        apply-macro CONFIG {
            CONFIG
"ftp://ztdadmin:jnpr1234@10.1.1.4://config/acx_global_config.conf";
        }
        apply-macro CODE {
            VERSION "15.1X54-D25";
            INACTIVE "active";
            IMAGE
"ftp://ztdadmin:jnpr1234@10.1.1.4://build/jinstall-ppc-15.1X54-D25-domestic-signed.tgz";

        }
    }
    GR-ZTP-STAGE-0 {
        event-options {
            generate-event {
                ztp_script time-interval 60;
            }
            policy ztp_script {
                events ztp_script;
                then {
                    execute-commands {
                        commands {
                            "op url
ftp://ztdadmin:jnpr1234@10.1.1.4://scripts/ztp_script_0_basic.slax";
                        }
                    }
                }
            }
        }
    }
}
apply-groups [ global GR-ZTP-STAGE-0 ];
interfaces {
    ge-1/2/0 {
        unit 0 {
            family inet {
                dhcp-client {
                    vendor-id CSR-acx;
                }
            }
        }
    }
}
```

2. Save the configuration as **acx2100.conf**.

## Creating the Global Configuration File for the New ACX Router

**Step-by-Step Procedure**

To create a global configuration template for the new ACX 2100 router:

1. Copy the following configuration to a text editor.

```
groups {
    GR-ZTP-SCENARIOS {
        apply-macro SCENARIO-1 {
/*-------------------------------------------------------------------------*/
/*  METHOD: Defines type of the model is used:                             */
/*     - push - enables push model, skips sw upgrades and fetches for */
```

```
/*           configlet file generated by Space NMS at FTP location    */
/*     - pull - enables pull model - upgrades junos sw, fetches the   */
/*                                                          <S/N>.conf */
/*     - If not specified, pull method will be used by default        */
/*  STAGE 0: Assigns DHCP address to CSR in OAM VLAN (mandatory)      */
/*  STAGE 1: Enables Static IP connectivity (mandatory):              */
/*     - Enables Static IP connectivity with OAM VLAN                 */
/*     - Enables Global/(VLAN 1) OSPF IP connectivity                 */
/*     - Creates host-name                                            */
/*     - Loads box-specific config for outbound management (lab only) */
/*  STAGE 2: Upgrades Junos image  (optional)                         */
/*        Loads box specific configuration/template                   */
/*                          or                                        */
/*        Loads platform specific configuration/configlet             */
/*        to initiate a call to the Home Space NMS                    */
/*        (a combined "pull" and "push" model)                        */
/*  To manage ZTP scenarios work flow you can use following           */
/*  values to process stages:                                         */
/*   enabled   (enables ZTP stage)                                    */
/*   disabled  (disables/skip ZTP stage)                              */
/*   stop      (disables ZTP stage and stop ZTP process at this point)*/
/*-------------------------------------------------------------------*/

            STAGE-0 enabled;
            METHOD "pull";
            STAGE-1 enabled;
            STAGE-2 stop;
        }
    }
    GR-ZTP-CALLHOME {
        apply-macro CALLHOME {
            CONFIG "ftp://ztdadmin:jnpr1234@10.1.1.4://config";
            SPACE-CONFIGLET "space_configlet.conf";
        }
    }
    GR-ZTP-PLATFORM {
        apply-macro ZTP-acx2100 {
            NNI1 ge-1/2/0;
            NNI2 ge-1/2/1;
            HOST_NAME csr-;
            OAM_VLAN 2;
        }
        apply-macro ZTP-acx4000 {
             NNI1 ge-0/1/0;
             NNI2 ge-0/1/1;
             NNI3 ge-0/0/1;
             HOST_NAME csr-;
             OAM_VLAN 2;
       }
       apply-macro ZTP-acx5096 {
             NNI1 xe-0/0/0;
             NNI2 xe-0/0/2;
             NNI3 xe-0/0/46;
             HOST_NAME agg-;
             OAM_VLAN 2;
        }
    }
    GR-ZTP-STAGE-1 {
        event-options {
            generate-event {
                ztp_stage_1 time-interval 60;
```

```
            }
            policy ztp_script {
                events ztp_stage_1;
                then {
                    execute-commands {
                        commands {
                            "op url
ftp://ztdadmin:jnpr1234@10.1.1.4://scripts/ztp_script_1_oam.slax";
                        }
                    }
                }
            }
        }
    }
    GR-ZTP-STAGE-2 {
        event-options {
            generate-event {
                ztp_stage_2 time-interval 60;
            }
            policy ztp_script {
                events ztp_stage_2;
                then {
                    execute-commands {
                        commands {
                            "op url
ftp://ztdadmin:jnpr1234@10.1.1.4://scripts/ztp_script_2.slax";
                        }
                    }
                }
            }
        }
    }
    global {
        system {
            domain-name lab.example.com;
            time-zone America/Los_Angeles;
            debugger-on-panic;
            debugger-on-break;
            dump-on-panic;
            authentication-order [ password radius ];
            root-authentication {
                encrypted-password "$ABC123"; ## SECRET-DATA
            }
            name-server {
                192.168.5.68;
                192.168.60.131;
            }
            radius-server {
                192.168.69.162 secret "$ABC123"; ## SECRET-DATA
            }
            login {
                user user {
                    uid 928;
                    class superuser;
                    shell csh;
                    authentication {
                        encrypted-password "$ABC123"; ## SECRET-DATA
                    }
                }
            }
            services {
```

```
            finger;
            ftp;
            rlogin;
            rsh;
            ssh;
            telnet;
            xnm-clear-text;
            netconf {
                ssh;
            }
        }
        syslog {
            host log {
                kernel info;
                any notice;
                pfe info;
                interactive-commands any;
            }
            file messages {
                kernel info;
                any notice;
                authorization info;
                pfe info;
                archive world-readable;
            }
            file security {
                interactive-commands any;
                archive world-readable;
            }
        }
        processes {
            routing enable;
            ntp enable;
            management enable;
            watchdog enable;
            snmp enable;
            inet-process enable;
            mib-process enable;
        }
        ntp {
            boot-server 172.17.28.5;
            server 172.17.28.5;
        }
    }
    chassis {
        dump-on-panic;
    }
    interfaces {
        lo0 {
            unit 0 {
                family inet {
                    address 127.0.0.1/32;
                }
            }
        }
    }
    snmp {
        location "Solution lab";
        community public {
            authorization read-only;
        }
```

```
            community private {
                authorization read-write;
            }
        }
    }
    GR-CSR-ACCESS-INTF {
        interfaces {
            <*> {
                traps;
                mtu 9192;
            }
        }
    }
    GR-NNI-TAG {
        interfaces {
            "<[g|x]e-*>" {
                traps;
                mtu 9182;
                hold-time up 5000 down 0;
            }
        }
    }
}
```

2. Save the configuration as **acx_global_config.conf**.

## Configuring DHCP, TFTP, and FTP Services on the Configuration Server

**Step-by-Step Procedure**

To create a enable DHCP, TFTP, and FTP services on a Linux-based configuration server:

1. Copy the following configuration to a text editor:

```
# DHCP server sample configuration (dhcpd.conf)

ddns-update-style none;
default-lease-time 600;
max-lease-time 7200;
if exists dhcp-parameter-request-list {
    option dhcp-parameter-request-list=concat(option
dhcp-parameter-request-list,96);
}

# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.

authoritative;

# Use this to send dhcp log messages to a different log file (you also
# have to hack syslog.conf to complete the redirection).

log-facility local7;

#-------------------------------------------------------#
#                Main ZTD Configuration                 #
#-------------------------------------------------------#

option option-150 code 150 = ip-address;
class "acx2100" {
```

```
        match if (substring (option vendor-class-identifier,0, 15) =
"Juniper-acx2100");
             option bootfile-name "/boot-config/acx2100.conf";
        }
class "acx4000" {
        match if (substring (option vendor-class-identifier,0, 15) =
"Juniper-acx4000");
             option bootfile-name "/boot-config/acx4000.conf";
        }
class "acx5096" {
        match if (substring (option vendor-class-identifier,0, 15) =
"Juniper-acx5096");
             option bootfile-name "/boot-config/acx5096.conf";
        }
class "CSR" {
        match if (substring (option vendor-class-identifier,0, 15) =
"CSR-acx");
        }
class "unknown" {
        match if ((substring (option vendor-class-identifier,0, 15) !=
"Juniper-acx2100")
              and(substring (option vendor-class-identifier,0, 15) !=
"Juniper-acx1100")
              and(substring (option vendor-class-identifier,0, 15) !=
"Juniper-acx5096")
              and(substring (option vendor-class-identifier,0, 15) !=
"CSR-acx"));
        }

# LOCAL SUBNET. REQUIRED TO MAKE DHCP SERVER RUN AT LOCAL NTERFACE
# BUT NOT USED AS A PART OF THE ZTP PROCESS
subnet 10.1.1.0 netmask 255.255.255.0 {
       option routers 10.1.1.2;
       option broadcast-address 10.1.1.255;
       default-lease-time 300;
       max-lease-time 600;
       range 10.1.1.249 10.1.1.253;
       option subnet-mask 255.255.255.0;
}

# ZTP SUBNET for ACCESS SEGMENT 1
# The first pool of the range will be used to bootstrap devices (ztp stage
 0)
# The second pool of the range will be used assign an IP address to the
CSR (cell site / ACX router) with basic configuration file loaded (ztp
stage 1)

subnet 192.168.1.0 netmask 255.255.255.0 {
       option domain-name "area.1.1.lo.1.0.net";
       option routers 192.168.1.251;
       option broadcast-address 192.168.1.255;
       default-lease-time 300;
       max-lease-time 600;
       pool {
           range 192.168.1.220 192.168.1.249;
           allow members of "acx4000";
           allow members of "acx5096";
           allow members of "acx2100";
           option option-150 10.1.1.4;
           option subnet-mask 255.255.255.0;
           option dhcp-option-overload 3;
```

```
      }
       pool {
            range 192.168.1.2 192.168.1.219;
            allow members of "unknown";
            allow members of "CSR";
            option option-150 10.1.1.4;
            option subnet-mask 255.255.255.0;
            option dhcp-option-overload 3;
            default-lease-time 1250000;
            max-lease-time 2500000;
      }

}
# ZTP SUBNET for ACCESS SEGMENT 10
# The first pool of the range will be used to bootstrap devices (ztp stage
 0)
# The second pool of the range will be used assign an IP address to the
CSR (cell site / ACX router) with basic configuration file loaded (ztp
stage 1)

subnet 192.168.10.0 netmask 255.255.255.0 {
       option domain-name "area.10.1.lo.1.0.net";
       option routers 192.168.10.252;
       option broadcast-address 192.168.10.255;
       default-lease-time 300;
       max-lease-time 600;
       pool {
            range 192.168.10.220 192.168.10.249;
            allow members of "acx4000";
            allow members of "acx5096";
            allow members of "acx2100";
            option option-150 10.1.1.4;
            option subnet-mask 255.255.255.0;
            option dhcp-option-overload 3;
    }
       pool {
            range 192.168.10.1 192.168.10.219;
            allow members of "unknown";
            allow members of "CSR";
            option option-150 10.1.1.4;
            option subnet-mask 255.255.255.0;
            option dhcp-option-overload 3;
            default-lease-time 1250000;
            max-lease-time 2500000;
    }

}
```

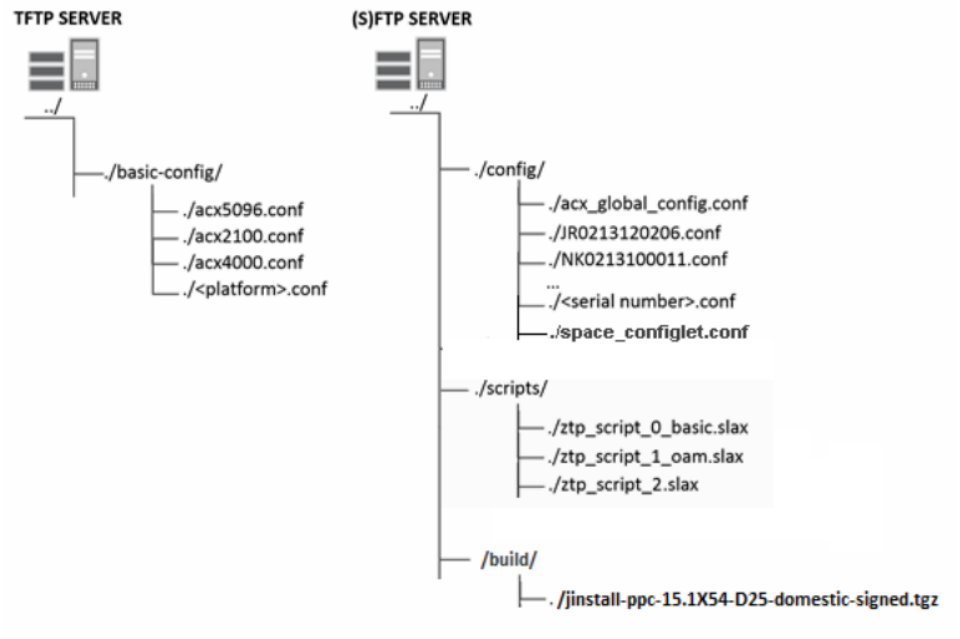
2. Save the file as **dhcpd.conf**, in the **/etc/dhcp/** directory on the Linux system.

3. Enable TFTP and FTP server functionality on the configuration server. Use the username *ztdadmin*.

4. Setup TFTP and FTP directory structures and save the basic configuration files, global configuration file, router-specific configuration files, Junos OS images, and ZTD scripts, as shown in Figure 7 on page 33.

Figure 7: TFTP and FTP Directory Structure




NOTE: The ZTD scripts can be found in the section "ACX Deployment Scripts" on page 77.

## Configuring Aggregation Routers

**CLI Quick Configuration**

To quickly configure aggregation routers Agg1 and Agg2 for the ZTD process, copy the following commands, remove any line breaks, and then paste the commands into the CLI.

**Router Agg1**

```
[edit]
set interfaces ge-1/2/0 apply-groups GR-AGG-INTF-TAG
set interfaces ge-1/2/0 flexible-vlan-tagging
set interfaces ge-1/2/0 native-vlan-id 2
set interfaces ge-1/2/0 encapsulation flexible-ethernet-services
set interfaces ge-1/2/0 unit 2 encapsulation vlan-bridge
set interfaces ge-1/2/0 unit 2 vlan-id 2
set interfaces ge-1/2/0 unit 2 family bridge
set interfaces irb unit 1 family inet address 192.168.1.251/24 vrrp-group 1
virtual-address 192.168.1.250
set interfaces irb unit 1 family inet address 192.168.1.251/24 vrrp-group 1
priority 254
## VRRP config above included as reference for multi-homed scenarios
set forwarding-options dhcp-relay server-group ZTD-DHCP 10.1.1.4
set forwarding-options dhcp-relay active-server-group ZTD-DHCP
```

```
set forwarding-options dhcp-relay group ZTD-DHCP interface irb.1
set routing-instances VSI-ZTP-1 instance-type virtual-switch
set routing-instances VSI-ZTP-1 protocols vstp interface ge-1/2/0
set routing-instances VSI-ZTP-1 protocols vstp vlan 2
set routing-instances VSI-ZTP-1 bridge-domains VLAN-OAM vlan-id 2
set routing-instances VSI-ZTP-1 bridge-domains VLAN-OAM interface ge-1/2/0.2
set routing-instances VSI-ZTP-1 bridge-domains VLAN-OAM routing-interface
irb.1
```

**Router Agg2**

NOTE: This configuration is included for reference only. This configuration example does not require router Agg2.

```
[edit]
set interfaces ge-1/3/0 apply-groups GR-AGG-INTF-TAG
set interfaces ge-1/3/0 flexible-vlan-tagging
set interfaces ge-1/3/0 native-vlan-id 2
set interfaces ge-1/3/0 encapsulation flexible-ethernet-services
set interfaces ge-1/3/0 unit 2 encapsulation vlan-bridge
set interfaces ge-1/3/0 unit 2 vlan-id 2
set interfaces ge-1/3/0 unit 2 family bridge
set interfaces irb unit 10 family inet address 192.168.10.252/24 vrrp-group
10 virtual-address 192.168.10.250
set interfaces irb unit 10 family inet address 192.168.10.252/24 vrrp-group
10 priority 254
## VRRP config above included as reference for multi-homed scenarios
set forwarding-options dhcp-relay server-group ZTD-DHCP 10.1.1..4
set forwarding-options dhcp-relay active-server-group ZTD-DHCP
set forwarding-options dhcp-relay group ZTD-DHCP interface irb.10
set routing-instances VSI-ZTP-10 instance-type virtual-switch
set routing-instances VSI-ZTP-10 protocols vstp interface ge-1/3/0
set routing-instances VSI-ZTP-10 protocols vstp vlan 2
set routing-instances VSI-ZTP-10 bridge-domains VLAN-OAM vlan-id 2
set routing-instances VSI-ZTP-10 bridge-domains VLAN-OAM interface ge-1/3/0.2
set routing-instances VSI-ZTP-10 bridge-domains VLAN-OAM routing-interface
irb.10
```

**Step-by-Step Procedure**

To configure the aggregation routers:

NOTE: For brevity, only aggregation router Agg1 is shown here.

1. Create the access-facing interface toward the new ACX router.

```
user@Agg1# set interfaces ge-1/2/0 apply-groups GR-AGG-INTF-TAG
user@Agg1# set interfaces ge-1/2/0 flexible-vlan-tagging
user@Agg1# set interfaces ge-1/2/0 native-vlan-id 2
user@Agg1# set interfaces ge-1/2/0 encapsulation flexible-ethernet-services
user@Agg1# set interfaces ge-1/2/0 unit 2 encapsulation vlan-bridge
user@Agg1# set interfaces ge-1/2/0 unit 2 vlan-id 2
user@Agg1# set interfaces ge-1/2/0 unit 2 family bridge
```

2. Create the IRB interface to act as the ACX router's default gateway.

```
user@Agg1# set interfaces irb unit 1 family inet address 192.168.1.251/24 vrrp-group
  1 virtual-address 192.168.1.250
user@Agg1# set interfaces irb unit 1 family inet address 192.168.1.251/24 vrrp-group
  1 priority 254
```

NOTE: This configuration example uses a stand-alone aggregation device. The VRRP configuration elements are included as a reference for cases where dual-homed connectivity is desired for the access devices. If your environment uses this setup, ensure that the router-option value in the DHCP server configuration uses the virtual IP address.

3. Create a bridge domain.

```
user@Agg1# set routing-instances VSI-ZTP-1 instance-type virtual-switch
user@Agg1# set routing-instances VSI-ZTP-1 protocols vstp interface ge-1/2/0
user@Agg1# set routing-instances VSI-ZTP-1 protocols vstp vlan 2
user@Agg1# set routing-instances VSI-ZTP-1 bridge-domains VLAN-OAM vlan-id
  2
user@Agg1# set routing-instances VSI-ZTP-1 bridge-domains VLAN-OAM interface
  ge-1/2/0.2
user@Agg1# set routing-instances VSI-ZTP-1 bridge-domains VLAN-OAM
  routing-interface irb.1
```

4. Create the DHCP relay settings to forward the ACX router's DHCP requests to the DHCP (configuration) server.

```
user@Agg1# set forwarding-options dhcp-relay server-group ZTD-DHCP 10.1.1.4
user@Agg1# set forwarding-options dhcp-relay active-server-group ZTD-DHCP
user@Agg1# set forwarding-options dhcp-relay group ZTD-DHCP interface irb.1
```

## Deploying the ACX Router

**Step-by-Step Procedure**

To deploy the ACX router:

1. Unpack the ACX router, power it on, and connect the designated interface to the network.

   For this example, the new the ACX2100 router uses interface ge-1/2/0.

   Once the router boots up, the autoinstallation function of the ACX router communicates with the DHCP server, acquires an initial IP address, downloads and commits the basic configuration template, and proceeds to use the ZTD scripts to setup the device, as follows:

   - When the basic configuration is committed to the ACX router, the event options in the basic configuration trigger the execution of the ZTD Phase 0 script. This script downloads and commits the global configuration.

- When the global configuration is committed to the ACX router, the event options in the global configuration trigger the execution of the ZTD Phase 1 script. The ZTD Phase 1 script assigns the NNIs to a management (OAM) VLAN.
- The global configuration also triggers the execution of the ZTD Phase 2 script. The ZTD Phase 2 script validates and upgrades the Junos OS to the recommended version, and commits the router-specific configuration to the ACX router.

NOTE: The ZTD scripts can be found in the section "ACX Deployment Scripts" on page 77.

2. Verify that the ACX router has its full and proper configuration using the verification steps in the next section.

## Verification

Use the following procedures to verify the deployment of ACX routers using the Pull method:

NOTE: To verify the basic configuration, global configuration, configuration through scripts, and Junos OS upgrades, it is helpful to be connected to a system log server to receive notifications about the execution of scripts, configuration downloads, and so on, as they happen. As you receive appropriate notifications, you can then further verify the configuration downloads and Junos OS upgrades by establishing an SSH connection to the ACX router and using the verification steps below.



### Verifying the Progress of the ZTD Process

**Purpose** Verify the progress of the ZTD process and verify the basic configuration, global configuration, configuration through scripts, and Junos OS upgrades.

NOTE: This step can be used in conjunction with the appropriate verification procedures below, depending on the stage of the ZTD process.

The **monitor start** command allows you to view entries being added to a log file in real-time. In this example, the ZTD process is logged to the file **op-script.log**. You can use this method to monitor the ZTD process as it is happening.

Immediately after the basic configuration is applied, you should be able to gain access to the ACX router using SSH and the initial IP address provided by DHCP server. Note that as the process moves forward, your session will be broken a few times, as the initial IP address is reconfigured to a permanent IP address, and the router reboots to complete its OS upgrade.

If the process has an issue at any point, you will be able to review the log file generated by script and identify what went wrong.

NOTE: While not typical, if you happen to have console access to the ACX router, you can use this method to monitor the process and read ZTD log file in real-time, without interruption.

**Action** SSH to the ACX router. Enter configuration mode and execute the **monitor start op-script.log** command.

```
user# monitor start op-script.log
Jun 26 09:32:55 op script processing begins
Jun 26 09:32:55 reading op script input details
Jun 26 09:32:55 testing op details
Jun 26 09:32:55 running op script
'/var/tmp/tmp_opRmTq8z/...transferring.file.........cgcuaW/ztp_script_0_basic.slax'
Jun 26 09:32:55 opening op script
'/var/tmp/tmp_opRmTq8z/...transferring.file.........cgcuaW/ztp_script_0_basic.slax'
Jun 26 09:32:55 reading op script
'/var/tmp/tmp_opRmTq8z/...transferring.file.........cgcuaW/ztp_script_0_basic.slax'
Jun 26 09:32:59 ZTP Phase 0 SCRIPT: ZTP phase 0 BEGIN
Jun 26 09:32:59 ZTP Phase 0 SCRIPT: Found a group GR-ZTP
Jun 26 09:32:59 ZTP Phase 0 SCRIPT: Looking for apply-macro CONFIG in group GR-ZTP
Jun 26 09:32:59 ZTP Phase 0 SCRIPT: Found apply-macro CONFIG
Jun 26 09:32:59 ZTP Phase 0 SCRIPT: Config file:
ftp://ztdadmin:jnpr1234@10.1.1.4://config/ztd_global.conf will be loaded and
merged to existed configuration
Jun 26 09:32:59 ZTP Phase 0 SCRIPT: Got config lock
Jun 26 09:32:59 ZTP Phase 0 SCRIPT: Processing config url
ftp://ztdadmin:jnpr1234@10.1.1.4://config/ztd_global.conf
Jun 26 09:33:00 ZTP Phase 0 SCRIPT: Configuration was loaded
Jun 26 09:33:15 ZTP Phase 0 SCRIPT: Configuration was committed
Jun 26 09:33:15 ZTP Phase 0 SCRIPT: release config lock
Jun 26 09:33:15 ZTP Phase 0 SCRIPT: ZTP Global configuration template loaded
Jun 26 09:33:15 ZTP Phase 0 SCRIPT: Found a group GR-ZTP-STAGE-1
Jun 26 02:33:28 init: can not access /usr/sbin/relayd: No such file or directory
```

```
Jun 26 02:33:28 init: relay-process (PID 0) started
Jun 26 09:33:29 ZTP Phase 0 SCRIPT: ZTP Phase 0 completed SUCCESSFULLY
Jun 26 09:33:29 op script output
Jun 26 09:33:29 begin dump
Jun 26 09:33:29 end dump
Jun 26 09:33:29 inspecting op output
'/var/tmp/tmp_opRmTq8z/...transferring.file.........cgcuaW/ztp_script_0_basic.slax'
Jun 26 09:33:29 finished op script
'/var/tmp/tmp_opRmTq8z/...transferring.file.........cgcuaW/ztp_script_0_basic.slax'
Jun 26 09:33:29 op script processing ends
Jun 26 02:33:51 op script processing begins
Jun 26 02:33:51 reading op script input details
Jun 26 02:33:51 testing op details
Jun 26 02:33:51 running op script
'/var/tmp/tmp_opWFU5xI/...transferring.file.........Xz8Rfd/ztp_script_1_oam.slax'
Jun 26 02:33:51 opening op script
'/var/tmp/tmp_opWFU5xI/...transferring.file.........Xz8Rfd/ztp_script_1_oam.slax'
Jun 26 02:33:51 reading op script
'/var/tmp/tmp_opWFU5xI/...transferring.file.........Xz8Rfd/ztp_script_1_oam.slax'
Jun 26 02:33:55 ZTP Phase 1 SCRIPT: ZTP Phase 1 BEGIN
Jun 26 02:33:55 ZTP Phase 1 SCRIPT: Deactivating event-options for ZTP Phase 1
script...
Jun 26 02:34:08 ZTP Phase 1 SCRIPT: Configuration for event-options for ZTP Phase
 1 was disabled
Jun 26 02:34:08 ZTP Phase 1 SCRIPT: Getting box Serial Number
Jun 26 02:34:08 ZTP Phase 1 SCRIPT: serial no = NK021310xxxx
Jun 26 02:34:08 ZTP Phase 1 SCRIPT: Retrieve type of platform
Jun 26 02:34:08 ZTP Phase 1 SCRIPT: platform = acx2100
Jun 26 02:34:08 ZTP Phase 1 SCRIPT: Looking for acx2100 specific parameters.
Jun 26 02:34:08 ZTP Phase 1 SCRIPT: Found a group GR-ZTP-PLATFORM
Jun 26 02:34:08 ZTP Phase 1 SCRIPT: Looking for apply-macro ZTP-acx2100 in group
 GR-ZTP-PLATFORM
Jun 26 02:34:08 ZTP Phase 1 SCRIPT: Found apply-macro ZTP-acx2100
Jun 26 02:34:08 ZTP Phase 1 SCRIPT: Retrieve DHCP client bindings
Jun 26 02:34:09 ZTP Phase 1 SCRIPT: FOUND HOST PREFIX: csr-
Jun 26 02:34:09 ZTP Phase 1 SCRIPT: FOUND VLAN: 2
Jun 26 02:34:09 ZTP Phase 1 SCRIPT: DHCP client runs on ZTP interface: ge-1/2/0.0
Jun 26 02:34:09 ZTP Phase 1 SCRIPT: Found DHCP Option for router: [ 192.168.1.250
 ]
Jun 26 02:34:09 ZTP Phase 1 SCRIPT: Derived next-hop: 192.168.1.250
Jun 26 02:34:09 ZTP Phase 1 SCRIPT: DHCP ip address found: 192.168.1.2/24
Jun 26 02:34:09 ZTP Phase 1 SCRIPT: Parsing ZTP Interface: ifd=ge-1/2/0 unit=0
Jun 26 02:34:09 ZTP Phase 1 SCRIPT: Configuring NNI interface: ge-1/2/0
Jun 26 02:34:09 ZTP Phase 1 SCRIPT: Configuring NNI interface: ge-1/2/1
Jun 26 02:34:09 ZTP Phase 1 SCRIPT: Configuring bridge-domain: BD-ZTP-OAM
Jun 26 02:34:22 init: dhcp-service (PID 9053) terminate signal sent
Jun 26 02:34:22 init: general-authentication-service (PID 9054) terminate signal
 sent
Jun 26 02:34:22 init: can not access /usr/sbin/relayd: No such file or directory
Jun 26 02:34:22 init: relay-process (PID 0) started
Jun 26 02:34:22 init: general-authentication-service (PID 9054) exited with
status=0 Normal Exit
Jun 26 02:34:22 init: dhcp-service (PID 9053) exited with status=0 Normal Exit
Jun 26 02:34:23 ZTP Phase 1 SCRIPT: Platform specific configuration committed
successfully.
Jun 26 02:34:23 ZTP Phase 1 SCRIPT: Looking for box (S/N: NK021310xxxx) specific
 parameters.
Jun 26 02:34:24 ZTP Phase 1 SCRIPT: Found a group GR-ZTP-BOX
Jun 26 02:34:24 ZTP Phase 1 SCRIPT: Looking for apply-macro NK021310xxxx in group
 GR-ZTP-BOX
Jun 26 02:34:24 ZTP Phase 1 SCRIPT: Can't find apply-macro NK021310xxxx
```

```
Jun 26 02:34:24 ZTP Phase 1 SCRIPT: Box specific configuration for S/N:
NK02131Oxxxx was not found in global template. Skip this step.
Jun 26 02:34:24 ZTP Phase 1 SCRIPT: Found a group GR-ZTP-BOX
Jun 26 02:34:24 ZTP Phase 1 SCRIPT: Deleting configuration group GR-ZTP-BOX
Jun 26 02:34:24 ZTP Phase 1 SCRIPT: Found a group GR-ZTP-STAGE-2
Jun 26 02:34:38 ZTP Phase 1 SCRIPT: Configuration was committed
Jun 26 02:34:38 ZTP Phase 1 SCRIPT: ZTP Phase 1 completed successfully
Jun 26 02:34:38 op script output
Jun 26 02:34:38 begin dump
Jun 26 02:34:38 end dump
Jun 26 02:34:38 inspecting op output
'/var/tmp/tmp_opWFU5xI/...transferring.file.........Xz8Rfd/ztp_script_1_oam.slax'
Jun 26 02:34:38 finished op script
'/var/tmp/tmp_opWFU5xI/...transferring.file.........Xz8Rfd/ztp_script_1_oam.slax'
Jun 26 02:34:38 op script processing ends
Jun 26 02:34:56 op script processing begins
Jun 26 02:34:56 reading op script input details
Jun 26 02:34:56 testing op details
Jun 26 02:34:56 running op script
'/var/tmp/tmp_opMsRyrF/...transferring.file.........PQHCP5/ztp_script_2.slax'
Jun 26 02:34:56 opening op script
'/var/tmp/tmp_opMsRyrF/...transferring.file.........PQHCP5/ztp_script_2.slax'
Jun 26 02:34:56 reading op script
'/var/tmp/tmp_opMsRyrF/...transferring.file.........PQHCP5/ztp_script_2.slax'
Jun 26 02:34:59 ZTP Phase 2 SCRIPT: ZTP Phase 2 BEGIN
Jun 26 02:34:59 ZTP Phase 2 SCRIPT: Found a group GR-ZTP-SCENARIOS
Jun 26 02:34:59 ZTP Phase 2 SCRIPT: Looking for apply-macro SCENARIO-1 in group
GR-ZTP-SCENARIOS
Jun 26 02:35:00 ZTP Phase 2 SCRIPT: Found apply-macro SCENARIO-1
Jun 26 02:35:00 ZTP Phase 2 SCRIPT: ZTD Pull model is in use.
Jun 26 02:35:00 ZTP Phase 2 SCRIPT: Going to proceed with Junos S/W upgrade
Jun 26 02:35:00 ZTP Phase 2 SCRIPT: Found a group GR-ZTP
Jun 26 02:35:00 ZTP Phase 2 SCRIPT: Looking for apply-macro CODE in group GR-ZTP
Jun 26 02:35:00 ZTP Phase 2 SCRIPT: Found apply-macro CODE
Jun 26 02:35:00 ZTP Phase 2 SCRIPT: running_version = 12.3X54-D15.3
Jun 26 02:35:00 ZTP Phase 2 SCRIPT: production_version = 15.1X54-D25
Jun 26 02:35:00 ZTP Phase 2 SCRIPT: upgrade required
Jun 26 02:35:00 ZTP Phase 2 SCRIPT: Processing
Jun 26 02:35:00 ZTP Phase 2 SCRIPT: Skip software upgrade due to Upgrade flag in
 basic template set to inactive
Jun 26 02:35:13 ZTP Phase 2 SCRIPT: ZTP PHASE 2 Configuration group was deactivated
Jun 26 02:35:13 ZTP Phase 2 SCRIPT: Found a group GR-ZTP-CALLHOME
Jun 26 02:35:13 ZTP Phase 2 SCRIPT: Looking for apply-macro CALLHOME in group
GR-ZTP-CALLHOME
Jun 26 02:35:13 ZTP Phase 2 SCRIPT: Found apply-macro CALLHOME
Jun 26 02:35:13 ZTP Phase 2 SCRIPT: Getting box Serial Number
Jun 26 02:35:14 ZTP Phase 2 SCRIPT: serial no = NK02131Oxxxx
Jun 26 02:35:14 ZTP Phase 2 SCRIPT: Config file:
ftp://ztdadmin:jnpr1234@10.1.1.4://config/NK02131Oxxxx.conf will be loaded and
merged to existed configuration
Jun 26 02:35:14 ZTP Phase 2 SCRIPT: Configuration was locked
Jun 26 02:35:14 ZTP Phase 2 SCRIPT: Processing config url
ftp://ztdadmin:jnpr1234@10.1.1.4://config/NK02131Oxxxx.conf
Jun 26 02:35:14 ZTP Phase 2 SCRIPT: Configuration was loaded
Jun 26 02:35:26 init: can not access /usr/sbin/relayd: No such file or directory
Jun 26 02:35:26 init: relay-process (PID 0) started
Jun 26 02:35:26 init: service-deployment (PID 11813) started
Jun 26 02:35:27 ZTP Phase 2 SCRIPT: Configuration was committed
Jun 26 02:35:27 ZTP Phase 2 SCRIPT: release config lock
Jun 26 02:35:28 ZTP Phase 2 SCRIPT: Box specific configuration was loaded
Jun 26 02:35:28 ZTP Phase 2 SCRIPT: ZTD Phase 2 completed successfully
```

**Meaning** The output confirms that the installation process worked correctly, and the ZTP script phases completed successfully.

### Verifying the Basic Configuration on the ACX Router

**Purpose** Verify that the basic configuration is downloaded and committed on the ACX router.

This occurred during Phase 0 of the ZTD process.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration** command.

```
user> show configuration
groups {
    global {
        system {
            debugger-on-panic;
            debugger-on-break;
            dump-on-panic;
            authentication-order [ password radius ];
            root-authentication {
                encrypted-password "$ABC123"; ## SECRET-DATA
            }
            login {
                user user {
                    uid 928;
                    class superuser;
                    shell csh;
                    authentication {
                        encrypted-password "$ABC123"; ## SECRET-DATA
                    }
                }
            }
            services {
                finger;
                ftp;
                rlogin;
                rsh;
                ssh;
                telnet;
                xnm-clear-text;
                netconf {
                    ssh;
                }
            }
        }
    }
    GR-ZTP {
        apply-macro CONFIG {
            CONFIG
"ftp://ztdadmin:jnpr1234@10.1.1.4://config/acx_global_config.conf";
        }
        apply-macro CODE {
            VERSION "15.1X54-D25";
            INACTIVE "active";
```

```
            IMAGE
"ftp://ztdadmin:jnpr1234@10.1.1.4://build/jinstall-ppc-15.1X54-D25-domestic-signed.tgz";

        }
    }
    GR-ZTP-STAGE-0 {
        event-options {
            generate-event {
                ztp_script time-interval 60;
            }
            policy ztp_script {
                events ztp_script;
                then {
                    execute-commands {
                        commands {
                            "op url
ftp://ztdadmin:jnpr1234@10.1.1.4://scripts/ztp_script_0_basic.slax";
                        }
                    }
                }
            }
        }
    }
}
apply-groups [ global GR-ZTP-STAGE-0 ];
interfaces {
    ge-1/2/0 {
        unit 0 {
            family inet {
                dhcp-client {
                    vendor-id CSR-acx;
                }
            }
        }
    }
}
```

**Meaning** Phase 0 of the ZTD deployment process was successfully initiated. and the basic configuration was downloaded and committed on the ACX router.

### Verifying the Global Configuration on the ACX Router

**Purpose** Verify that the global configuration is downloaded and committed on the ACX router.

This occurred during Phase 0 of the ZTD process.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration** command.

```
user> show configuration
groups {
    GR-ZTP-SCENARIOS {
        apply-macro SCENARIO-1 {
   /*-----------------------------------------------------------------------*/
   /*   METHOD: Defines type of the model is used:                          */
   /*         - push - enables push model, skips sw upgrades and fetches for */
```

```
/*               configlet file generated by Space NMS at FTP location    */
/*         - pull - enables pull model - upgrades junos sw, fetches the   */
/*                                                              <S/N>.conf */
/*         - If not specified, pull method will be used by default        */
/*   STAGE 0: Assigns DHCP address to CSR in OAM VLAN (mandatory)         */
/*   STAGE 1: Enables Static IP connectivity (mandatory):                 */
/*            - Enables Static IP connectivity with OAM VLAN              */
/*            - Enables Global/(VLAN 1) OSPF IP connectivity              */
/*            - Creates host-name                                         */
/*            - Loads box-specific config for outbound management         */
/*                                                            (lab only) */
/*   STAGE 2: Upgrades Junos image  (optional)                            */
/*            Loads box specific configuration/template                   */
/*                              or                                        */
/*            Loads platform specific configuration/configlet             */
/*            to initiate a call to the Home Space NMS                    */
/*            (a combined “pull” and “push” model)                        */
/*   To manage ZTP scenarios work flow you can use following values to    */
/*   process stages:                                                      */
/*       enabled   (enables ZTP stage)                                    */
/*       disabled  (disables/skip ZTP stage)                              */
/*       stop      (disables ZTP stage and stop ZTP process at this point)*/
/*-----------------------------------------------------------------------*/
        STAGE-0 enabled;
        METHOD "pull";
        STAGE-1 enabled;
        STAGE-2 stop;
    }
}
GR-ZTP-CALLHOME {
    apply-macro CALLHOME {
        CONFIG "ftp://ztdadmin:jnpr1234@10.1.1.4://config";
        SPACE-CONFIGLET "space_configlet.conf";
    }
}
GR-ZTP-PLATFORM {
    apply-macro ZTP-acx2100 {
        NNI1 ge-1/2/0;
        NNI2 ge-1/2/1;
        HOST_NAME csr-;
        OAM_VLAN 2;
    }
    apply-macro ZTP-acx4000 {
        NNI1 ge-0/1/0;
        NNI2 ge-0/1/1;
        NNI3 ge-0/0/1;
        HOST_NAME csr-;
        OAM_VLAN 2;
    }
    apply-macro ZTP-acx5096 {
        NNI1 xe-0/0/0;
        NNI2 xe-0/0/2;
        NNI3 xe-0/0/46;
        HOST_NAME agg-;
        OAM_VLAN 2;
    }
}
GR-ZTP-STAGE-1 {
    event-options {
        generate-event {
            ztp_stage_1 time-interval 60;
```

```
            }
            policy ztp_script {
                events ztp_stage_1;
                then {
                    execute-commands {
                        commands {
                            "op url
ftp://ztdadmin:jnpr1234@10.1.1.4://scripts/ztp_script_1_oam.slax";
                        }
                    }
                }
            }
        }
     }
     GR-ZTP-STAGE-2 {
        event-options {
            generate-event {
                ztp_stage_2 time-interval 60;
            }
            policy ztp_script {
                events ztp_stage_2;
                then {
                    execute-commands {
                       commands {
                       "op url
ftp://ztdadmin:jnpr1234@10.1.1.4://scripts/ztp_script_2.slax";
                        }
                    }
                }
            }
        }
    }
    global {
        system {
            domain-name lab.example.com;
            time-zone America/Los_Angeles;
            debugger-on-panic;
            debugger-on-break;
            dump-on-panic;
            authentication-order [ password radius ];
            root-authentication {
                encrypted-password "$ABC123"; ## SECRET-DATA
            }
            name-server {
                192.168.5.68;
                192.168.60.131;
            }
            radius-server {
                192.168.69.162 secret "$ABC123"; ## SECRET-DATA
            }
            login {
                user user {
                    uid 928;
                    class superuser;
                    shell csh;
                    authentication {
                        encrypted-password "$ABC123"; ## SECRET-DATA
                    }
                }
            }
            services {
```

```
            finger;
            ftp;
            rlogin;
            rsh;
            ssh;
            telnet;
            xnm-clear-text;
            netconf {
                ssh;
            }
        }
        syslog {
            host log {
                kernel info;
                any notice;
                pfe info;
                interactive-commands any;
            }
            file messages {
                kernel info;
                any notice;
                authorization info;
                pfe info;
                archive world-readable;
            }
            file security {
                interactive-commands any;
                archive world-readable;
            }
        }
        processes {
            routing enable;
            ntp enable;
            management enable;
            watchdog enable;
            snmp enable;
            inet-process enable;
            mib-process enable;
        }
        ntp {
            boot-server 172.17.28.5;
            server 172.17.28.5;
        }
    }
    chassis {
        dump-on-panic;
    }
    interfaces {
        lo0 {
            unit 0 {
                family inet {
                    address 127.0.0.1/32;
                }
            }
        }
    }
    snmp {
        location "Solution lab";
        community public {
            authorization read-only;
        }
```

```
            community private {
                authorization read-write;
            }
        }
    }
    GR-CSR-ACCESS-INTF {
        interfaces {
            <*> {
                traps;
                mtu 9192;
            }
        }
    }
    GR-NNI-TAG {
        interfaces {
            <[g|x]e-*>" {
                traps;
                mtu 9182;
                hold-time up 5000 down 0;
            }
        }
    }
}
```

**Meaning** Phase 0 of the ZTD deployment process completed successfully, and the global configuration is committed on the ACX router.

### Verifying the IRB Interface Configuration

**Purpose** Verify that the ACX router has its IRB interface configuration.

This occurred during Phase 1 of the ZTD process.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration interfaces irb** command.

```
user> show configuration interfaces irb
 unit 2 {
    family inet {
        address 192.168.1.2/24;  ## address from the DHCP pool
    }
}
```

**Meaning** Layer 3 IP routing is enabled on the IRB interface.

### Verifying the Core-Facing Interface (NNI) Configuration

**Purpose** Verify that the core-facing interface (NNI) of the ACX router is configured correctly.

This occurred during Phase 1 of the ZTD process.

NOTE: This example requires just the single NNI. However, the global configuration template and script used in this case were configured to provide two NNIs, thus both are verified here as enabled and configured.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration interfaces ge-1/2/0** and **show configuration interfaces ge-1/2/1** commands.

```
user> show configuration interfaces ge-1/2/0
apply-groups GR-NNI-TAG;
flexible-vlan-tagging;
native-vlan-id 2;
encapsulation flexible-ethernet-services;
unit 2 {
    description "OAM VLAN ENABLES ZTP AND NMS ACCESS";
    encapsulation vlan-bridge;
    vlan-id 2;
}
```

```
user> show configuration interfaces ge-1/2/1
apply-groups GR-NNI-TAG;
flexible-vlan-tagging;
native-vlan-id 2;
encapsulation flexible-ethernet-services;
unit 2 {
    description "OAM VLAN ENABLES ZTP AND NMS ACCESS";
    encapsulation vlan-bridge;
    vlan-id 2;
}
```

**Meaning** The NNI interfaces of the ACX router have been correctly configured, including a configuration group and management VLAN.

NOTE: A possible use case for the second interface is to allow another ACX router to connect to this one, creating a chain topology.

### Verifying the Management Bridge Domain Configuration

**Purpose** Verify that the management (OAM) bridge domain is configured to connect the ACX router to the aggregation segment.

This occurred during Phase 1 of the ZTD process.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration bridge-domains** command.

```
user> show configuration bridge-domains
BD-ZTP-OAM {
    vlan-id 2;
    interface ge-1/2/0.2;
    interface ge-1/2/1.2;
    routing-interface irb.2;
}
```

NOTE: When verifying bridge domains on an ACX5000 Series router, use the **show configuration vlans** command.

**Meaning** The NNIs and IRB interface are associated to the management bridge domain.

### Verifying ACX Reachability to the Aggregation Router

**Purpose** Verify that the ACX router has a default route to provide reachability to the aggregation router.

This occurred during Phase 1 of the ZTD process.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration routing-options** command.

```
user> show configuration routing-options
static {
   route 0.0.0.0/0 next-hop 192.168.1.251;   ## address from DHCP server's "option
 routers" (GW) value
}
```

**Meaning** The default route is configured, providing connectivity between the ACX router and aggregation router.

### Verifying the VSTP and LLDP Configuration

**Purpose** Verify that VSTP and LLDP are enabled on the core-facing (NNI) interfaces.

This occurred during Phase 1 of the ZTD process.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration protocols** command.

```
user> show configuration protocols
vstp {
```

```
    interface ge-1/2/0;
    interface ge-1/2/1;
    vlan 2;
}
lldp {
    interface ge-1/2/0;
    interface ge-1/2/1;
}
```

**Meaning** VSTP and LLDP are correctly configured and include the NNI interfaces.

### Verifying the Junos OS Upgrade

**Purpose** Verify that the Junos OS is upgraded to the recommended or production version on the ACX router.

This occurred during Phase 2 of the ZTD process.

**Action** SSH to the ACX router. Enter operational mode and execute the **show version** command.

```
user@csr-1.2-acx2100> show version
Hostname: csr-1.2-acx2100
Model: acx2100
Junos: 15.1X54-D25
JUNOS Base OS boot [15.1X54-D25]
JUNOS Online Documentation [15.1X54-D25]
JUNOS Crypto Software Suite [15.1X54-D25]
JUNOS Base OS Software Suite [15.1X54-D25]
JUNOS Kernel Software Suite [15.1X54-D25]
JUNOS Packet Forwarding Engine Support (acx5k) [15.1X54-D25]
JUNOS Enterprise Software Suite [15.1X54-D25]
JUNOS Routing Software Suite [15.1X54-D25]
JUNOS py-base-i386 [15.1X54-D25]
JUNOS Host Software [15.1X54-D25]
```

**Meaning** Phase 2 of the ZTD deployment process completed successfully, and the Junos OS is upgraded to the desired version, in this case 15.1X54-D25.

Completing this step also indicates that the ZTD process is complete.

**Related Documentation**

- Customer Use Cases for ACX Series Routers on page 5
- ACX Autoinstallation Overview on page 8
- Deployment Methods for ACX Routers on page 12
- Example: Deploying ACX Routers Using the ZTD Push Method on page 49
- ACX Deployment Scripts on page 77

## Example: Deploying ACX Routers Using the ZTD Push Method

This example demonstrates zero touch deployment (ZTD) and one touch deployment (OTD) of ACX routers using the Push method.



### Requirements

This example uses the following hardware and software components:

- Three MX240 routers
- Five provisioned ACX2100 routers
- One unboxed ACX2100 router
- Configuration server with DHCP, TFTP, and FTP services enabled
- Junos OS Release 15.1 and later
- USB storage device
- Junos Space Network Management Platform running Release 14.1 or later

The following elements from the ZTD Pull method are also required for the Push method:

- Configuration server
- Basic configuration template
- Global configuration template
- ZTD scripts
- Aggregation routers

NOTE: For more details on these elements, see "Example: Deploying ACX Routers Using the ZTD Pull Method" on page 21.

For more details on the ZTD scripts, see "ACX Deployment Scripts" on page 77.

If you are deploying ACX routers using the OTD Push method, you must ensure Layer 3 connectivity between the ACX router, the DHCP server, and the Junos Space Platform. If you are using the OTD method with Layer 3 connectivity to the DHCP server, you do not need to enable DHCP relay functionality on the aggregation routers.

## Overview

This example describes the tasks required to deploy new ACX routers in a service provider network while ensuring minimal involvement from the deployment technician and the MSP NOC personnel. At the end of this ZTD process, the ACX router can also be managed by Junos Space.

When the ZTD method is used, a basic configuration is automatically downloaded onto the ACX router through the autoinstallation feature. The event options in the basic configuration trigger the download and execution of the Phase 0 script, which in turn downloads and commits the global configuration to the ACX router.

Event options in the global configuration trigger the Phase 1 script, which creates a management VLAN (also known as the OAM VLAN) to enable a ZTD management plane. The global configuration also enables management access, software processes, system log preferences, and user authentication settings.

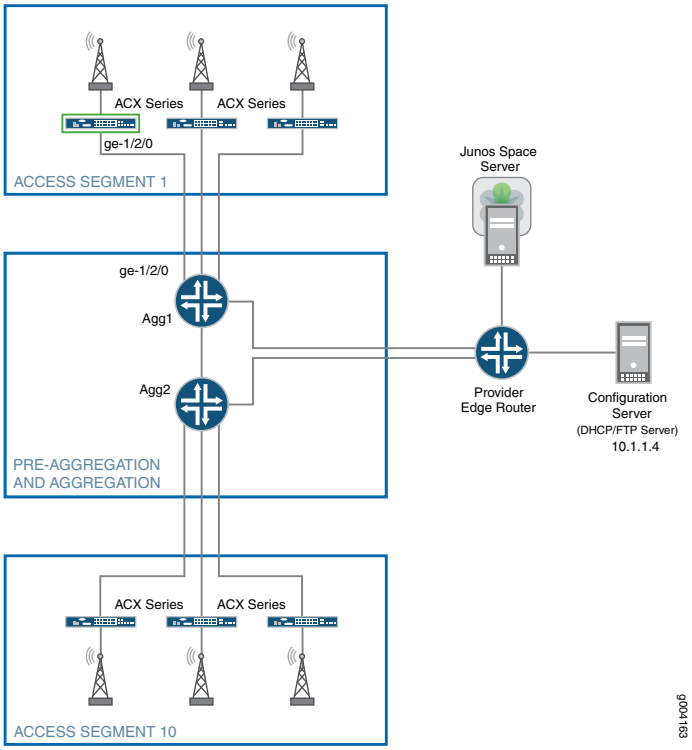
The global configuration template contains the URL to a configlet (initially generated by Junos Space) stored on the configuration server, named **space_configlet.conf**. The Phase 2 script downloads and commits the configlet on the ACX router, after which the router initiates a connection to, and is discovered by, Junos Space. After the discovery is successful, the device template associated with the ACX router is deployed and the Junos OS version is upgraded on the ACX router.

This example also describes the one touch deployment (OTD) method for scenarios where connectivity between the access nodes (ACX router) and aggregation router is provided through a third-party wholesale operator, and the new ACX router is not able to use DHCP to obtain location information for its home configuration server. In this scenario, some initial parameters must be loaded onto the ACX router manually. To do this, a configlet generated from Junos Space, which contains the connection parameters and Junos Space server configuration, is loaded onto the ACX router using a USB flash drive.

### Topology

Figure 8 on page 51 depicts a topology for ZTD of an ACX router using the Push method. The topology has three ACX routers in each access segment. The new ACX router being deployed in ACCESS SEGMENT 1 has its interface ge-1/2/0 connected to interface ge-1/2/0 on router Agg1 in the aggregation segment. Aggregation router Agg1 has an interface is connected to the provider edge router. The configuration server and Junos Space server are also connected to the provider edge router. The configuration server acts as a DHCP, TFTP, and FTP server, and hosts the configlets and scripts required to deploy the new ACX router.

Figure 8: Topology Diagram for Zero Touch Deployment By Using the Push Method




## Configuration

To deploy ACX routers using the ZTD Push method, perform these tasks:

- Creating Basic Configuration Files and a Global Configuration Template on page 51
- Configuring DHCP, TFTP, and FTP Services on the Configuration Server on page 52
- Configuring Aggregation Routers on page 52
- Creating a Device Template on page 52
- Uploading Junos OS Images on page 53
- Creating a Connection Profile on page 54
- Creating a Modeled Instance on page 56
- Adding the Configlet file to the Configuration Server on page 58
- Deploying the ACX Router on page 59

### Creating Basic Configuration Files and a Global Configuration Template

**CLI Quick Configuration**

To create basic configuration files and a global configuration template, use the CLI samples provided in the Pull method configuration example at "Example: Deploying ACX Routers Using the ZTD Pull Method" on page 21.

Make the following modifications to the Pull method configuration templates as part of the preparation for the Push method:

- Basic configuration file: in the group GR-ZTP, under the statement **apply-macro CODE**, replace the parameter **INACTIVE "active"** with **INACTIVE "inactive"**. This deactivates the Junos image upgrade procedure as part of the Phase 2 script execution.
- Global configuration file: in the group GR-ZTP-SCENARIOS, under the statement **apply-macro SCENARIO-1**, replace the parameter **METHOD "pull"** with **METHOD "push"**.

NOTE: **The configurations in this example include deployment-specific parameters such as MODE, root authentication, FTP URLs, server IP address, Junos OS version, configlet file name, and so on. Adjust these variables to suit your network environment.**

### Configuring DHCP, TFTP, and FTP Services on the Configuration Server

**CLI Quick Configuration**

To enable DHCP, TFTP, and FTP services on the configuration server, use the CLI samples provided in the Pull method configuration example at "Example: Deploying ACX Routers Using the ZTD Pull Method" on page 21.

### Configuring Aggregation Routers

**CLI Quick Configuration**

To configure aggregation routers Agg1 and Agg2 for the ZTD process, use the CLI samples provided in the Pull method configuration example at "Example: Deploying ACX Routers Using the ZTD Pull Method" on page 21.

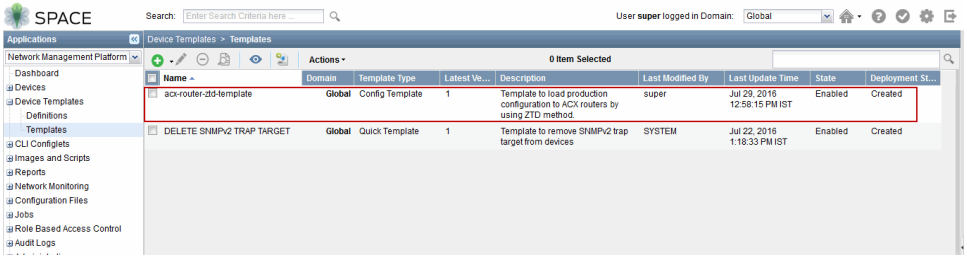
### Creating a Device Template

**Step-by-Step Procedure**

To use the Junos Space Network Management Platform to create a device template:

1. Log in to Junos Space and click the **Device Templates** workspace.
2. Click **Template Definitions** and create a template definition.
3. Click **Templates** and create a device template using the template definition.

The following display indicates a device template created to load the production configuration for the ACX router.




## Uploading Junos OS Images

**Step-by-Step Procedure**

The Push method can include upgrading the Junos OS to a recommended image through Junos Space.

To import the recommended version of Junos OS image to Junos Space Platform:

1. In Junos Space, select **Images and Scripts > Images**.

   The Images page is displayed.

2. Click the **Import Image** icon.

   The Import Images dialog box appears.

3. Click **Browse**.

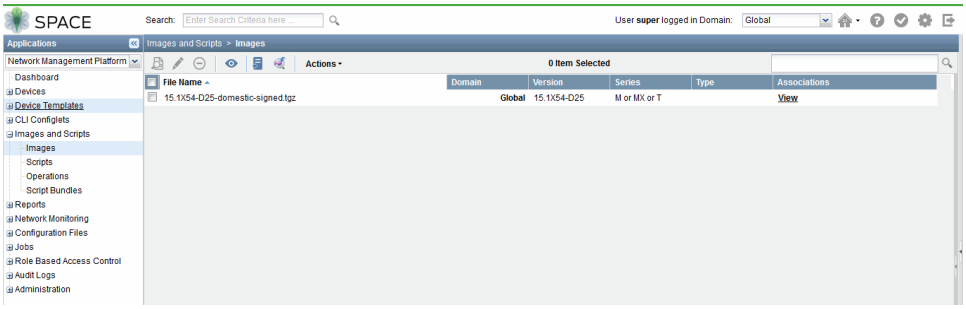
   The File Upload dialog box displays the directories and folders on your local file system.

4. Navigate to the device image file that you want to import and click **Open**.

5. Click **Upload** in the Import Images dialog box.

   Once the file is imported to the Junos Space server, it is listed on the Images page. You can associate the image to the ACX router when you create the modeled instance.

The following display shows the recommended Junos OS version uploaded and available in Junos Space.




## Creating a Connection Profile

**Step-by-Step Procedure**

A connection profile specifies connectivity-related parameters for devices added to Junos Space using the Modeling devices feature. A connection profile contains device interface details, the NAT configuration details for Junos Space, and the protocol used to assign IP addresses to devices.

To create a connection profile:

1. In Junos Space, select **Devices > Model Devices > Connection Profiles**.

   The Connection Profiles page is displayed.

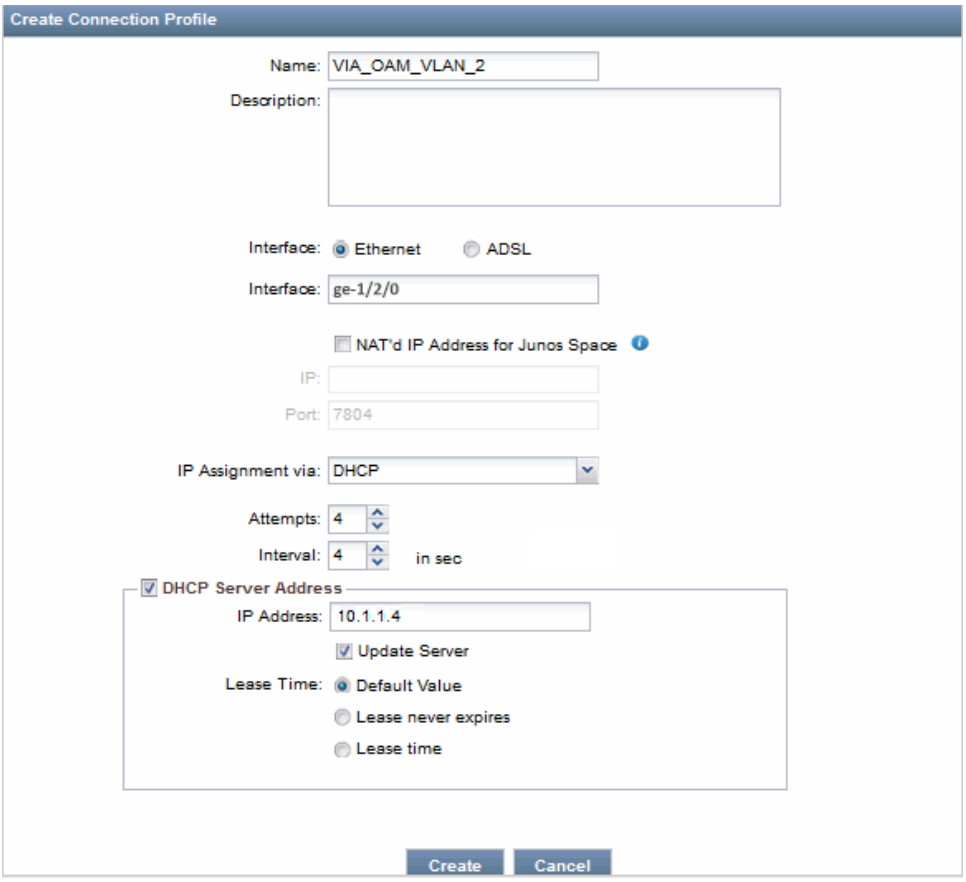
2. Click the **Create Connection Profile** icon on the Actions menu.

   The Create Connection Profile page is displayed.

3. Create a connection profile by specifying the following details:

   - Profile Name
   - Interface Type (Ethernet or ADSL)
   - Interface (Specify the ACX router interface that will be connected to the network)
   - IP Assignment via (for Ethernet: Static, DHCP, or PPPoE)
   - Attempts
   - Interval
   - DHCP Server Address
   - Lease Time

The following display indicates sample settings to create a connection profile.




4. Click **Create**.

   The connection profile is created.

## Creating a Modeled Instance

**Step-by-Step Procedure**

A modeled instance allows you associate a list of devices with a common set of connectivity parameters. To make it easier to add large numbers of devices, you can upload a CSV file to Junos Space. The CSV file should list each device on its own line, using the format <ID>,<serial-number>,<device-type>.

NOTE: The Create Modeled Instance page includes a sample CSV file.

To create a modeled instance to deploy the ACX routers:

1. In Junos Space, select **Devices > Device Management > Model Devices**.

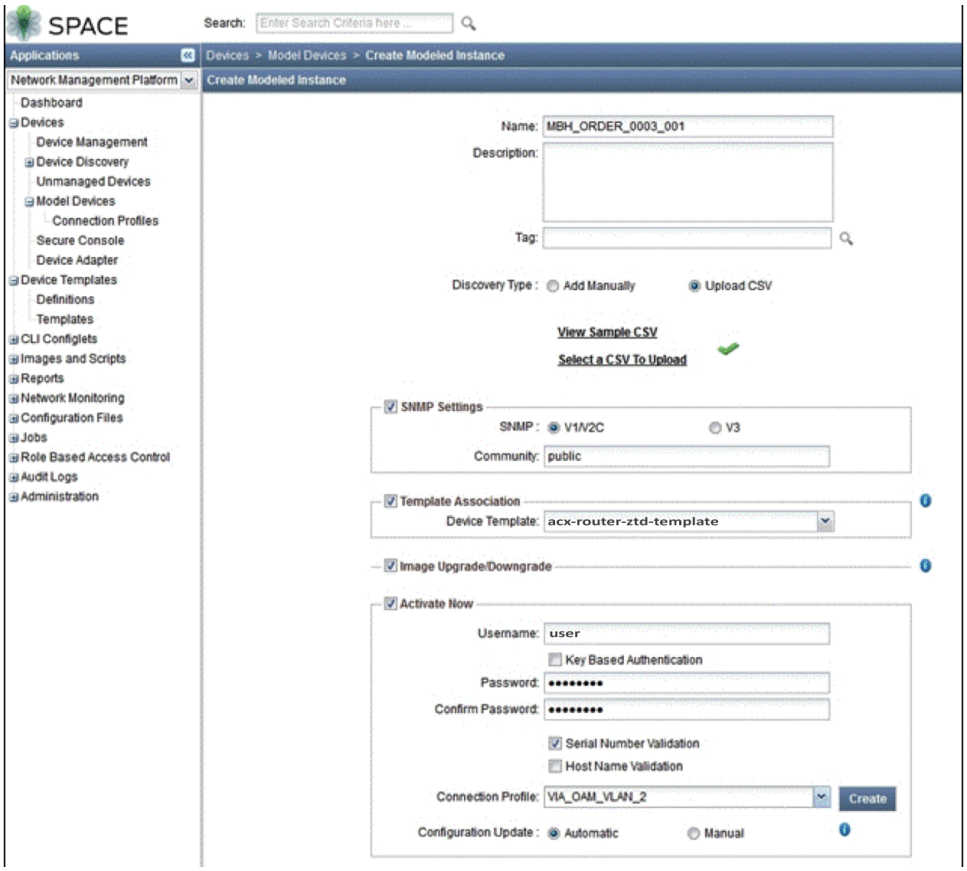
   The Model Devices page is displayed.

2. Click the **Create Modeled Instance** icon on the Actions menu.

   The Create Modeled Instance page is displayed.

3. Create a modeled instance by specifying the following details:
   - Name
   - Discovery Type (Add Manually or Upload CSV)
   - Template Association (Associate the device template created earlier)
   - Image Upgrade/Downgrade
   - Specify the following options in the Activate Now section:
     - Username (Account on ACX router, in this case **user**)
     - Password
     - Confirm Password
     - Serial Number Validation
     - Connection Profile (Associate the connection profile created earlier)
     - Configuration Update

The following display indicates sample settings to create a modeled instance.




4. Click **Next**.

   The Create Modeled Instance page displays the device instance of the ACX routers.

5. (Optional) Modify the default hostname, platform, IP address, and gateway details on a per-device basis.

6. Click **Finish**.

   The modeled instance is created. You are redirected to the Model Devices page.

   The instances of the ACX router display the Connection Status as Down and Managed Status as Waiting for Deployment in the Device Management page.

## Adding the Configlet file to the Configuration Server

**Step-by-Step Procedure**

At the end of Phase 2 of the ZTD process, a configlet file is used to start a device-initiated connection to Junos Space. This configlet file is created on the Junos Space server, but must be added to the configuration server.

To add the configlet to the configuration server:

1. In Junos Space, select **Devices > Model Devices**.

   The Model Devices page is displayed.

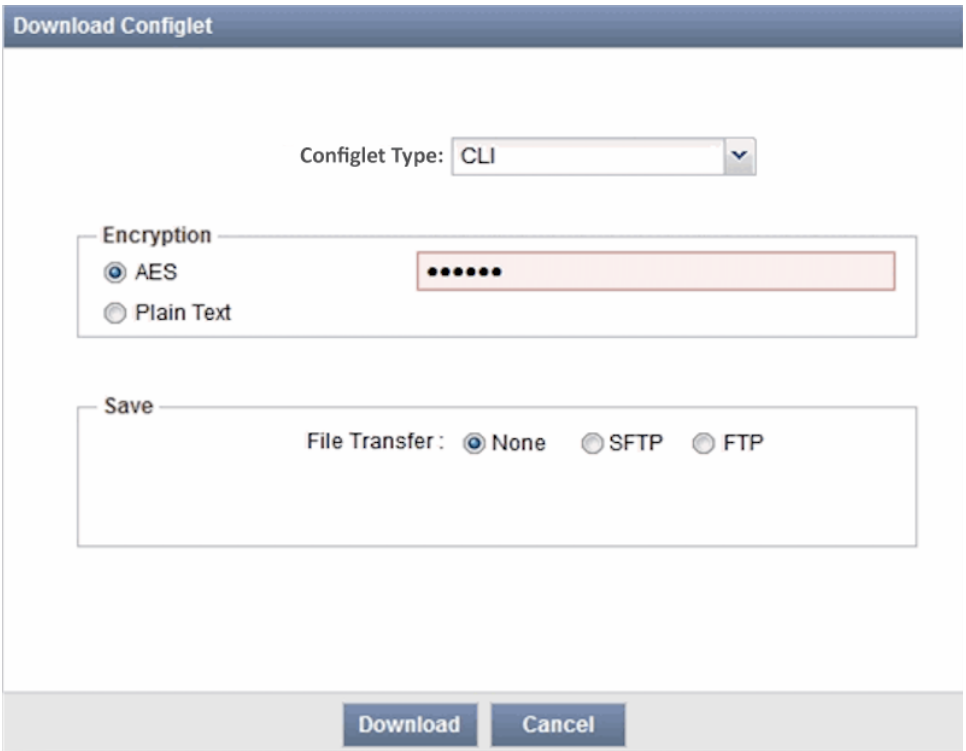
2. Select the modeled instance whose configlet you want to download, and select **Download Configlet** from the Actions menu.

   The Download Configlet page is displayed.

3. Transfer the configlet to the configuration server by specifying the following details:

   - Configlet Type (CLI )
   - Encryption (AES or Plain Text)
   - Save (None - download to your computer)

   The following display indicates sample settings to save the configlet.



4. Click **Download**.

   The **Configlets.zip** file is downloaded to your local computer.

5. Rename the configlet as a configuration file, **space_configlet.conf**.

6. Open the file with a text editor and remove the interface configuration from the configlet.

7. Upload the configlet file to the configuration repository of the FTP server.

### Deploying the ACX Router

**Step-by-Step Procedure**

To deploy the ACX router:

1. Unpack the ACX router, power it on, and connect the designated interface to the network.

   For this example, the new the ACX2100 router uses interface ge-1/2/0.

   Once the router boots up, the autoinstallation function of the ACX router communicates with the DHCP server, acquires an initial IP address, downloads and commits the basic configuration template, and proceeds to use the ZTD scripts to setup the device, as follows:

   - When the basic configuration is committed to the ACX router, the event options in the basic configuration trigger the execution of the ZTD Phase 0 script. This script downloads and commits the global configuration.
   - When the global configuration is committed to the ACX router, the event options in the global configuration trigger the execution of the ZTD Phase 1 script. The ZTD Phase 1 script creates a management (OAM) VLAN to enable a ZTD management plane through the access and aggregation segments.
   - The global configuration also triggers the execution of the ZTD Phase 2 script. The ZTD Phase 2 script commits the router-specific configuration, validates and upgrades the Junos OS to the recommended version, and commits the production configuration from Junos Space to the ACX router.

   NOTE: The ZTD scripts can be found in the section "ACX Deployment Scripts" on page 77.

2. Verify that the ACX router has its full and proper configuration using the verification steps in the section "Verification" on page 61.

## Deploying ACX Routers Using the One Touch Deployment (OTD) Push Method

If you are deploying ACX routers using the OTD Push method, perform these additional tasks:


- Creating a Configlet USB on page 60
- Deploying the ACX Router on page 61


### Creating a Configlet USB

**Step-by-Step Procedure**

With the OTD Push method, the ACX router is activated by connecting a USB device and installing a configlet, which enables the device to initiate a connection to Junos Space.

To download the configlet to a USB drive:

1. In Junos Space, select **Devices > Model Devices**.

   The Model Devices page is displayed.

2. Select the modeled instance whose configlet you want to download, and select **Download Configlet** from the Actions menu.

   The Download Configlet page is displayed.

3. Specify the following details to download the configlet:
   - Configlet Type (CLI)
   - Encryption (AES)
   - Save (None, to download the file to your local computer)
4. Click **Download**.

   The **Configlets.zip** file is downloaded to your local computer.

5. Unzip the .ZIP file and copy the configlet to the USB device.

   The following is a sample of a DHCP static configlet for OTD, generated by the Junos Space Platform:

```
system {
    services {
        ssh {
            protocol-version v2;
        }
        outbound-ssh {
            client cluster_228.161.210.86 {
                device-id B988D4;
                secret "$ABC123";
                services netconf;
                10.1.1.1 port 7804; ## Junos Space server IP address
            }
        }
    }
```

```
    login {
      user user {
        class super-user;
        authentication {
          encrypted-password "$ABC123";
        }
      }
    }
  }
interfaces ge-0/0/0 {
  unit 0 {
    family inet {
      dhcp-client {
        retransmission-attempt 4;
        retransmission-interval 4;
      }
    }
  }
}
```

### Deploying the ACX Router

**Step-by-Step Procedure**

To deploy the ACX router:

1. Unpack the ACX router, power it on, and connect the interface specified in the connection profile to the network.

2. Plug the USB device into the USB port on the ACX router.

3. Reboot the ACX router.

   A persistent SSH connection is established with the Junos Space Platform.

4. Verify that the ACX router has its full and proper configuration using the verification steps in the section "Verification" on page 61.

## Verification

Use the following procedures to verify the deployment of ACX routers during the Push method:

NOTE: To verify the basic configuration, global configuration, configuration through scripts, and Junos OS upgrades, it is helpful to be connected to a system log server to receive notifications about the execution of scripts, configuration downloads, and so on, as they happen. As you receive appropriate notifications, you can then further verify the configuration downloads and Junos OS upgrades by establishing an SSH connection to the ACX router and using the verification steps below.



### Verifying the Progress of the ZTD Process

**Purpose** Verify the progress of the ZTD process and verify the basic configuration, global configuration, configuration through scripts, and Junos OS upgrades.

NOTE: This step can be used in conjunction with the appropriate verification procedures below, depending on the stage of the ZTD process.

The **monitor start** command allows you to view entries being added to a log file in real-time. In this example, the ZTD process is logged to the file **op-script.log**. You can use this method to monitor the ZTD process as it is happening.

Immediately after the basic configuration is applied, you should be able to gain access to the ACX router using SSH and the initial IP address provided by DHCP server. Note that as the process moves forward, your session will be broken a few times, as the initial IP address is reconfigured to a permanent IP address, and the router reboots to complete its OS upgrade.

If the process has an issue at any point, you will be able to review the log file generated by script and identify what went wrong.

NOTE: While not typical, if you happen to have console access to the ACX router, you can use this method to monitor the process and read ZTD log file in real-time, without interruption.

**Action** SSH to the ACX router. Enter configuration mode and execute the **monitor start op-script.log** command.

```
user1# monitor start op-script.log
Jun 26 09:41:07 op script processing begins
Jun 26 09:41:07 reading op script input details
Jun 26 09:41:07 testing op details
Jun 26 09:41:07 running op script
'/var/tmp/tmp_op8Pc5Pw/...transferring.file.........7xJn46/ztp_script_0_basic.slax'
Jun 26 09:41:07 opening op script
'/var/tmp/tmp_op8Pc5Pw/...transferring.file.........7xJn46/ztp_script_0_basic.slax'
Jun 26 09:41:07 reading op script
'/var/tmp/tmp_op8Pc5Pw/...transferring.file.........7xJn46/ztp_script_0_basic.slax'
Jun 26 09:41:11 ZTP Phase 0 SCRIPT: ZTP phase 0 BEGIN
Jun 26 09:41:11 ZTP Phase 0 SCRIPT: Found a group GR-ZTP
Jun 26 09:41:11 ZTP Phase 0 SCRIPT: Looking for apply-macro CONFIG in group GR-ZTP
Jun 26 09:41:11 ZTP Phase 0 SCRIPT: Found apply-macro CONFIG
Jun 26 09:41:11 ZTP Phase 0 SCRIPT: Config file:
ftp://ztdadmin:jnpr1234@10.1.1.4://config/ztd_global.conf will be loaded and
merged to existed configuration
Jun 26 09:41:11 ZTP Phase 0 SCRIPT: Got config lock
Jun 26 09:41:11 ZTP Phase 0 SCRIPT: Processing config url
ftp://ztdadmin:jnpr1234@10.1.1.4://config/ztd_global.conf
Jun 26 09:41:11 ZTP Phase 0 SCRIPT: Configuration was loaded
Jun 26 09:41:25 ZTP Phase 0 SCRIPT: Configuration was committed
Jun 26 09:41:25 ZTP Phase 0 SCRIPT: release config lock
Jun 26 09:41:25 ZTP Phase 0 SCRIPT: ZTP Global configuration template loaded
Jun 26 09:41:25 ZTP Phase 0 SCRIPT: Found a group GR-ZTP-STAGE-1
Jun 26 02:41:38 init: can not access /usr/sbin/relayd: No such file or directory
Jun 26 02:41:38 init: relay-process (PID 0) started
Jun 26 09:41:39 ZTP Phase 0 SCRIPT: ZTP Phase 0 completed SUCCESSFULLY
Jun 26 09:41:39 op script output
Jun 26 09:41:39 begin dump
Jun 26 09:41:39 end dump
Jun 26 09:41:39 inspecting op output
'/var/tmp/tmp_op8Pc5Pw/...transferring.file.........7xJn46/ztp_script_0_basic.slax'
Jun 26 09:41:39 finished op script
'/var/tmp/tmp_op8Pc5Pw/...transferring.file.........7xJn46/ztp_script_0_basic.slax'
Jun 26 09:41:39 op script processing ends
Jun 26 02:42:02 op script processing begins
Jun 26 02:42:02 reading op script input details
Jun 26 02:42:02 testing op details
Jun 26 02:42:02 running op script
'/var/tmp/tmp_op2ipdq8/...transferring.file.........T9EWHw/ztp_script_1_oam.slax'
Jun 26 02:42:02 opening op script
'/var/tmp/tmp_op2ipdq8/...transferring.file.........T9EWHw/ztp_script_1_oam.slax'
Jun 26 02:42:02 reading op script
'/var/tmp/tmp_op2ipdq8/...transferring.file.........T9EWHw/ztp_script_1_oam.slax'
Jun 26 02:42:05 ZTP Phase 1 SCRIPT: ZTP Phase 1 BEGIN
Jun 26 02:42:05 ZTP Phase 1 SCRIPT: Deactivating event-options for ZTP Phase 1
script...
```

```
Jun 26 02:42:17 ZTP Phase 1 SCRIPT: Configuration for event-options for ZTP Phase
 1 was disabled
Jun 26 02:42:17 ZTP Phase 1 SCRIPT: Getting box Serial Number
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: serial no = NK021310xxxx
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Retrieve type of platform
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: platform = acx2100
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Looking for acx2100 specific parameters.
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Found a group GR-ZTP-PLATFORM
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Looking for apply-macro ZTP-acx2100 in group
 GR-ZTP-PLATFORM
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Found apply-macro ZTP-acx2100
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Retrieve DHCP client bindings
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Found host name prefix: csr-
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Found vlan-id for OAM plane: 2
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: DHCP client runs on ZTP interface: ge-1/2/0.0
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Found DHCP Option for router: [ 192.168.1.250
 ]
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Derived next-hop: 192.168.1.250
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: DHCP ip address found: 192.168.2.250/24
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Parsing ZTP Interface: ifd=ge-1/2/0 unit=0
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Configuring NNI interface: ge-1/2/0
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Configuring NNI interface: ge-1/2/1
Jun 26 02:42:18 ZTP Phase 1 SCRIPT: Configuring bridge-domain: BD-ZTP-OAM
Jun 26 02:42:31 init: dhcp-service (PID 12205) terminate signal sent
Jun 26 02:42:31 init: general-authentication-service (PID 12206) terminate signal
 sent
Jun 26 02:42:31 init: can not access /usr/sbin/relayd: No such file or directory
Jun 26 02:42:31 init: relay-process (PID 0) started
Jun 26 02:42:31 init: general-authentication-service (PID 12206) exited with
status=0 Normal Exit
Jun 26 02:42:31 init: dhcp-service (PID 12205) exited with status=0 Normal Exit
Jun 26 02:42:32 ZTP Phase 1 SCRIPT: Platform specific configuration committed
successfully.
Jun 26 02:42:32 ZTP Phase 1 SCRIPT: Looking for box (S/N: NK021310xxxx) specific
 parameters.
Jun 26 02:42:32 ZTP Phase 1 SCRIPT: Found a group GR-ZTP-BOX
Jun 26 02:42:32 ZTP Phase 1 SCRIPT: Looking for apply-macro NK021310xxxx in group
 GR-ZTP-BOX
Jun 26 02:42:32 ZTP Phase 1 SCRIPT: Can't find apply-macro NK0213100096
Jun 26 02:42:32 ZTP Phase 1 SCRIPT: Box specific configuration for S/N:
NK021310xxxx was not found in global template. Skip this step.
Jun 26 02:42:32 ZTP Phase 1 SCRIPT: Found a group GR-ZTP-BOX
Jun 26 02:42:32 ZTP Phase 1 SCRIPT: Deleting configuration group GR-ZTP-BOX
Jun 26 02:42:33 ZTP Phase 1 SCRIPT: Found a group GR-ZTP-STAGE-2
Jun 26 02:42:46 init: can not access /usr/sbin/relayd: No such file or directory
Jun 26 02:42:46 init: relay-process (PID 0) started
Jun 26 02:42:47 ZTP Phase 1 SCRIPT: Configuration was committed
Jun 26 02:42:47 ZTP Phase 1 SCRIPT: ZTP Phase 1 completed successfully
Jun 26 02:42:47 op script output
Jun 26 02:42:47 begin dump
Jun 26 02:42:47 end dump
Jun 26 02:42:47 inspecting op output
'/var/tmp/tmp_op2ipdq8/...transferring.file.........T9EWHw/ztp_script_1_oam.slax'
Jun 26 02:42:47 finished op script
'/var/tmp/tmp_op2ipdq8/...transferring.file.........T9EWHw/ztp_script_1_oam.slax'
Jun 26 02:42:47 op script processing ends
Jun 26 02:43:10 op script processing begins
Jun 26 02:43:10 reading op script input details
Jun 26 02:43:10 testing op details
Jun 26 02:43:10 running op script
'/var/tmp/tmp_opBHWUiz/...transferring.file.........Xcljsg/ztp_script_2.slax'
```

```
Jun 26 02:43:10 opening op script
'/var/tmp/tmp_opBHWUiz/...transferring.file.........Xcljsg/ztp_script_2.slax'
Jun 26 02:43:10 reading op script
'/var/tmp/tmp_opBHWUiz/...transferring.file.........Xcljsg/ztp_script_2.slax'
Jun 26 02:43:14 ZTP Phase 2 SCRIPT: ZTP Phase 2 BEGIN
Jun 26 02:43:14 ZTP Phase 2 SCRIPT: Found a group GR-ZTP-SCENARIOS
Jun 26 02:43:14 ZTP Phase 2 SCRIPT: Looking for apply-macro SCENARIO-1 in group
GR-ZTP-SCENARIOS
Jun 26 02:43:14 ZTP Phase 2 SCRIPT: Found apply-macro SCENARIO-1
Jun 26 02:43:14 ZTP Phase 2 SCRIPT: ZTD Push model is in use.
Jun 26 02:43:14 ZTP Phase 2 SCRIPT: Found a group GR-ZTP-CALLHOME
Jun 26 02:43:14 ZTP Phase 2 SCRIPT: Looking for apply-macro CALLHOME in group
GR-ZTP-CALLHOME
Jun 26 02:43:14 ZTP Phase 2 SCRIPT: Found apply-macro CALLHOME
Jun 26 02:43:15 ZTP Phase 2 SCRIPT: Next ZTP phase will be enabled by Space after
 completion of the call home phase
Jun 26 02:43:28 ZTP Phase 2 SCRIPT: Config file:
ftp://ztdadmin:jnpr1234@10.1.1.4://config/space_configlet.conf will be loaded and
 merged to existed configuration
Jun 26 02:43:28 ZTP Phase 2 SCRIPT: Configuration was locked
Jun 26 02:43:28 ZTP Phase 2 SCRIPT: Processing config url
ftp://ztdadmin:jnpr1234@10.1.1.4://config/space_configlet.conf
Jun 26 02:43:28 ZTP Phase 2 SCRIPT: Configuration was loaded
Jun 26 02:43:42 init: can not access /usr/sbin/relayd: No such file or directory
Jun 26 02:43:42 init: relay-process (PID 0) started
Jun 26 02:43:42 init: service-deployment (PID 14965) started
Jun 26 02:43:43 ZTP Phase 2 SCRIPT: Configuration was committed
Jun 26 02:43:43 ZTP Phase 2 SCRIPT: release config lock
Jun 26 02:43:43 ZTP Phase 2 SCRIPT: Phase 2 completed successfully
Jun 26 02:43:43 ZTP Phase 2 SCRIPT: Now calling home to continue ZTP process.
Jun 26 02:43:43 op script output
Jun 26 02:43:43 begin dump
Jun 26 02:43:43 end dump
Jun 26 02:43:43 inspecting op output
'/var/tmp/tmp_opBHWUiz/...transferring.file.........Xcljsg/ztp_script_2.slax'
Jun 26 02:43:43 finished op script
'/var/tmp/tmp_opBHWUiz/...transferring.file.........Xcljsg/ztp_script_2.slax'
Jun 26 02:43:43 op script processing ends
```

**Meaning** The output confirms that the installation process worked correctly, and the ZTP script phases completed successfully.

### Verifying the Basic Configuration on the ACX Router

**Purpose** Verify that the basic configuration is downloaded and committed on the ACX router.

This occurred during Phase 0 of the ZTD process.

NOTE: This procedure is required to verify only zero touch deployment.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration** command.

```
user> show configuration
groups {
    global {
        system {
            debugger-on-panic;
            debugger-on-break;
            dump-on-panic;
            authentication-order [ password radius ];
            root-authentication {
                encrypted-password "$ABC123"; ## SECRET-DATA
            }
            login {
                user user {
                    uid 928;
                    class superuser;
                    shell csh;
                    authentication {
                        encrypted-password "$ABC123"; ## SECRET-DATA
                    }
                }
            }
            services {
                finger;
                ftp;
                rlogin;
                rsh;
                ssh;
                telnet;
                xnm-clear-text;
                netconf {
                    ssh;
                }
            }
        }
    }
    GR-ZTP {
        apply-macro CONFIG {
            CONFIG
"ftp://ztdadmin:jnpr1234@10.1.1.4://config/acx_global_config.conf";
        }
    }
    GR-ZTP-STAGE-0 {
        event-options {
            generate-event {
                ztp_script time-interval 60;
            }
            policy ztp_script {
                events ztp_script;
                then {
                    execute-commands {
                        commands {
                            "op url
ftp://ztdadmin:jnpr1234@10.1.1.4://scripts/ztp_script_0_basic.slax";
                        }
                    }
                }
            }
        }
```

```
    }
}
apply-groups [ global GR-ZTP-STAGE-0 ];
interfaces {
    ge-1/2/0 {
        unit 0 {
            family inet {
                dhcp-client {
                    vendor-id CSR-acx;
                }
            }
        }
    }
}
```

**Meaning** Phase 0 of the ZTD deployment process was successfully initiated. and the basic configuration was downloaded and committed on the ACX router.

### Verifying the Global Configuration on the ACX Router

**Purpose** Verify that the global configuration is downloaded and committed on the ACX router.

This occurred during Phase 0 of the ZTD process.

NOTE: This procedure is required to verify only zero touch deployment.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration** command.

```
user> show configuration
groups {
    GR-ZTP-SCENARIOS {
        apply-macro SCENARIO-1 {
    /*-------------------------------------------------------------------------*/
    /*   METHOD: Defines type of the model is used:                            */
    /*         - push - enables push model, skips sw upgrades and fetches for */
    /*              configlet file generated by Space NMS at FTP location    */
    /*         - pull - enables pull model - upgrades junos sw, fetches the   */
    /*                                                              <S/N>.conf */
    /*         - If not specified, pull method will be used by default        */
    /*   STAGE 0: Assigns DHCP address to CSR in OAM VLAN(mandatory)          */
    /*   STAGE 1: Enables Static IP connectivity (mandatory):                 */
    /*            - Enables Static IP connectivity with OAM VLAN              */
    /*            - Enables Global/(VLAN 1) OSPF IP connectivity              */
    /*            - Creates host-name                                         */
    /*            - Loads box-specific config for outbound management         */
    /*                                                            (lab only) */
    /*   STAGE 2: Upgrades Junos image  (optional)                            */
    /*            Loads box specific configuration/template                   */
    /*                              or                                        */
    /*            Loads platform specific configuration/configlet             */
```

```
    /*            to initiate a call to the Home Space NMS                    */
    /*            (a combined “pull” and “push” model)                        */
    /*   To manage ZTP scenarios work flow you can use following values to    */
    /*   process stages:                                                      */
    /*       enabled   (enables ZTP stage)                                    */
    /*       disabled  (disables/skip ZTP stage)                              */
    /*       stop      (disables ZTP stage and stop ZTP process at this point)*/
    /*-----------------------------------------------------------------------*/
            STAGE-0 enabled;
            METHOD "push";
            STAGE-1 enabled;
            STAGE-2 stop;
        }
    }
    GR-ZTP-CALLHOME {
        apply-macro CALLHOME {
            CONFIG "ftp://ztdadmin:jnpr1234@10.1.1.4://config";
            SPACE-CONFIGLET "space_configlet.conf";
        }
    }
    GR-ZTP-PLATFORM {
        apply-macro ZTP-acx2100 {
            NNI1 ge-1/2/0;
            NNI2 ge-1/2/1;
            HOST_NAME csr-;
            OAM_VLAN 2;
        }
        apply-macro ZTP-acx4000 {
            NNI1 ge-0/1/0;
            NNI2 ge-0/1/1;
            NNI3 ge-0/0/1;
            HOST_NAME csr-;
            OAM_VLAN 2;
        }
        apply-macro ZTP-acx5096 {
            NNI1 xe-0/0/0;
            NNI2 xe-0/0/2;
            NNI3 xe-0/0/46;
            HOST_NAME agg-;
            OAM_VLAN 2;
        }
    }
    GR-ZTP-STAGE-1 {
        event-options {
            generate-event {
                ztp_stage_1 time-interval 60;
            }
            policy ztp_script {
                events ztp_stage_1;
                then {
                    execute-commands {
                        commands {
                            "op url
ftp://ztdadmin:jnpr1234@10.1.1.4://scripts/ztp_script_1_oam.slax";
                        }
                    }
                }
            }
        }
    }
    GR-ZTP-STAGE-2 {
```

```
        event-options {
            generate-event {
                ztp_stage_2 time-interval 60;
            }
            policy ztp_script {
                events ztp_stage_2;
                then {
                    execute-commands {
                        commands {
                            "op url ftp://ztdadmin:jnpr1234@10.1.1.4://scripts/
ztp_script_2_pull.slax";
                        }
                    }
                }
            }
        }
    }
    global {
        system {
            domain-name lab.example.com;
            time-zone America/Los_Angeles;
            debugger-on-panic;
            debugger-on-break;
            dump-on-panic;
            authentication-order [ password radius ];
            root-authentication {
                encrypted-password "$ABC123"; ## SECRET-DATA
            }
            name-server {
                192.168.5.68;
                192.168.60.131;
            }
            radius-server {
                192.168.69.162 secret "$ABC123"; ## SECRET-DATA
            }
            login {
                user user {
                    uid 928;
                    class superuser;
                    shell csh;
                    authentication {
                        encrypted-password "$ABC123"; ## SECRET-DATA
                    }
                }
            }
            services {
                finger;
                ftp;
                rlogin;
                rsh;
                ssh;
                telnet;
                xnm-clear-text;
                netconf {
                    ssh;
                }
            }
            syslog {
                host log {
                    kernel info;
                    any notice;
```

```
                    pfe info;
                    interactive-commands any;
                }
                file messages {
                    kernel info;
                    any notice;
                    authorization info;
                    pfe info;
                    archive world-readable;
                }
                file security {
                    interactive-commands any;
                    archive world-readable;
                }
            }
            processes {
                routing enable;
                ntp enable;
                management enable;
                watchdog enable;
                snmp enable;
                inet-process enable;
                mib-process enable;
            }
            ntp {
                boot-server 172.17.28.5;
                server 172.17.28.5;
            }
        }
        chassis {
            dump-on-panic;
        }
        interfaces {
            lo0 {
                unit 0 {
                    family inet {
                        address 127.0.0.1/32;
                    }
                }
            }
        }
        snmp {
            location "Solution lab";
            community public {
                authorization read-only;
            }
            community private {
                authorization read-write;
            }
        }
    }
    GR-CSR-ACCESS-INTF {
        interfaces {
            <*> {
                traps;
                mtu 9192;
            }
        }
    }
    GR-NNI-TAG {
        interfaces {
```

```
            "<[g|x]e-*>" {
                traps;
                mtu 9182;
                hold-time up 5000 down 0;
            }
        }
    }
}
```

**Meaning** Phase 0 of the ZTD deployment process completed successfully, and the global configuration is committed on the ACX router.

### Verifying the IRB Interface Configuration

**Purpose** Verify that the ACX router has its IRB interface configuration.

This occurred during Phase 1 of the ZTD process.

NOTE: This procedure is required to verify only zero touch deployment.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration interfaces irb** command.

```
user> show configuration interfaces irb
 unit 2 {
    family inet {
        address 192.168.1.2/24;   ## address from the DHCP pool
    }
}
```

**Meaning** The IRB interface is configured with the IP address provided by DHCP server, and Layer 3 IP routing is enabled on the IRB interface.

### Verifying the Core-Facing Interface (NNI) Configuration

**Purpose** Verify that the core-facing interface (NNI) of the ACX router is configured correctly.

This occurred during Phase 1 of the ZTD process.

NOTE: This procedure is required to verify only zero touch deployment.

This example requires just the single NNI. However, the global configuration template and script used in this case were configured to provide two NNIs, thus both are verified here as enabled and configured.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration interfaces ge-1/2/0** and **show configuration interfaces ge-1/2/1** commands.

```
user> show configuration interfaces ge-1/2/0
apply-groups GR-NNI-TAG;
flexible-vlan-tagging;
native-vlan-id 2;
encapsulation flexible-ethernet-services;
unit 2 {
    description "OAM VLAN ENABLES ZTP AND NMS ACCESS";
    encapsulation vlan-bridge;
    vlan-id 2;
}
```

```
user> show interfaces ge-1/2/1
apply-groups GR-NNI-TAG;
flexible-vlan-tagging;
native-vlan-id 2;
encapsulation flexible-ethernet-services;
unit 2 {
    description "OAM VLAN ENABLES ZTP AND NMS ACCESS";
    encapsulation vlan-bridge;
    vlan-id 2;
}
```

**Meaning** The NNI interfaces of the ACX router have been correctly configured, including a configuration group and management VLAN.

NOTE: A possible use case for the second interface is to allow another ACX router to connect to this one, creating a chain topology.

### Verifying the Management Bridge Domain Configuration

**Purpose** Verify that the OAM bridge domain is configured to connect the access and aggregation segments.

This occurred during Phase 1 of the ZTD process.

NOTE: This procedure is required to verify only zero touch deployment.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration bridge-domains** command.

```
user> show configuration bridge-domains
BD-ZTP-OAM {
    vlan-id 2;
    interface ge-1/2/0.2;
```

```
    interface ge-1/2/1.2;
    routing-interface irb.2;
}
```

NOTE: When verifying bridge domains on an ACX5000 Series router, use the **show configuration vlans** command.

**Meaning** The NNIs and IRB interface are associated to the management bridge domain.

### Verifying ACX Reachability to the Aggregation Router

**Purpose** Verify that the ACX router has a default route to provide reachability to the aggregation router.

This occurred during Phase 1 of the ZTD process.

NOTE: This procedure is required to verify only zero touch deployment.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration routing-options** command.

```
user> show configuration routing-options
static {
    route 0.0.0.0/0 next-hop 192.168.1.251;   ## address from DHCP server's "option
 routers" (GW) value
}
```

**Meaning** The default route is configured, providing connectivity between the ACX router and aggregation router.

### Verifying the VSTP and LLDP Configuration

**Purpose** Verify that VSTP and LLDP are enabled on the core-facing (NNI) interfaces.

This occurred during Phase 1 of the ZTD process.

NOTE: This procedure is required to verify only zero touch deployment.

**Action** SSH to the ACX router. Enter operational mode and execute the **show configuration protocols** command.

```
user> show configuration protocols
vstp {
    interface ge-1/2/0;
    interface ge-1/2/1;
    vlan 2;
}
lldp {
    interface ge-1/2/0;
    interface ge-1/2/1;
}
```

**Meaning** VSTP and LLDP are correctly configured and include the NNI interfaces.

### Verifying the Junos OS Upgrade

**Purpose** Verify that the Junos OS is upgraded to the recommended or production version on the ACX router once the ACX router is discovered by Junos Space.

NOTE: This procedure is required to verify both zero touch deployment and one touch deployment.

**Action** SSH to the ACX router. Enter operational mode and execute the **show version** command.

```
user@csr1.2.acx2100> show version
 Hostname: csr1.2.acx2100
Model: acx2100
Junos: 15.1X54-D25
JUNOS Base OS boot [15.1X54-D25]
JUNOS Online Documentation [15.1X54-D25]
JUNOS Crypto Software Suite [15.1X54-D25]
JUNOS Base OS Software Suite [15.1X54-D25]
JUNOS Kernel Software Suite [15.1X54-D25]
JUNOS Packet Forwarding Engine Support (acx5k) [15.1X54-D25]
JUNOS Enterprise Software Suite [15.1X54-D25]
JUNOS Routing Software Suite [15.1X54-D25]
JUNOS py-base-i386 [15.1X54-D25]
JUNOS Host Software [15.1X54-D25]
```

**Meaning** The ZTD process is complete and the Junos OS is upgraded to the desired version, in this case 15.1X54-D25.

### Verifying the Device Template Deployment from Junos Space

**Purpose** Verify that the device template is deployed on the ACX router.

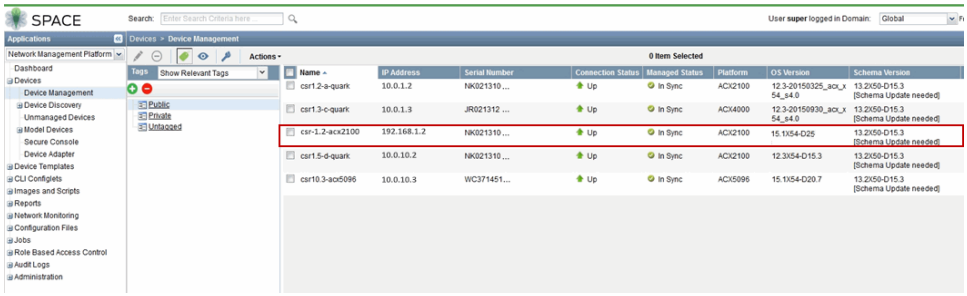
NOTE: This procedure is required to verify only one touch deployment.

**Action**

1. in Junos Space, navigate to **Devices > Device Management**.
2. Find the appropriate device in the list and verify that the Managed Status column indicates In Sync.

   The following display shows devices and their management status.

Figure 9: Device Management Page




3. Additionally, you can SSH to the ACX router, enter operational mode, and execute the **show configuration** command to verify that the ACX router has the correct configuration.

**Meaning**

The Device Management page indicates that the Managed Status of the new ACX router In Sync, confirming the device template was successfully deployed to the ACX router.

**Related Documentation**

- Customer Use Cases for ACX Series Routers on page 5
- ACX Autoinstallation Overview on page 8
- Deployment Methods for ACX Routers on page 12
- Example: Deploying ACX Routers Using the ZTD Pull Method on page 21
- ACX Deployment Scripts on page 77

CHAPTER 3

# Appendix



## ACX Deployment Scripts

The following scripts are used to deploy ACX routers as part of the ZTD Push and Pull methods.



### Phase 0 Script

```
version 1.1;
/* ------------------------------------------------------------------- */
/* This program performs Zero touch provisioning for the ACX platform */
/* It will load global configuration template from a remote location  */
/* specified in the original configuration file loaded as a result    */
/* of the autoconfiguration process.                                  */
/* Version 1.0 User1  user1@example.com                       */
/* ------------------------------------------------------------------- */
/* XML namespaces                                                     */
/* ------------------------------------------------------------------- */
/* Juniper */
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
ns exsl extension = "http://exslt.org/common";
/* EXSLT */
ns str extension = "http://exslt.org/strings";
/* private namespace for this script */
ns ztp_script = "http://xml.juniper.com/ztp_script/1.0";
import '../import/junos.xsl';
/* ------------------------------------------------------------------- */
/* Constants                                                          */
/* ------------------------------------------------------------------- */
var $APPNAME = 'ztp_script[' _ $junos-context/pid _ ']';
var $JNPR_ZTP_SCRIPT = "ZTP Phase 0 SCRIPT";
var $SYSLOG = 'user.info';
var $TMPDIR = '/var/tmp';
var $CODEDIR = '/var/tmp';
var $ZTP_GROUP_NAME = "GR-ZTP";
var $ZTP_GROUP_STAGE_0 = "GR-ZTP-STAGE-0";
```

```
var $ZTP_GROUP_STAGE_1 = "GR-ZTP-STAGE-1";
var $ZTP_GROUP_STAGE_2 = "GR-ZTP-STAGE-2";
var $ZTP_CODE_MACRO_NAME = "CODE";
var $ZTP_CONFIG_MACRO_NAME = "CONFIG";
var $ZTP_CONFIG = "CONFIG";
var $ZTP_LOCKFILE = '/tmp/ztp_script.lock';
/* ------------------------------------------------------------------- */
/* Global variables */
/* ------------------------------------------------------------------- */
var $jnx = jcs:open();
/* ------------------------------------------------------------------- */
/* MAIN */
/* ------------------------------------------------------------------- */
match / {
    if( not( $jnx )) {
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": ERROR: unable to connect to
Junos API");
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: unable to connect to
 Junos API");
        expr jcs:output( $JNPR_ZTP_SCRIPT _ ": ERROR: unable to connect to
Junos API");
        terminate;
    }
    var $running = ztp_script:only_once();
    if( $running ) {
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": process already running,
backing off" );
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": process already running,
backing off" );
       expr jcs:output( $JNPR_ZTP_SCRIPT _ ": process already running, backing
 off" );
        terminate;
    }
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": ZTP phase 0 BEGIN" );
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ZTP phase 0 BEGIN" );
    /*-----------------------------------------------------------------*/
    /*      PHASE-0: LOAD GLOBAL CSR TEMPLATE                          */
    /*-----------------------------------------------------------------*/
    /* Check if configuration for the next ZTP PHASE exists */
   if ( not( ztp_script:ztp_grp_exists($ZTP_GROUP_NAME, $ZTP_CONFIG_MACRO_NAME)
 ) ) {
   expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Configuration file name for global
 template (phase 0) not specified. Stop at this point" );
    var $die = ztp_script:terminate();
    }
    var $get = <get-configuration> {
         <configuration> {
            <groups> {
               <name> $ZTP_GROUP_NAME;
            }
        }
    }
    var $got = jcs:execute( $jnx, $get );
    var $got_copy = $got;
    /* global ztp parameters for access node */
    var $ztp_config_file =
$got/groups[name=$ZTP_GROUP_NAME]/apply-macro[name=$ZTP_CONFIG_MACRO_NAME]/data[name=$ZTP_CONFIG]/value;

    /*-----------------------------------------------------------*/
    /*  LOAD NEW ZTP PARAMETERS                                  */
    /*-----------------------------------------------------------*/
```

```
    if (not (ztp_script:load_config($ztp_config_file))) {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: Global configuration
 template was not loaded. File not found or bad configuration");
        var $die = ztp_script:terminate();
    } else {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ZTP Global configuration
template loaded" );
    }
    /*--------------------------------------------------------------------*/
    /*  Deactivate configuration group for PHASE-0                        */
    /*  Activate   configuration group for PHASE-1 if exists              */
    /*--------------------------------------------------------------------*/
    var $options_s0 := {
        <commit-options> {
            <log> "Enabling ZTP PHASE-1";
        }
    }
    var $change_s0 = {
        <configuration> {
            <apply-groups delete = "delete"> $ZTP_GROUP_STAGE_0;
            if ( ztp_script:ztp_grp_exists($ZTP_GROUP_STAGE_1) ) {
                <apply-groups> $ZTP_GROUP_STAGE_1;
            } else {
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Group name "
_$ZTP_GROUP_STAGE_1 _ " with PHASE-1 script name and location NOT FOUND in
global template. STOP ZTP AT THIS POINT" );
                var $die = ztp_script:terminate();
            }
        }
    }
    /* Load configuration */
    var $results_stage_0 := { call jcs:load-configuration( $action="merge",
$commit-options=$options_s0, $configuration=$change_s0, $connection = $jnx );
 }
    if ($results_stage_0//xnm:warning) {
        for-each ($results_stage_0//xnm:warning) {
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": commit warning: " _
message );
        }
    }
    if ($results_stage_0//xnm:error) {
        for-each ($results_stage_0//xnm:error) {
           expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": commit error: " _ message
 );
        }
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ZTP PHASE 0 Script failed."
 );
        var $die = ztp_script:terminate();
    } else {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ZTP Phase 0 completed
SUCCESSFULLY" );
    }
    var $die = ztp_script:terminate();
}
/* ------------------------------------------------------------------- */
/*   FUNCTION load_config LOADS CONFIGURATION FROM FILE                */
/* ------------------------------------------------------------------- */
function ztp_script:load_config($config_url, $action = "merge") {
    mvar $load_config = true();
    var $phrase = {
        if ($action == "merge") {
```

```
            expr "merged to existed configuration";
        } else {
            if ($action == "replace") {
                expr " and replace existed statements";
            } else {
                expr " be merged (default action) to existed configuration";
            }
        }
    }
    var $new_action = {
        if (($action == "merge") or ($action == "replace")) {
            expr $action;
        } else {
            expr "merge";
        }
    }
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Config file: " _ $config_url _
" will be loaded and " _ $phrase  );
    /* lock the config */
    var $lock = <lock-configuration>;
    var $did_lock = jcs:execute( $jnx, $lock );
    if ( $did_lock//self::xnm:error ) {
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": ERROR: unable to lock config"
 );
       expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: unable to lock config"
 );
        set $load_config = false();
    } else {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Got config lock" );
    }
    if ($load_config) {
        set $load_config = false();
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Processing config url " _
$config_url );
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": Processing config url " _
$config_url );
        /* load in new config */
        var $do_load = <load-configuration action="merge" url=$config_url
format="text">;
        var $did_load = jcs:execute( $jnx, $do_load );
        if( not( $did_load/load-success )) {
            expr jcs:syslog( $SYSLOG, $APPNAME _ ": ERROR: unable to load
config " _ $config_url );
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: unable to load
config " _ $config_url );
        } else {
           expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Configuration was loaded"
 );
            var $commit = <commit-configuration> {
                <full>;
                <synchronize>;
                <force-synchronize>;
                <log> "Initial config load";
            }
            var $did_commit = jcs:execute( $jnx, $commit );
            if ( $did_commit//self::xnm:error ) {
               expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: Commit failed"
 );
                expr jcs:syslog( $SYSLOG, $APPNAME _ ": ERROR: Commit failed"
 );
            } else {
```

```
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Configuration was
committed" );
                /*
                * reboot here
                * var $reboot := jcs:execute( $jnx, 'request-reboot' );
                */
                var $unlock = <unlock-configuration>;
                var $did_unlock = jcs:execute( $jnx, $unlock );
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": release config lock"
 );
                set $load_config = true();
            }
        }
    } /* end of foreach config_url */
    if (not($load_config)) {
        /* if we make it here, we failed to load the config */
        var $unlock = <unlock-configuration>;
        var $did_unlock = jcs:execute( $jnx, $unlock );
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": release config lock" );
    }
    result $load_config;
}
/*
------------------------------------------------------------------------------
 */
/* FUNCTION ztp_grp_exists VERIFIES IF  configuration group name and apply-macro
 name*/
/* EXIST IN THE CONFIGURATION and RETURNES True or False
/*
------------------------------------------------------------------------------
 */
function ztp_script:ztp_grp_exists($group_name, $macro_name = "N/A") {
    var $get_grp = <get-configuration> {
         <configuration> {
            <groups> {
                <name>;
            }
        }
    }
    /* geting variables from apply-macro */
    var $got_grp = jcs:execute( $jnx, $get_grp );
    mvar $grp_flag = "skip";
    for-each ($got_grp/groups/name) {
        set $grp_flag = {
            if  ((../name!=$group_name) and ($grp_flag!="exists")) {
                expr "skip";
            } else {
                expr "exists";
            } /* End if */
        }
    }
    /* Verification for the apply-macro */
    if ($grp_flag != "exists") {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Can't find a group " _
$group_name);
        result false();
    } else {
       expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Found a group " _ $group_name);

        if ( $macro_name!="N/A") {
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Looking for apply-macro
```

```
 " _ $macro_name _ " in group " _ $group_name );
            /* Look for apply-macro with specified name */
            var $get_mcr = <get-configuration> {
                <configuration> {
                    <groups> {
                        <name> $group_name;
                    }
                }
            }
            /* getting variables from apply-macro */
            var $got_mcr = jcs:execute( $jnx, $get_mcr );
            mvar $mcr_flag = "skip";
            for-each ($got_mcr/groups[name=$group_name]/apply-macro/name) {
                set $mcr_flag = {
                    if  ((../name!=$macro_name) and ($mcr_flag!="exists")) {
                        expr "skip";
                    } else {
                        expr "exists";
                    } /* End if */
                }
            }
            /* Verification for the group */
            if ($mcr_flag != "exists") {
              expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Can't find apply-macro
 " _ $macro_name );
                result false();
            } else {
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Found apply-macro "
 _ $macro_name );
                result true();
            }
        } else {
            result true();
        }
    }
}
/* ------------------------------------------------------------------------ */
/* Helper routines
/* ------------------------------------------------------------------------ */
function ztp_script:file-exists( $filename ) {
    var $ls_file = <file-list> { <path> $filename; }
    var $ls_got = jcs:execute( $jnx, $ls_file );
    var $retval = boolean( $ls_got//file-information );
    result $retval;
}
function ztp_script:file-delete( $filename ) {
    var $do_rm = <file-delete> { <path> $filename; }
    var $did_rm = jcs:execute( $jnx, $do_rm );
    /* @@@ trap error */
    result true();
}
function ztp_script:only_once() {
    if( ztp_script:file-exists( $ZTP_LOCKFILE )) {
        result true();
    } else {
        var $do_lock = <file-put> {
             <filename> $ZTP_LOCKFILE;
             <encoding> 'ascii';
             <file-contents> 'locked';
        }
    var $did_lock = jcs:execute( $jnx, $do_lock );
```

```
    result false();
    }
}
function ztp_script:terminate() {
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": SCRIPT-TERMINATE" );
    var $rm_lock = ztp_script:file-delete( $ZTP_LOCKFILE );
    terminate;
}
```

## Phase 1 Script

```
version 1.1;
/* ------------------------------------------------------------------- */
/* This script performs PHASE-1 of the zero touch provisioning for    */
/* the ACX platform. Script used for provisioning of the Layer 2 oam  */
/* plane which enables management access to the access node on its    */
/* interface. At the end of PHASE-1 Script will enable configuration  */
/* for the following elements:                                        */
/* 1. IRB Interface                                                   */
/* 2. Static IP configuration copied from Dynamic IP settings of the  */
/*    DHCP Client                                                     */
/* 3. OAM VLAN                                                        */
/* 4. OAM Bridge Domain                                               */
/* 5. Core facing interfaces (NNI)                                    */
/* 6. Generates a unique host name of the node                        */
/* 7. Deletes DHCP client configuration                               */
/* 8. Enables Phase 2 of the ZTP process                              */
/* ------------------------------------------------------------------*/
/* Version 1.0 User1 user1@example.com                      */
/* ------------------------------------------------------------------- */
/* XML namespaces*/
/* Juniper */
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
ns exsl extension = "http://exslt.org/common";
/* EXSLT */
ns str extension = "http://exslt.org/strings";
/* private namespace for this script */
ns ztp_script = "http://xml.juniper.com/ztp_script/1.0";
import '../import/junos.xsl';
version 1.1;
/* ------------------------------------------------------------------- */
/* This program performs Zero touch provisioning for the ACX platform */
/* It will load global configuration template from a remote location  */
/* specified in the original configuration file loaded as a result    */
/* of the autoconfiguration process.                                  */
/* */
/* Version 1.1 Vasily Mukhin                                      */
/* Based on jctyztp script by Jeremy Schulman and Brian Sherwood      */
/* */
/* ------------------------------------------------------------------- */
/* XML namespaces */
/* ------------------------------------------------------------------- */
/* Juniper */
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
ns exsl extension = "http://exslt.org/common";
/* EXSLT */
```

```
ns str extension = "http://exslt.org/strings";
/* private namespace for this script */
ns ztp_script = "http://xml.juniper.com/ztp_script/1.0";
import '../import/junos.xsl';
/* ------------------------------------------------------------------------ */
/* Constants                                                                */
/* ------------------------------------------------------------------------ */
var $APPNAME = 'ztp_script[' _ $junos-context/pid _ ']';
var $JNPR_ZTP_SCRIPT = "ZTP Phase 1 SCRIPT";
var $SYSLOG = 'user.info';
var $TMPDIR = '/var/tmp';
var $CODEDIR = '/var/tmp';
var $ZTP_GROUP_PLATFORM = "GR-ZTP-PLATFORM";
var $ZTP_GROUP_BOX = "GR-ZTP-BOX";
var $ZTP_CODE_MACRO_NAME = "CODE";
var $ZTP_CONFIG_MACRO_NAME = "CONFIG";
var $ZTP_MACRO_NAME_PREFIX = "ZTP-";
var $ZTP_GROUP_NNI_TAG = "GR-NNI-TAG";
var $ZTP_GROUP_AGG_NNI_TAG = "GR-AGG-INTF-TAG";
var $ZTP_GROUP_STAGE_1 = "GR-ZTP-STAGE-1";
var $ZTP_GROUP_STAGE_2 = "GR-ZTP-STAGE-2";
var $ZTP_HOSTNAME = "HOST_NAME";
var $ZTP_BD_OAM = "BD-ZTP-OAM";
var $ZTP_BD_BOOT = "BD-ZTP-BOOTP";
var $ZTP_OAM_VLAN = "OAM_VLAN";
var $ZTP_LOCKFILE = '/tmp/ztp_script.lock';
var $ZTP_IP_VLAN = "1";
/* ------------------------------------------------------------------------ */
/* Global variables                                                         */
/* ------------------------------------------------------------------------ */
var $jnx = jcs:open();
/* ------------------------------------------------------------------------ */
/* MAIN                                                                     */
/* ------------------------------------------------------------------------ */
match / {
    if( not( $jnx )) {
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": ERROR: unable to connect to
Junos API");
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: unable to connect to
 Junos API");
        expr jcs:output( $JNPR_ZTP_SCRIPT _ ": ERROR: unable to connect to
Junos API");
        terminate;
    }
    var $running = ztp_script:only_once();
    if( $running ) {
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": process already running,
backing off" );
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": process already running,
backing off" );
       expr jcs:output( $JNPR_ZTP_SCRIPT _ ": process already running, backing
 off" );
        terminate;
    }
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": ZTP Phase 1 BEGIN" );
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ZTP Phase 1 BEGIN" );
    /*-------------------------------------------------------------*/
    /*  DEACTIVATE PHASE-1 Configuration group                     */
    /*-------------------------------------------------------------*/
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Deactivating event-options for
ZTP Phase 1 script..." );
```

```
    var $options_0 := {
        <commit-options> {
            <log> "Deactivating configuration group for Phase 1";
        }
    }
    var $change_0 = {
        <configuration> {
            <apply-groups delete="delete"> $ZTP_GROUP_STAGE_1;
        }
    }
    /* Load new configuration */
    var $results_stage_0 := { call jcs:load-configuration( $action="merge",
$commit-options=$options_0, $configuration=$change_0, $connection = $jnx );
}
    if ($results_stage_0//xnm:warning) {
        for-each ($results_stage_0//xnm:warning) {
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": commit warning: " _
message );
        }
    }
    if ($results_stage_0//xnm:error) {
        for-each ($results_stage_0//xnm:error) {
           expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": commit error: " _ message
 );
        }
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Commit failed." );
        var $die = ztp_script:terminate();
    } else {
       expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Configuration for event-options
 for ZTP Phase 1 was disabled" );
    }
    mvar $nRollback = 1;

/*-------------------------------------------------------------------------------------*/

    /* Verify that configuration group and apply-macro with platform specific
 ZTP parameters */
    /* exist in the global configuration template
                */

/*-------------------------------------------------------------------------------------*/

    var $serial_no = ztp_script:get_serial_number();
    var $platform = ztp_script:get_platform();
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Looking for " _ $platform _ "
specific parameters." );
   expr jcs:syslog( $SYSLOG, $APPNAME _ "Looking for " _ $platform _ " specific
 parameters." );
    if ( not( ztp_script:ztp_grp_exists($ZTP_GROUP_PLATFORM,
$ZTP_MACRO_NAME_PREFIX _ $platform) ) ) {
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: Failed to find platform
specific ZTP parameters in global template." );
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Note: Make sure that apply-macro
 " _ $ZTP_MACRO_NAME_PREFIX _ $platform _ " exists under " _ $ZTP_GROUP_PLATFORM
 _ " group in global template." );
    var $ztd_rollback = ztp_script:ztp_rollback_phase0();
 expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Script terminated." );
    var $die = ztp_script:terminate();
    }

    /* Creating static configuration and enabling management access to the
```

```
node */
    /*
        0. Read parameters assigned dynamically via DHCP:
           - ip address and mask a.b.c.d/n
           - default route
        1. Create host-name: csr<c.d>-<platform>)
        2. Creates OAM VLAN on NNIs:
            - NNIs are listed under apply-macros (platform specific list)
        3. Creates IRB interfaces
            - Copies address assigned by DHCP server to IRB
        4. Deletes DHCP-client configuration
        5. Creates OAM Bridge-domain and placess following interfaces into it

            - NNI IFLs
            - IRB
        6. Enables VSTP and LLDP protocols on NNI interfaces
    */
    var $results_step_1 = ztp_script:set_dhcp_to_static($platform);
    if ($results_step_1//xnm:warning) {
        for-each ($results_step_1//xnm:warning) {
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": set_dhcp_to_static()
commit warning: " _ message );
        }
    }
    if ($results_step_1//xnm:error) {
        for-each ($results_step_1//xnm:error) {
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": dhcp_to_static() commit
 error: " _ message );
        }
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ " ERROR: Commit failed." );
  expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Note: Verify parametrs in global
conviguration template." );
  var $ztd_rollback = ztp_script:ztp_rollback_phase0();
  expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Script terminated." );
  var $die = ztp_script:terminate();
    } else {
       expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Platform specific configuration
 committed successfully." );
        set $nRollback = $nRollback + 1;
    }
    /*-----------------------------------------*/
    /* Looking for box specific ZTP parameters */
    /*-----------------------------------------*/
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Looking for box (S/N: " _
$serial_no _ ") specific parameters." );
   expr jcs:syslog( $SYSLOG, $APPNAME _ "Looking for box (S/N: " _ $serial_no
 _ ") specific parameters." );
    if ( not( ztp_script:ztp_grp_exists($ZTP_GROUP_BOX, $serial_no) ) ) {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Box specific configuration
for S/N: " _ $serial_no _ " was not found in global template. Skip this step."
 );
    }
    else {
        /* Configuring box specific parameters (fxp0 and host name) */
        var $results_step_2 = ztp_script:set_box_specific_config($platform,
$serial_no);
        if ($results_step_2//xnm:warning) {
            for-each ($results//xnm:warning) {
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ":
set_box_specific_config() commit warning: " _ message );
            }
```

```
        }
        if ($results_step_2//xnm:error) {
            for-each ($results_step_1//xnm:error) {
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ":
set_box_specific_config() commit error: " _ message );
            }
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: Failed to commit
 box specific parameters." );
          expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Skip this step and continue
 ZTP process." );
            var $ztd_rollback = ztp_script:rollback_cfg(0);
            /* var $die = ztp_script:terminate(); */
        } else {
          expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Box specific configuration
 committed successfully." );
            set $nRollback = $nRollback + 1;
        }
    }
    /*-------------------------------------------------------------------*/
    /*  DELETE Box specific configuration if exists                      */
    /*  ACTIVATE   PHASE 2 Configuration group if exists                 */
    /*-------------------------------------------------------------------*/
    var $options_1 := {
        <commit-options> {
            <log> "Enabling next ZTP Phase";
        }
    }
    var $change_1 = {
        <configuration> {
            if ( ztp_script:ztp_grp_exists($ZTP_GROUP_BOX) ) {
                <groups delete="delete"> {
                    <name> $ZTP_GROUP_BOX;
                    expr jcs:syslog( $SYSLOG, $APPNAME _ ": Deleting
configuration group " _ $ZTP_GROUP_BOX );
                    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Deleting
configuration group " _ $ZTP_GROUP_BOX );
                }
            }
            if ( ztp_script:ztp_grp_exists($ZTP_GROUP_STAGE_2) )  {
                <apply-groups> $ZTP_GROUP_STAGE_2;
            } else {
                expr jcs:syslog( $SYSLOG, $APPNAME _ ": WARNING: No
configuration group for Phase 2 found in global template. STOP at this point.");

                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": WARNING: No
configuration group for Phase 2 found in global template. STOP at this point.");

            }
        }
    }
    /* Load new configuration */
    var $results_stage_1 := { call jcs:load-configuration( $action="merge",
$commit-options=$options_1, $configuration=$change_1, $connection = $jnx );
}
    if ($results_stage_1//xnm:warning) {
        for-each ($results_stage_1//xnm:warning) {
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": commit warning: " _
message );
        }
    }
    if ($results_stage_1//xnm:error) {
```

```
        for-each ($results_stage_1//xnm:error) {
           expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": commit error: " _ message
 );
        }
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Commit failed. " );
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Rollback to beginning of
phase 1 and exiting ztp_script now." );
        var $ztd_rollback = ztp_script:rollback_cfg($nRollback);
        var $die = ztp_script:terminate();
    } else {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Configuration was committed"
 );
    }
   expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ZTP Phase 1 completed successfully"
 );
    var $die = ztp_script:terminate();
}
/*
------------------------------------------------------------------------------
 */
/* FUNCTION ztp_grp_exists VERIFIES IF  configuration group name and apply-macro
    */
/* name EXIST IN THE CONFIGURATION and RETURNES True or False
        */
/*
------------------------------------------------------------------------------
 */
function ztp_script:ztp_grp_exists($group_name, $macro_name = "N/A") {
    var $get_grp = <get-configuration> {
         <configuration> {
            <groups> {
                <name>;
            }
        }
    }
    /* getting variables from apply-macro */
    var $got_grp = jcs:execute( $jnx, $get_grp );
    mvar $grp_flag = "skip";
    for-each ($got_grp/groups/name) {
        set $grp_flag = {
            if  ((../name!=$group_name) and ($grp_flag!="exists")) {
                expr "skip";
            } else {
                expr "exists";
            } /* End if */
        }
    }
    /* Verification for the apply-macro */
    if ($grp_flag != "exists") {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Can't find a group " _
$group_name);
        result false();
    } else {
       expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Found a group " _ $group_name);

        if ( $macro_name!="N/A") {
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Looking for apply-macro
 " _ $macro_name _ " in group " _ $group_name );
            /* Look for apply-macro with specified name */
            var $get_mcr = <get-configuration> {
                <configuration> {
```

```
                    <groups> {
                        <name> $group_name;
                    }
                }
            }
            /* geting variables from apply-macro */
            var $got_mcr = jcs:execute( $jnx, $get_mcr );
            mvar $mcr_flag = "skip";
            for-each ($got_mcr/groups[name=$group_name]/apply-macro/name) {
                set $mcr_flag = {
                    if  ((../name!=$macro_name) and ($mcr_flag!="exists")) {
                        expr "skip";
                    } else {
                        expr "exists";
                    } /* End if */
                }
            }
            /* Verification for the group */
            if ($mcr_flag != "exists") {
              expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Can't find apply-macro
 " _ $macro_name );
                result false();
            } else {
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Found apply-macro "
 _ $macro_name );
                result true();
            }
        } else {
            result true();
        }
    }
}
/* ------------------------------------------------------------------- */
/* Function: remove_old_cfg - DELETES CONFIGURATION GROUPS             */
/* ------------------------------------------------------------------- */
function ztp_script:remove_old_cfg($group_name) {
   expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Removing old configuration group:
 " _ $group_name );
    expr jcs:syslog( $SYSLOG, $APPNAME _ "Removing old configuration group:
" _ $group_name );
    var $get = <get-configuration> {
         <configuration> {
            <groups> {
                <name>;
            }
        }
    }
    /* looking for the given group-name */
    var $got = jcs:execute( $jnx, $get );
    mvar $delete_flag = "skip";
    for-each ($got/groups/name) {
        set $delete_flag = {
            if  ((../name!=$group_name) and ($delete_flag!="delete")) {
                expr "skip";
            } else {
                expr "delete";
            } /* End if */
        }
    }
    var $options := {
         <commit-options> {
```

```
            <log> "deleting configuration group " _ $group_name;;
        }
    }
    var $change = {
        <configuration> {
            <groups delete="delete"> {
                <name> $group_name;
            }
        }
    }
    /* Deleting config for the group */
        if ($delete_flag == "delete") {
            var $results := { call jcs:load-configuration($action="merge",
$commit-options=$options, $configuration=$change, $connection = $jnx ); }
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": group " _ $group_name _
 " was deleted" );
        } else {
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Group: " _ $group_name
_ " was not found. Nothing to delete ");
        }
}
/* ------------------------------------------------------------------- */
/* Function: set_dhcp_to_static - GENERATES NEW STATIC CONFIGURATION  */
/* ------------------------------------------------------------------- */
function ztp_script:set_dhcp_to_static($platform) {
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Retrieve DHCP client bindings"
);
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": Retrieve DHCP client bindings" );

     /* GET ZTP VLAN-ID */
    var $get = <get-configuration> {
         <configuration> {
            <groups> {
               <name> $ZTP_GROUP_PLATFORM;
               <apply-macro> {
                  <name> $ZTP_MACRO_NAME_PREFIX _ $platform;
                }
            }
        }
    }
    /* getting variables from apply-macro */
    var $got = jcs:execute( $jnx, $get );
    mvar $ztp_oam_vlan = "2";
    mvar $ztp_host_prefix = "csr";
    for-each
($got/groups[name=$ZTP_GROUP_PLATFORM]/apply-macro[name=$ZTP_MACRO_NAME_PREFIX
 _ $platform]/data/name) {
        if (../name == $ZTP_OAM_VLAN ) {
            set $ztp_oam_vlan = ../value;
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Found vlan-id for OAM
plane: " _ ../value );
        }
        if (../name == $ZTP_HOSTNAME ) {
            set $ztp_host_prefix = ../value;
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Found host name prefix:
 " _ ../value );
        }
    }
     /* GET VALUES OF THE DHCP OPTIONS */
    var $get_dhcp_detail = <get-dhcp-client-binding-information> {
                                <detail>;
```

```
                            }
    var $dhcp_bind := jcs:execute( $jnx, $get_dhcp_detail );
    /* - GET ZTP INTERFACE - */
    var $ztp_interface = $dhcp_bind/dhcp-binding/interface-name;
   expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": DHCP client runs on ZTP interface:
 " _ $ztp_interface );
    /* - GET ROUTER IP ADDRESS - */
    var $ztp_router =
$dhcp_bind/dhcp-binding/dhcp-option-table/dhcp-option[dhcp-option-name="router"]/dhcp-option-value;

    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Found DHCP Option for router: "
 _ $ztp_router );
    var $post_str = {
        if ( contains($ztp_router, '[ ')) {
            expr substring-after($ztp_router, '[ ');
        } else {
            expr $ztp_router;
        }
    }
    /* - GET NEXT-HOP FOR DEFAULT ROUTE - */
    var $ztp_next_hop = {
        if ( contains($post_str, ' ')) {
            expr substring-before($post_str, ' ');
        } else {
            expr $post_str;
        }
    }
   expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Derived next-hop: " _ $ztp_next_hop
 );
    /* Looling for real ip-address assigned to dhcp interface */
    var $get_interface_terse = <get-interface-information> {
                               <terse>;
                               <interface-name> $ztp_interface;
                            }
    var $interface_info := jcs:execute( $jnx, $get_interface_terse );
    /* - GET DHCP IP ADDRESS - */
    var $irb_ip_address =
$interface_info/logical-interface/address-family[address-family-name="inet"]/interface-address/ifa-local;

    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": DHCP ip address found: " _
$irb_ip_address);
    /* Create configuration with static IP */
    var $options := {
         <commit-options> {
            <log> "setting irb interface";
        }
    }
    /* - CREATE CONFIGURATION - */
    /*
        - Deletes DHCP-client configuration
        - Creates OAM Bridge-domain
        - Configures OAM vlan-bridge unit on ZTP interface
        - Moves OAM unit into bridge domain
        - Creates irb
        - Copies DHCP-client IP-address to static irb interface
    */
    var $ztp_device = substring-before($ztp_interface, '.');
    var $ztp_device_unit = substring-after($ztp_interface,'.');
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Parsing ZTP Interface: ifd=" _
$ztp_device _ " unit=" _ $ztp_device_unit );
    var $irb_unit_num = $ztp_oam_vlan;
```

```
    var $ip_3_4 =
substring-after(substring-after(substring-before($irb_ip_address,'/'),'.'),'.');

    var $change = {
        <configuration> {
            <interfaces> {
                <interface> {
                    call emit-deactivate-dhcp-client($platform, $ztp_device,
 $ztp_device_unit);
                }
                <interface> {
                    <name> $ztp_device;
                    <flexible-vlan-tagging>;
                    <encapsulation> "flexible-ethernet-services";
                    <unit> {
                        <name> $ztp_oam_vlan;
                        <encapsulation> "vlan-bridge";
                        <vlan-id> $ztp_oam_vlan;
                    }
                }
                <interface> {
                    <name> "irb";
                    <unit> {
                        <name> $irb_unit_num;
                        <family> {
                            <inet> {
                                <address> $irb_ip_address;
                            }
                        }
                    }
                }
                call create-nni-interfaces($platform, $ztp_oam_vlan, $got);
            }
            call emit-bridge-domain($platform, $irb_unit_num, $ztp_device,
$ztp_oam_vlan, $got);
            <routing-options> {
                <static> {
                    <route> {
                        <name> "0.0.0.0/0";
                        <next-hop> $ztp_next_hop;
                    }
                }
            }
            <protocols> {
                <vstp> {
                    for-each
($got/groups[name=$ZTP_GROUP_PLATFORM]/apply-macro[name=$ZTP_MACRO_NAME_PREFIX
 _ $platform]/data/name) {
                        if (contains(../name,"NNI" )) {
                            <interface> {
                                <name> ../value;
                            }
                            <vlan> {
                                <name> $ztp_oam_vlan;
                            }
                        }
                    }
                }
                <lldp> {
                    for-each
($got/groups[name=$ZTP_GROUP_PLATFORM]/apply-macro[name=$ZTP_MACRO_NAME_PREFIX
```

```
 _ $platform]/data/name) {
                        if (contains(../name,"NNI" )) {
                            <interface> {
                                <name> ../value;
                            }
                        }
                    }
                }
            }
            call create-re0-group($platform, $ip_3_4, $ztp_host_prefix);
        }
    }
    /* Load configuration for group re0 */
    var $results := { call jcs:load-configuration( $action="merge",
$commit-options=$options, $configuration=$change, $connection = $jnx ); }
    result $results;
}
/*-------------------------------------------------------------------------*/
/*  Template: create-re0-group - CREATES re0 group and a host-name    */
/*-------------------------------------------------------------------------*/
template create-re0-group($platform, $ip_3_4, $ztp_host_prefix) {
    if (($platform=="acx5096") or ($platform=="acx5048")) {
        <groups> {
            <name> "member0";
            <system> {
                <host-name> $ztp_host_prefix _ $ip_3_4 _ "-" _ $platform;
            }
        }
        <apply-groups> "member0";
    } else {
        <groups> {
            <name> "re0";
            <system> {
                <host-name> $ztp_host_prefix _ $ip_3_4 _ "-" _ $platform;
            }
        }
        <apply-groups> "re0";
    }
}
/* ------------------------------------------------------------------------ */
/*  Template: create-nni-interfaces - CONFIGURES CORE-FACING INTERFACES */
/* ------------------------------------------------------------------------ */
template create-nni-interfaces($platform, $ztp_oam_vlan, $got) {
    for-each
($got/groups[name=$ZTP_GROUP_PLATFORM]/apply-macro[name=$ZTP_MACRO_NAME_PREFIX
 _ $platform]/data/name) {
        if ( contains(../name,"NNI" )){
          expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Configuring NNI interface:
 " _ ../value);
            <interface> {
                <name> ../value;
                <flexible-vlan-tagging>;
                <encapsulation> "flexible-ethernet-services";
                <apply-groups> $ZTP_GROUP_NNI_TAG;
                <native-vlan-id> $ztp_oam_vlan;
                <unit> {
                    <name> $ztp_oam_vlan;
                    <description> "OAM VLAN ENABLES ZTP AND NMS ACCESS";
                    <encapsulation> "vlan-bridge";
                    <vlan-id> $ztp_oam_vlan;
                }
```

```
            }
        }
    }
}
/*------------------------------------------------------------------------ */
/* Template: emit-nni-interfaces - ADDS INTERFACES TO OSPF CONFIGURATION */
/*------------------------------------------------------------------------ */
template emit-nni-interfaces($platform, $got) {
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Configuring OSPF Interfaces ");

    for-each
($got/groups[name=$ZTP_GROUP_PLATFORM]/apply-macro[name=$ZTP_MACRO_NAME_PREFIX
 _ $platform]/data/name) {
        if (contains(../name,"NNI" )) {
            <interface> {
                <name> ../value _ ".0";
            }
        }
    }
}
/*------------------------------------------------------------------- */
/*  Template: emit-deactivate-dhcp-client DEACTIVATE DHCP CLIENT    */
/*------------------------------------------------------------------- */
template emit-deactivate-dhcp-client($platform, $ztp_device, $ztp_device_unit)
 {
    if (($platform=="acx5096") or ($platform=="acx5048")) {
        <name> $ztp_device;
        <flexible-vlan-tagging>;
        <encapsulation> "flexible-ethernet-services";
        <unit> {
            <name> $ztp_device_unit;
            <vlan-id> $ZTP_IP_VLAN;
            <family> {
                <inet> {
                    <dhcp inactive="inactive">;
                }
            }
        }
    } else {
        <name> $ztp_device;
        <flexible-vlan-tagging>;
        <encapsulation> "flexible-ethernet-services";
        <unit> {
            <name> $ztp_device_unit;
            <vlan-id> $ZTP_IP_VLAN;
            <family> {
                <inet> {
                    <dhcp-client inactive="inactive">;
                }
            }
        }
    }
}
/*-------------------------------------------------------------------*/
/*  Template: emit-bridge-domain - CONFIGURES BRIDGE DOMAIN WITH IRB  */
/*-------------------------------------------------------------------*/
template emit-bridge-domain($platform, $irb_unit_num, $ztp_device,
$ztp_oam_vlan, $got) {
    if (($platform=="acx5096") or ($platform=="acx5048")) {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Configuring bridge-domain
(vlans): " _ $ZTP_BD_OAM);
```

```
        <vlans> {
            <vlan> {
                <name> $ZTP_BD_OAM;
                <vlan-id> $ztp_oam_vlan;
                <l3-interface> "irb." _ $irb_unit_num;
                <interface> $ztp_device _ "." _ $ztp_oam_vlan;
                for-each
($got/groups[name=$ZTP_GROUP_PLATFORM]/apply-macro[name=$ZTP_MACRO_NAME_PREFIX
 _ $platform]/data/name) {
                    if (contains(../name,"NNI" )) {
                        <interface> ../value _ "." _ $ztp_oam_vlan;
                    }
                }
            }
        }
    } else {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Configuring bridge-domain:
" _ $ZTP_BD_OAM);
        <bridge-domains> {
            <domain> {
                <name> $ZTP_BD_OAM;
                <vlan-id> $ztp_oam_vlan;
                <routing-interface> "irb." _ $irb_unit_num;
                <interface> $ztp_device _ "." _ $ztp_oam_vlan;
                for-each
($got/groups[name=$ZTP_GROUP_PLATFORM]/apply-macro[name=$ZTP_MACRO_NAME_PREFIX
 _ $platform]/data/name) {
                    if (contains(../name,"NNI" )) {
                        <interface> ../value _ "." _ $ztp_oam_vlan;
                    }
                }
            }
        }
    }
}
/* ------------------------------------------------------------------- */
/* Function: get_platform - RETURNS PLATFORM TYPE                      */
/* ------------------------------------------------------------------- */
function ztp_script:get_platform() {
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Retrieve type of platform" );
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": Retrieve Platform Type" );
     /* get platform */
    var $chassis_software := jcs:execute( $jnx, 'get-software-information' );

    var $ztp_platform = {
        if ( $chassis_software//product-model) {
            expr $chassis_software//product-model;
        } else {
            expr $chassis_software/software-information/product-model;
        }
    }
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": platform = " _ $ztp_platform );
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": platform = " _ $ztp_platform );
    result $ztp_platform;
}
/* ------------------------------------------------------------------- */
/* Function: get_serial_number - RETURNES SERIAL NUMBER OF THE NODE    */
/* ------------------------------------------------------------------- */
function ztp_script:get_serial_number() {
     expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Getting box Serial Number" );
     expr jcs:syslog( $SYSLOG, $APPNAME _ ": Getting box Serial Number" );
```

```
     /* get our serial number */
    var $chassis_hardware := jcs:execute( $jnx, 'get-chassis-inventory' );
    var $serial_no = $chassis_hardware/chassis/serial-number;
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": serial no = " _ $serial_no );
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": serial no = " _ $serial_no );
    result $serial_no;
}
/* ------------------------------------------------------------------------ */
/* Function: set_box_specific_config - CONFIGURES fxp0/em0 and host-name */
/* ------------------------------------------------------------------------ */
function ztp_script:set_box_specific_config($platform, $serial_no) {
/* get the apply-macro */
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Setting fxp0 and host-name for
lab boxes" );
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": Setting fxp0 and host-name for
lab boxes" );
    var $mng_if_name = {
        if (($platform=="acx5096") or ($platform=="acx5048")) {
            expr "em0";
        } else {
             expr "fxp0";
        } /* End If */
    }
    var $re_group_name = {
        if (($platform=="acx5096") or ($platform=="acx5048")) {
            expr "member0";
        } else {
             expr "re0";
        } /* End If */
    }

    var $get = <get-configuration> {
         <configuration> {
            <groups> {
               <name> $ZTP_GROUP_BOX;
               <apply-macro> {
                  <name> $serial_no;
                }
            }
        }
    }
    /* getting variables from apply-macro */
    var $got = jcs:execute( $jnx, $get );
    var $fxp_address =
$got/groups[name=$ZTP_GROUP_BOX]/apply-macro[name=$serial_no]/data[name='address']/value;

    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Current Platform = " _ $platform
 );
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": MNG interface name = " _
$mng_if_name );
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": " _ $mng_if_name _ " ip address
 = " _ $fxp_address );
    if
($got/groups[name=$ZTP_GROUP_BOX]/apply-macro[name=$serial_no]/data[name='host-name']/value)
 {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": New host-name = " _
$got/groups[name=$ZTP_GROUP_BOX]/apply-macro[name=$serial_no]/data[name='host-name']/value);

    }
   expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": RE0 Group name = " _ $re_group_name
 );
```

```
    /* Create configuration for group re0 */
    var $options := {
         <commit-options> {
            <log> "setting fxp0 and host-name. For lab only";
        }
    }
    var $unit_num = "0";
    var $change = {
        <configuration> {
            <groups> {
                <name> $re_group_name;
                <interfaces> {
                    <interface> {
                        <name> $mng_if_name;
                        <unit> {
                            <name> $unit_num;
                            <family> {
                                <inet> {
                                    <address> $fxp_address;
                                }
                            }
                        }
                    }
                }
                if
($got/groups[name=$ZTP_GROUP_BOX]/apply-macro[name=$serial_no]/data[name='host-name']/value){

                    <system> {
                         <host-name>
$got/groups[name=$ZTP_GROUP_BOX]/apply-macro[name=$serial_no]/data[name='host-name']/value;

                    }
                }
            }
            <apply-groups> $re_group_name;
        }
    }

    /* Load configuration for group re0 */
    var $results := { call jcs:load-configuration( $action="merge",
$commit-options=$options, $configuration=$change, $connection = $jnx ); }
    result $results;
} /* End of the set_box_specific_config function */
/* ------------------------------------------------------------------- */
/* Function rolback_cfg($nRollback) rollback configuration to         */
/* nRollback version                                                   */
/* ------------------------------------------------------------------- */
function ztp_script:rollback_cfg( $cfg_version=0, $rollback_log="Rollback
configuration" ) {
    var $rollback_options := {
        <commit-options> {
            <full>;
            <log> $rollback_log;
        }
    }
    var $rollback_configuration := { call jcs:load-configuration(
$commit-options = $rollback_options, $rollback = $cfg_version, $connection =
 $jnx ); }
}
/* ------------------------------------------------------------------- */
```

```
/* Function: ztp_rollback_phase0 - Rollbacks configuration to basic   */
/* configuration template of Phase 0                                  */
/* ------------------------------------------------------------------ */
function ztp_script:ztp_rollback_phase0() {
    var $lRollback = "ZTD Phase 0";
    mvar $nRollback = 0;
    mvar $bRollback = false();
    var $got_rollback := jcs:execute( $jnx, 'get-commit-information' );
    for-each ($got_rollback/commit-history/sequence-number) {
        if (((../client == "autoinstall") || (../log == $lRollback)) && ( not
 ($bRollback))) {
            set $bRollback = true();
            set $nRollback = ../sequence-number;
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Starting rollback " _
$nRollback _ " to ZTD phase 0" );
        }
    }
    var $rollback = ztp_script:rollback_cfg( $nRollback, $lRollback );
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": rollback " _ $nRollback _ "
completed successfully" );
    result $nRollback;
}
/* ------------------------------------------------------------------ */
/* Helper routines
/* ------------------------------------------------------------------ */
function ztp_script:file-exists( $filename ) {
    var $ls_file = <file-list> { <path> $filename; }
    var $ls_got = jcs:execute( $jnx, $ls_file );
    var $retval = boolean( $ls_got//file-information );
    result $retval;
}
function ztp_script:file-delete( $filename ) {
    var $do_rm = <file-delete> { <path> $filename; }
    var $did_rm = jcs:execute( $jnx, $do_rm );
    /* @@@ trap error */
    result true();
}
function ztp_script:only_once() {
    if( ztp_script:file-exists( $ZTP_LOCKFILE )) {
        result true();
    } else {
        var $do_lock = <file-put> {
             <filename> $ZTP_LOCKFILE;
             <encoding> 'ascii';
             <file-contents> 'locked';
        }
    var $did_lock = jcs:execute( $jnx, $do_lock );
    result false();
    }
}
function ztp_script:terminate() {
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": SCRIPT-TERMINATE" );
    var $rm_lock = ztp_script:file-delete( $ZTP_LOCKFILE );
    terminate;
}
```

## Phase 2 Script

```
version 1.1;
/* ------------------------------------------------------------------ */
```

```
/* This program performs Zero touch provisioning for the ACX platform */
/* Script enables workflow for pull or push ZTD model depending on    */
/* the parameter configured in global configuration template under    */
/* configuration group GR-ZTP-SCENARIOS:                              */
/* Pull Model:                                                        */
/*   - checks current sw image and initiate upgrade procedure         */
/*   - fetches for <S/N>.conf configuration file on FTP server and    */
/*     applies to the access node                                     */
/* Push Model:                                                        */
/*   - fetches for Space configlet file on FTP server and applies it  */
/*     to the access nodes                                            */
/*   - Calls home to Space network management platform                */
/*--------------------------------------------------------------------*/
/* Version 1.0 User1 user1@example.com                        */
/* ------------------------------------------------------------------ */
/* XML namespaces                                                     */
/* ------------------------------------------------------------------ */
/* Juniper */
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
ns exsl extension = "http://exslt.org/common";
/* EXSLT */
ns str extension = "http://exslt.org/strings";
/* private namespace for this script */
ns ztp_script = "http://xml.juniper.com/ztp_script/1.0";
import '../import/junos.xsl';
/* ------------------------------------------------------------------ */
/* Constants */
/* ------------------------------------------------------------------ */
var $APPNAME = 'ztp_script[' _ $junos-context/pid _ ']';
var $SYSLOG = 'user.info';
var $TMPDIR = '/var/tmp';
var $CODEDIR = '/var/tmp';
var $JNPR_ZTP_SCRIPT = "ZTP Phase 2 SCRIPT";
var $ZTP_GROUP_SCENARIOS = "GR-ZTP-SCENARIOS";
var $ZTP_SCENARIO = "SCENARIO-1";
var $ZTP_METHOD = "METHOD";
var $ZTP_GROUP_NAME = "GR-ZTP";
var $ZTP_CODE_MACRO_NAME = "CODE";
var $ZTP_GROUP_PLATFORM = "GR-ZTP-PLATFORM";
var $ZTP_GROUP_BOX = "GR-ZTP-BOX";
var $ZTP_GROUP_CALLHOME = "GR-ZTP-CALLHOME";
var $ZTP_CONFIG_MACRO_NAME = "CONFIG";
var $ZTP_CALLHOME_MACRO_NAME = "CALLHOME";
var $ZTP_MACRO_NAME_PREFIX = "ZTP-";
var $ZTP_BD_OAM = "BD-ZTP-OAM";
var $ZTP_OAM_VLAN = "OAM_VLAN";
var $ZTP_LOOPBACK = "Lo0";
var $ZTP_CONFIG = "CONFIG";
var $ZTP_SPACE_CONFIGLET = "SPACE-CONFIGLET";
var $ZTP_VERSION = "VERSION";
var $ZTP_IMAGE = "IMAGE";
var $ZTP_INACTIVE_FLAG = "INACTIVE";
var $ZTP_GROUP_STAGE_1 = "GR-ZTP-STAGE-1";
var $ZTP_GROUP_STAGE_2 = "GR-ZTP-STAGE-2";
var $ZTP_GROUP_STAGE_3 = "GR-ZTP-STAGE-3";
var $ZTP_GROUP_SPACE = "GR-ZTP-SPACE";
var $ZTP_LOCKFILE = '/tmp/ztp_script.lock';
var $PATTERN = "system";
var $ZTP_GROUP_NNI_TAG = "GR-NNI-TAG";
```

```
/* Global variables */
var $jnx = jcs:open();
/* MAIN */
match / {
    if( not( $jnx )) {
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": ERROR: unable to connect to
Junos API");
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: unable to connect to
 Junos API");
        expr jcs:output( $JNPR_ZTP_SCRIPT _ ": ERROR: unable to connect to
Junos API");
        terminate;
    }
    var $running = ztp_script:only_once();
    if( $running ) {
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": process already running,
backing off" );
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": process already running,
backing off" );
       expr jcs:output( $JNPR_ZTP_SCRIPT _ ": process already running, backing
 off" );
        terminate;
    }
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": ZTP Phase 2 BEGIN" );
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ZTP Phase 2 BEGIN" );
    /*--------------------------------------------------------------------*/
    /*     VERIFY WHAT MODEL IS USED - PULL or PUSH                       */
    /*--------------------------------------------------------------------*/
    var $ztp_method = ztp_script:ztp_get_method($ZTP_GROUP_SCENARIOS,
$ZTP_SCENARIO);
    if ( $ztp_method == "push" ) {
        /* Proceed with PUSH model work flow */
         expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ZTD Push model is in use."
 );
        /* Check if configuration for the next ZTP STAGE exists */
        if ( not( ztp_script:ztp_grp_exists($ZTP_GROUP_CALLHOME,
$ZTP_CALLHOME_MACRO_NAME) ) ) {
       expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": No configuration for the next
 ZTP phase exists. Stop at this point" );
        var $die = ztp_script:terminate();
        }
        var $get = <get-configuration> {
             <configuration> {
                <groups> {
                   <name> $ZTP_GROUP_CALLHOME;
                }
            }
        }
        var $got = jcs:execute( $jnx, $get );
       /*--------------------------------------------------------------------*/

        /*  Deactivate PHASE-2 Configuration group                         */

        /*     Note:   Next PHASE of ZTP process will be enabled after     */

        /*             Space NMS completes S/W upgrade and populates node  */

        /*             into NMS in its database                            */

       /*--------------------------------------------------------------------*/
```

```
        var $options_s2 := {
            <commit-options> {
                <log> "Disables ZTP Phase 2 on completion";
            }
        }
       expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Next ZTP phase will be enabled
 by Space after completion of the call home phase" );
        var $change_s2 = {
            <configuration> {
                <apply-groups delete = "delete"> $ZTP_GROUP_STAGE_2;
            }
        }
        /* Loading configuration */
       var $results_step_3 := { call jcs:load-configuration( $action="merge",
 $commit-options=$options_s2, $configuration=$change_s2, $connection = $jnx
); }
        if ($results_step_3//xnm:warning) {
            for-each ($results_step_3//xnm:warning) {
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": apply-group commit
warning: " _ message );
            }
        }
        if ($results_step_3//xnm:error) {
            for-each ($results_step_3//xnm:error) {
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: apply-group
commit error: " _ message );
            }
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: Script failed."
);
            var $die = ztp_script:terminate();
        }
        /* get location of the configlet file */
        mvar $ztp_config_src = " ";
        mvar $ztp_configlet = "space_configlet.conf";
        for-each
($got/groups[name=$ZTP_GROUP_CALLHOME]/apply-macro[name=$ZTP_CALLHOME_MACRO_NAME]/data/name)
 {
            if (../name == $ZTP_CONFIG ) {
                set $ztp_config_src = ../value;
            }
            if (../name == $ZTP_SPACE_CONFIGLET ) {
              if (( string-length(../value) > 2 ) && (not ( contains(../value,
 ' '')))) {
                    set $ztp_configlet = ../value;
                }
            }
        }
        /* Download configlet file */
        var $ztp_callhome_config = $ztp_config_src _ "/" _ $ztp_configlet;
       /*--------------------------------------------------------------------*/

        /*  LOAD CONFIGURATION FROM CONFIGLET                                 */

       /*--------------------------------------------------------------------*/

        if (not (ztp_script:load_config($ztp_callhome_config))) {
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Failed to load Space
configlet.");
            var $nRollback = 1;
           var $ztp_rollback = ztp_script:rollback_cfg( $nRollback, "Rollback
 configuration to beginning of phase 2" );
```

```
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Script rollbacks to
beginning of phase 2 and terminates.");
            var $die = ztp_script:terminate();
        }
       expr jcs:syslog( $SYSLOG, $APPNAME _ ": Phase 2 completed successfully"
 );
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": Now calling home to continue
 ZTP process" );
       expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Phase 2 completed successfully"
 );
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Now calling home to continue
 ZTP process." );
    }
    else { /*      Proceed with ZTD PULL mode work flow         */
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ZTD Pull model is in use."
);
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Going to proceed with Junos
 S/W upgrade" );
        /*-------------------------------------------------------------------*/
        /*      CHECK/UPGRADE VERSION OF THE JUNOS IMAGE                     */
        /*-------------------------------------------------------------------*/
        if ( not( ztp_script:ztp_grp_exists($ZTP_GROUP_NAME,
$ZTP_CODE_MACRO_NAME) ) ) {
       expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": No information for recommended
 Junos version found. Continue deployment with current version" );
        } else {
            var $code_upgrade = ztp_script:upgrade_code();
        }
        /*-------------------------------------------------------------------*/

        /*  DEACTIVATE PHASE-2 Configuration group                           */

        /*-------------------------------------------------------------------*/

        var $options_s2 := {
            <commit-options> {
              <log> "Disables configuration group for phase 2 on completion";

            }
        }
        var $change_s2 = {
            <configuration> {
                <apply-groups delete = "delete"> $ZTP_GROUP_STAGE_2;
            }
        }
        /* Loading configuration */
       var $results_step_3 := { call jcs:load-configuration( $action="merge",
 $commit-options=$options_s2, $configuration=$change_s2, $connection = $jnx
); }
        if ($results_step_3//xnm:warning) {
            for-each ($results_step_3//xnm:warning) {
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": apply-group commit
warning: " _ message );
            }
        }
        if ($results_step_3//xnm:error) {
            for-each ($results_step_3//xnm:error) {
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": apply-group commit
error: " _ message );
            }
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: Script failed."
```

```
);
            var $die = ztp_script:terminate();
        } else {
           expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ZTP PHASE 2 Configuration
 group was deactivated" );
        }
        /*-------------------------------------------------------------------*/
        /*      GET LOCATION OF THE BOX SPECIFIC CONFIGURATION FILE       */
        /*-------------------------------------------------------------------*/
        if ( not( ztp_script:ztp_grp_exists($ZTP_GROUP_CALLHOME,
$ZTP_CALLHOME_MACRO_NAME) ) ) {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": The information about site
specific configuration was not found in global template." );
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Stop script at this point"
);
        var $die = ztp_script:terminate();
        }
        var $get = <get-configuration> {
             <configuration> {
                <groups> {
                   <name> $ZTP_GROUP_CALLHOME;
                }
            }
        }
        var $got = jcs:execute( $jnx, $get );
        /* get full url of the configuration file */
        var $ztp_config_src =
$got/groups[name=$ZTP_GROUP_CALLHOME]/apply-macro[name=$ZTP_CALLHOME_MACRO_NAME]/data[name=$ZTP_CONFIG]/value;

        var $serial_no = ztp_script:get_serial_number();
       var $ztp_callhome_config = $ztp_config_src _ "/" _ $serial_no _ ".conf";

        /* Load configuration file and merge to existed configuration */

        if (not (ztp_script:load_config($ztp_callhome_config))) {
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: failed to load
box specific configuration. File was not found or bad configuration.");

   var $nRollback = 1;
   var $ztp_rollback = ztp_script:rollback_cfg( $nRollback, "Rollback
configuration to beginning of phase 2" );
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Script rollbacks to
beginning of phase 2 and terminates.");
            var $die = ztp_script:terminate();
        } else {
          expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Box specific configuration
 was loaded" );
        }
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": ZTD Phase 2 completed
successfully" );
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ZTD Phase 2 completed
successfully" );
    }
/*  var $delete_file = ztp_script:file-delete($filename); */
    var $die = ztp_script:terminate();
}
/* ------------------------------------------------------------------- */
/*   LOAD CONFIGURATION FROM FILE */
/* ------------------------------------------------------------------- */
function ztp_script:load_config($config_url, $action = "merge") {
    mvar $load_config = true();
```

```
    var $phrase = {
        if ($action == "merge") {
            expr "merged to existed configuration";
        } else {
            if ($action == "replace") {
                expr " and replace existed statements";
            } else {
                expr " be merged (default action) to existed configuration";
            }
        }
    }
    var $new_action = {
        if (($action == "merge") or ($action == "replace")) {
            expr $action;
        } else {
            expr "merge";
        }
    }
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Config file: " _ $config_url _
" will be loaded and " _ $phrase  );
    /* lock the config */
    var $lock = <lock-configuration>;
    var $did_lock = jcs:execute( $jnx, $lock );
    if ( $did_lock//self::xnm:error ) {
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": ERROR: unable to lock
configuration" );
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: unable to lock
configuration" );
        set $load_config = false();
    } else {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Configuration was locked" );

    }
    if ($load_config) {
        set $load_config = false();
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Processing config url " _
$config_url );
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": Processing config url " _
$config_url );
        /* load in new config */
        var $do_load = <load-configuration action="merge" url=$config_url
format="text">;
        var $did_load = jcs:execute( $jnx, $do_load );
        if( not( $did_load/load-success )) {
            expr jcs:syslog( $SYSLOG, $APPNAME _ ": ERROR: unable to load
config " _ $config_url );
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ERROR: unable to load
config " _ $config_url );
        } else {
           expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Configuration was loaded"
 );
            var $commit = <commit-configuration> {
                <full>;
                <synchronize>;
                <force-synchronize>;
                <log> "Initial config load";
            }
            var $did_commit = jcs:execute( $jnx, $commit );
            if ( $did_commit//self::xnm:error ) {
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Commit Failed" );
                expr jcs:syslog( $SYSLOG, $APPNAME _ ": Commit Failed" );
```

```
            } else {
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Configuration was
committed" );
                var $unlock = <unlock-configuration>;
                var $did_unlock = jcs:execute( $jnx, $unlock );
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": release config lock"
 );
                set $load_config = true();
                /*var $die = ztp_script:terminate();*/
            }
        }
    } /* end of foreach config_url */
    if (not($load_config)) {
        /* if we make it here, we failed to load the config */
        var $unlock = <unlock-configuration>;
        var $did_unlock = jcs:execute( $jnx, $unlock );
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": release config lock" );
    }
    result $load_config;
}
/* ------------------------------------------------------------------- */
/* Function ztp_grp_exists verifies if configuration group and        */
/* apply-macro (optional) name exist.                                 */
/* ------------------------------------------------------------------- */
function ztp_script:ztp_grp_exists($group_name, $macro_name = "N/A") {
    var $get_grp = <get-configuration> {
         <configuration> {
            <groups> {
                <name>;
            }
        }
    }
    /* geting variables from apply-macro */
    var $got_grp = jcs:execute( $jnx, $get_grp );
    mvar $grp_flag = "skip";
    for-each ($got_grp/groups/name) {
        set $grp_flag = {
            if  ((../name!=$group_name) and ($grp_flag!="exists")) {
                expr "skip";
            } else {
                expr "exists";
            } /* End if */
        }
    }
    /* Verification for the apply-macro */
    if ($grp_flag != "exists") {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Can't find a group " _
$group_name);
        result false();
    } else {
       expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Found a group " _ $group_name);

        if ( $macro_name!="N/A") {
            expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Looking for apply-macro
 " _ $macro_name _ " in group " _ $group_name );
            /* Look for apply-macro with specified name */
            var $get_mcr = <get-configuration> {
                <configuration> {
                    <groups> {
                        <name> $group_name;
                    }
```

```
                }
            }
            /* geting variables from apply-macro */
            var $got_mcr = jcs:execute( $jnx, $get_mcr );
            mvar $mcr_flag = "skip";
            for-each ($got_mcr/groups[name=$group_name]/apply-macro/name) {
                set $mcr_flag = {
                    if  ((../name!=$macro_name) and ($mcr_flag!="exists")) {
                        expr "skip";
                    } else {
                        expr "exists";
                    } /* End if */
                }
            }
            /* Verification for the group */
            if ($mcr_flag != "exists") {
              expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Can't find apply-macro
 " _ $macro_name );
                result false();
            } else {
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Found apply-macro "
 _ $macro_name );
                result true();
            }
        } else {
            result true();
        }
    }
}
/* ------------------------------------------------------------------- */
/* Function ztp_get_method returns ZTP METHOD: "pull" or "push"        */
/* ------------------------------------------------------------------- */
function ztp_script:ztp_get_method($group_name, $macro_name) {
    mvar $ztd_method = "pull";
    if ( not( ztp_script:ztp_grp_exists($group_name, $macro_name) ) ) {
        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": ZTD model not specified in
global template. Use pull method by default" );
    } else {
        var $get_grp = <get-configuration> {
             <configuration> {
                <groups> {
                   <name> $group_name;
                }
            }
        }
        /* getting variables from apply-macro */
        var $got_grp = jcs:execute( $jnx, $get_grp );
        for-each
($got_grp/groups[name=$group_name]/apply-macro[name=$macro_name]/data/name)
{
            if (../name == $ZTP_METHOD ) {
                set $ztd_method = ../value;
            }
        }
    }
    result $ztd_method;
}
/* ------------------------------------------------------------------- */
/* Function upgrade_code() checks the currently running Junos version */
/* and upgrades the code                                               */
/* ------------------------------------------------------------------- */
```

```
function ztp_script:upgrade_code() {
/* get the apply-macro */
var $get = <get-configuration> {
         <configuration> {
            <version>;
            <groups> {
               <name> $ZTP_GROUP_NAME;
               <apply-macro> {
                  <name> $ZTP_CODE_MACRO_NAME;
                }
            }
        }
    }
    var $got = jcs:execute( $jnx, $get );
    var $running_version = $got/version;
    var $production_version =
$got/groups[name=$ZTP_GROUP_NAME]/apply-macro[name=$ZTP_CODE_MACRO_NAME]/data[name=$ZTP_VERSION]/value;

    expr jcs:syslog( $SYSLOG, $APPNAME _ ": running_version = " _
$running_version);
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": running_version = " _
$running_version);
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": production_version = " _
$production_version);
    expr jcs:progress($JNPR_ZTP_SCRIPT _ ": production_version = " _
$production_version);
    if ($running_version == $production_version) {
       expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Code is up to date, no upgrade
 required");
        expr jcs:syslog( $SYSLOG, $APPNAME _ ": Code is up to date, no upgrade
 required");
        result false();
    } else {
        expr jcs:progress($JNPR_ZTP_SCRIPT _ ": upgrade required");
        var $image_file =
$got/groups[name=$ZTP_GROUP_NAME]/apply-macro[name=$ZTP_CODE_MACRO_NAME]/data[name=$ZTP_IMAGE]/value;

        expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Processing " _ $image_file
);
        /* Check s/w upgrade flag in basic template. If flag set to inactive
 skip upgrade procedure */
        mvar $ztd_inactive = "active";
        for-each
($got/groups[name=$ZTP_GROUP_NAME]/apply-macro[name=$ZTP_CODE_MACRO_NAME]/data/name)
 {
            if (../name == $ZTP_INACTIVE_FLAG ) {
                set $ztd_inactive = ../value;
            }
        }
        if ( $ztd_inactive == "inactive" ) {
            expr jcs:syslog( $SYSLOG, $APPNAME _ ": Skip software upgrade due
 to Upgrade flag in basic template set to inactive" );
            expr jcs:progress($JNPR_ZTP_SCRIPT _ ": Skip software upgrade due
 to Upgrade flag in basic template set to inactive" );
        } else {
            /* request system software add ... */
           expr jcs:syslog( $SYSLOG, $APPNAME _ ": installing image" );
           var $do_install := <request-package-add> {
                                <no-validate>;
                                <force>;
                                <reboot>;
```

```
                                <package-name> $image_file ;
            }
            var $install_results = jcs:execute( $jnx, $do_install );
            for-each( $install_results/../output ) {
                expr jcs:syslog( $SYSLOG, $APPNAME _ ": Install Error: ",
output );
                expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Install error: " _
output );
            }
            if ($install_results/../package-result == 0) {
               /* successfully installed package -- terminate script and wait
 for reboot */
                expr jcs:syslog( $SYSLOG, $APPNAME _ ": software installed"
);
                expr jcs:progress($JNPR_ZTP_SCRIPT _ ": software installed"
);
                var $die = ztp_script:terminate();
            }
        }
    } /* else */
} /* end function */
/* ------------------------------------------------------------------------ */
/* Function rolback_cfg($nRollback) rollback configuration to         */
/* nRollback version                                                        */
/* ------------------------------------------------------------------------ */
function ztp_script:rollback_cfg( $cfg_version=0, $rollback_log="Rollback
configuration" ) {
    var $rollback_options := {
        <commit-options> {
            <full>;
   <log> $rollback_log;
        }
    }
    var $rollback_configuration := { call jcs:load-configuration(
$commit-options = $rollback_options, $rollback = $cfg_version, $connection =
 $jnx ); }
}
/* ------------------------------------------------------------------------ */
/* Helper routines
/* ------------------------------------------------------------------------ */
function ztp_script:file-copy( $source, $filename, $destination ) {
    var $copy_get = <file-copy> {
        <source> $source _ "/" _ $filename;
        <destination> $destination _ "/" _ $filename;
        <staging-directory> $TMPDIR;
    }
    var $copy_got = jcs:execute( $jnx, $copy_get );
    if ( contains($copy_got,"failed")) {
        result false();
    } else {
        result true();
    }
}
function ztp_script:file-exists( $filename ) {
    var $ls_file = <file-list> { <path> $filename; }
    var $ls_got = jcs:execute( $jnx, $ls_file );
    var $retval = boolean( $ls_got//file-information );
    result $retval;
}
function ztp_script:file-delete( $filename ) {
    var $do_rm = <file-delete> { <path> $filename; }
```

```
    var $did_rm = jcs:execute( $jnx, $do_rm );
    /* @@@ trap error */
    result true();
}
function ztp_script:only_once() {
    if( ztp_script:file-exists( $ZTP_LOCKFILE )) {
        result true();
    } else {
        var $do_lock = <file-put> {
             <filename> $ZTP_LOCKFILE;
             <encoding> 'ascii';
             <file-contents> 'locked';
        }
    var $did_lock = jcs:execute( $jnx, $do_lock );
    result false();
    }
}
function ztp_script:terminate() {
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": SCRIPT-TERMINATE" );
    var $rm_lock = ztp_script:file-delete( $ZTP_LOCKFILE );
    terminate;
}
/* ------------------------------------------------------------------- */
/* GET SERIAL NUMBER */
/* ------------------------------------------------------------------- */
function ztp_script:get_serial_number() {
     expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": Getting box Serial Number" );
     expr jcs:syslog( $SYSLOG, $APPNAME _ ": Getting box Serial Number" );
     /* get our serial number */
    var $chassis_hardware := jcs:execute( $jnx, 'get-chassis-inventory' );
    var $serial_no = $chassis_hardware/chassis/serial-number;
    expr jcs:progress( $JNPR_ZTP_SCRIPT _ ": serial no = " _ $serial_no );
    expr jcs:syslog( $SYSLOG, $APPNAME _ ": serial no = " _ $serial_no );
    result $serial_no;
}
```

**Related Documentation**

- Customer Use Cases for ACX Series Routers on page 5
- ACX Autoinstallation Overview on page 8
- Deployment Methods for ACX Routers on page 12
- Example: Deploying ACX Routers Using the ZTD Pull Method on page 21
- Example: Deploying ACX Routers Using the ZTD Push Method on page 49