XC™ Series System Administration Guide (CLE 6.0.UP01)

Contents

About the XC Series System Administration Guide

This publication, released on June 20, 2016, includes administrative tasks for Cray XC series systems running SMW 8.0 UP01 and CLE 6.0 UP01.

Command Prompt Conventions

hostname in command prompts

Hostnames in command prompts indicate where the command must be run.

`hostname#`	Run the command on this hostname.
`smw#`	Run the command on the SMW.
`boot#`	Run the command on the boot node.
`sdb#`	Run the command on the SDB node.
`login#`	Run the command on any login node.
`smw1#` `smw2#`	For a system configured with the SMW failover feature there are two SMWs—one in an active role and the other in a passive role. The SMW that is active at the start of a procedure is *smw1*. The SMW that is passive is *smw2*.
`smwactive#` `smwpassive#`	In some scenarios, the active SMW is *smw1* at the start of a procedure—then the procedure requires a failover to the other SMW. In this case, the documentation will continue to refer to the formerly active SMW as *smw1*, even though *smw2* is now the active SMW. If further clarification is needed in a procedure, the active SMW will be called *smwactive* and the passive SMW will be called *smwpassive*.

account name in command prompts

The account that must run the command is also indicated in the prompt.

`smw#`
`boot#`
`sdb#`
`login#`
`hostname#`
	The `root` or super-user account always has the `#` character at the end of the prompt.

`crayadm@smw>` `crayadm@login>`	Any non-`root` account is indicated with *`account@hostname`*.

`user@hostname>`	A user account that is neither `root` nor `crayadm` is referred to as *user*.

command prompt inside chroot

If the `chroot` command is used, the prompt changes to indicate that it is inside a chroot'd environment on the hostname.

```
smw# chroot /path/to/chroot
chroot-smw#
```

directory path in command prompt

Sometimes the current path can be so long that including it in the prompt does not add clarity to the command example. Most of the time, the command can be executed from any directory. When it matters which directory the command is invoked within, the `cd` command is used to change into the directory, and the directory is referenced with a period (`.`) to indicate the current directory.

For example, here are actual prompts as they appear on the system:

```
smw:~ # cd /etc
smw:/etc# cd /var/tmp
smw:/var/tmp# ls ./file
smw:/var/tmp# su - crayadm
crayadm@smw:~> cd /usr/bin
crayadm@smw:/usr/bin> ./command
```

And here are the same prompts as they would appear in examples in this publication:

```
smw# cd /etc
smw# cd /var/tmp
smw# ls ./file
smw# su - crayadm
crayadm@smw> cd /usr/bin
crayadm@smw> ./command
```

Typographic Conventions

`Monospace`	Indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, key strokes (e.g., `Enter` and `Alt-Ctrl-F`), and other software constructs.
`Monospaced Bold`	Indicates commands that must be entered on a command line or in response to an interactive prompt.
`Oblique` or *`Italics`*	Indicates user-supplied values in commands or syntax definitions.
Proportional Bold	Indicates a graphical user interface window or element.
\ (backslash)	At the end of a command line, indicates the Linux® shell line continuation character (lines joined by a backslash are parsed as a single line). Do not type anything after the backslash or the continuation feature will not work correctly.

Scope and Audience

This publication covers a wide range of system management topics and is intended for experienced Cray system administrators.

Feedback

Visit the Cray Publications Portal at *http://pubs.cray.com* and make comments online using the **Contact Us** button in the upper-right corner or Email pubs@cray.com. Your comments are important to us and we will respond within 24 hours.

Trademarks

About the Cray Management System

With Cray Linux Environment (CLE) 6.0, Cray introduces a new management system built on these core principles:

- Separation of configuration data and software content
- Separation of the management infrastructure from the product content
- Modularity
- Prescriptive results
- Scalability

This Cray Management System (CMS) is intended to improve uptime through staging, reduce the risk associated with updates and changes, and enable users to extend functionality.

The CMS comprises these primary components:

IMPS Image Management and Provisioning System.

IMPS enables sites to manage software content in a prescriptive way. It leverages and extends industry-standard tools such as `zypper` and `rpm`. IMPS is used to create and distribute repository content (RPMs) and to create and update standard or custom images. Cray provides image recipes for different node types: service, login, compute, DAL, etc. The image recipes tie together the collections of software defined in the package collections and the repositories that contain the software. From them, IMPS builds a list of all the software and repositories referenced, and passes it to `zypper` or `yum`, which resolves the RPM dependencies and installs the software into the specified image root. See the IMPS man page for more information.

CMF Configuration Management Framework

The CMF is a combination of software and conventions that enable the modular management and application of configuration. Each application comes with the software needed to configure that application. All configuration information needed to operate the logical system is stored in a central repository called a config set. It is made available to every node in the system by means of the IMPS Distribution System (IDS), a read-only network share. Cray provides a configurator to enable sites to create, change, or add new configuration information. Configuration for all applications installed in an image is applied at boot time using cray-ansible, a wrapper that finds all Ansible plays installed on the system and executes them with Ansible.

NIMS Node Image Mapping Service

NIMS enables site administrators to assign any node or group of nodes any boot image. It also provides a mechanism for passing additional kernel parameters to the nodes on boot. See the NIMS man page for more information.

Ansible is installed into each image. During boot, each node runs all Ansible plays, pulling in the configuration information needed to self-configure ("pull" mode). Ansible is called twice during system boot—once from `initrd /init` before Linux has started up (*in_init*) and once after normal Linux startup with `systemd` (*multiuser*)—to cover both early and run level 3 use cases. Ansible can be run in “push” mode after boot to support reconfiguration.

Manage the System

Caution is encouraged when executing system management commands and procedures; hasty actions can result in down time and lost data.

IMPORTANT: Use persistent SCSI device names.

This does not apply to SMW disks: SCSI device names (`/dev/sd*`) are not guaranteed to be numbered the same from boot to boot. This inconsistency can cause serious system problems following a reboot. When installing CLE, the administrator **must** use persistent device names for file systems on the Cray system.

Cray recommends using the `/dev/disk/by-id/` persistent device names. Use `/dev/disk/by-id/` for the root file system in the `initramfs` image and in the `/etc/sysset.conf` installation configuration file as well as for other file systems, including Lustre (as specified in `/etc/fstab` and `/etc/sysset.conf`). For more information, see *XC™ Series Software Initial Installation and Configuration Guide*.

Alternatively, the administrator can define persistent names using a site-specific `udev` rule or `cray-scsidev-emulation`. However, only the `/dev/disk/by-id` method has been verified and tested.

CAUTION: The administrator must use `/dev/disk/by-id` when specifying the root file system. There is no support in the `initramfs` for `cray-scsidev-emulation` or custom `udev` rules.

Connect the SMW to the Console of a Service Node

The `xtcon` command is a console interface for service nodes. When it is executing, the `xtcon` command provides a two-way connection to the console of any running node.

With the release CLE 6.x, all service and compute nodes enable the `xtcon` console by default. If a node fails to boot, then the init boot sequence halts and drops into a console bash session waiting for the administrator to take action, such as debug the node. With release CLE 5.x, `xtcon` and the enablement of console on nodes is required via the kernel parameters.

See the `xtcon(8)` man page for additional information.

Configure Remote Access to SMW with VNC

Virtual network computing (VNC) software enables a user to view and interact with the SMW from another computer.

VNC is optional and enabling VNC is a site choice. With the DRAC on the SMW, many system administrators may prefer to use DRAC and not configure VNC.

To obtain a VNC client to connect to the server, download a VNC client from a reuptable website such as these:

- RealVNC™: *http://www.realvnc.com/*
- TightVNC™: *http://www.tightvnc.com/*

The VNC software requires a TCP/IP connection between the server and the viewer. Be aware that VNC is considered to be an insecure protocol, therefore Cray recommends that the VNC client only connect to the VNC server on the SMW via an SSH tunnel.

About the Integrated Dell Remote Access Controller (iDRAC)

The iDRAC is a systems management hardware and software solution that provides remote management capabilities, crashed system recovery, and power control functions for the System Management Workstation (SMW). The iDRAC alerts administrators to server issues, helps them perform remote server management, and reduces the need for physical access to the server. The iDRAC also facilitates inventory management and monitoring, deployment and troubleshooting. To help diagnose the probable cause of a system crash, the iDRAC can log event data and capture an image of the screen when it detects that the system has crashed.

For more information about the iDRAC, refer to online documentation at *http://www.dell.com*.

Change the Default iDRAC Password

About this task

This procedure describes how to log in to the iDRAC web interface and change a user password.

Procedure

1. Log in to the web interface as `root`.
2. Select **iDRAC settings** on the left navigation bar.
3. Select **Network/Security** on the main top navigation bar.
4. Select **Users** on the secondary top bar.
5. Select the user whose password is changing. For example, userid `2` and username `root`.
6. Select **Configure User**, then **Next**.
7. Enter the new password into the **New Password** and **Confirm New Password** fields.
8. Select **Apply** to complete the password change.

R815 SMW: Change the BIOS and iDRAC Settings

Prerequisites

This procedure assumes that the SMW is disconnected from the boot RAID and connected to a keyboard, monitor, and mouse.

About this task

This procedure changes the system setup for a Dell R815 SMW: the network connections, remote power control, and the remote console. Depending on the server model and version of BIOS configuration utility, there could be minor differences in the steps to configure the system. For more information, refer to the documentation for the

Dell server used at this site. Because Cray ships systems with most of the installation and configuration completed, some of the steps may have been done already.

For a Dell R630 SMW, see *R630 SMW: Change the BIOS and iDRAC Settings* on page 19.

Procedure

1. Remove SMW non-boot internal drives.

   Eject all the internal disk drives from the SMW except for the primary boot disk in slot 0 and the secondary boot disk in slot 1.

2. Power up the SMW. When the BIOS power-on self-test (POST) process begins, **quickly press the F2 key** after the following messages appear in the upper-right of the screen.

   ```
              F2 = System Setup
          F10 = System Services
       F11 = BIOS Boot Manager
                F12 = PXE Boot
   ```

   When the **F2** keypress is recognized, the **F2 = System Setup** line changes to **Entering System Setup**.

   After the POST process completes and all disk and network controllers have been initialized, the BIOS **System Setup** menu appears.

3. Change system time.

   The system time should be in UTC, not in the local timezone.

   a. Select **System Time** in the **System Setup** menu.

      The hours will be highlighted in blue.

   b. Set the correct time.

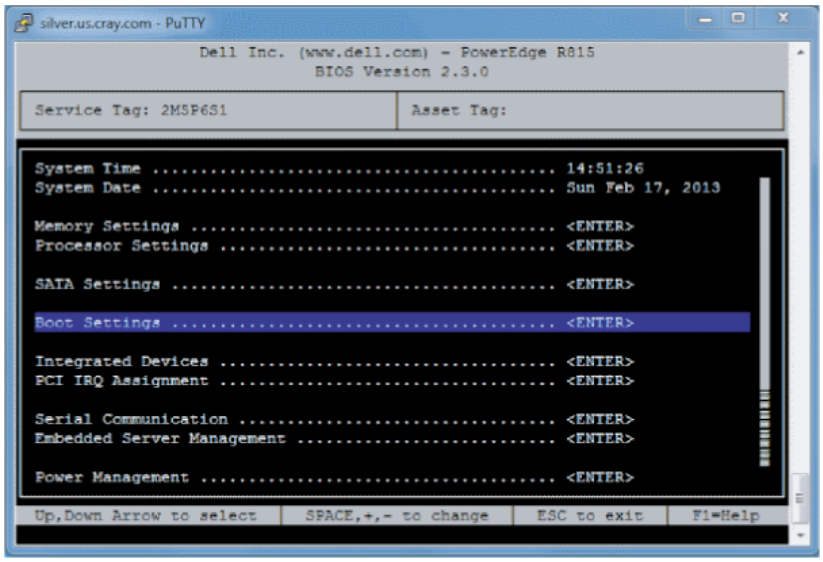
      1. Press the space key to change hours.
      2. Use the right-arrow key to select minutes, then change minutes with the space key.
      3. Use the right-arrow key to select seconds, then change seconds with the space key.

   c. Press **Esc** when the correct time is set.

4. Change boot settings.

   a. Select **Boot Settings** in the **System Setup** menu, then press **Enter**.

Figure 1. Dell R815 SMW Boot Settings Menu

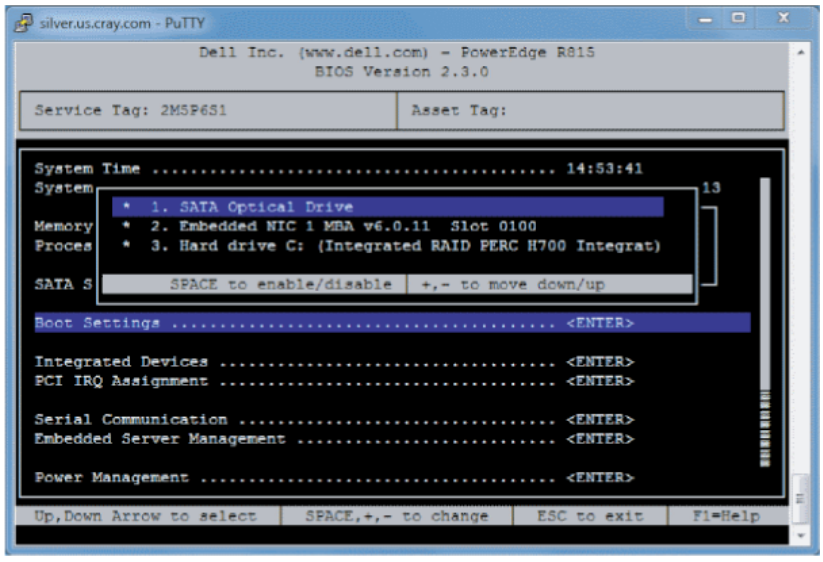


A pop-up menu with the following list appears:

```
Boot Mode ..................................... BIOS
Boot Sequence ................................. <ENTER>
USB Flash Drive Emulation Type................. <ENTER>
Boot Sequence Retry ........................... <Disabled>
```

b. Select **Boot Sequence**, then press **Enter**.

Figure 2. Dell R815 SMW Boot Sequence Settings



c. Change the order of items in the **Boot Sequence** list so that the optical (DVD) drive appears first, then the hard drive. If **Embedded NIC** appears in the list, it should end up below the optical drive and hard drive in the list.

d. Disable embedded NIC.

   Select `Embedded NIC`, then press **Enter** to disable it.

e. Press **Esc** to exit the **Boot Sequence** menu.

f. Press **Esc** again to exit the **Boot Settings** menu.

5. Change serial communication.

a. Select **Serial Communication** in the **System Setup** menu, then press **Enter**.

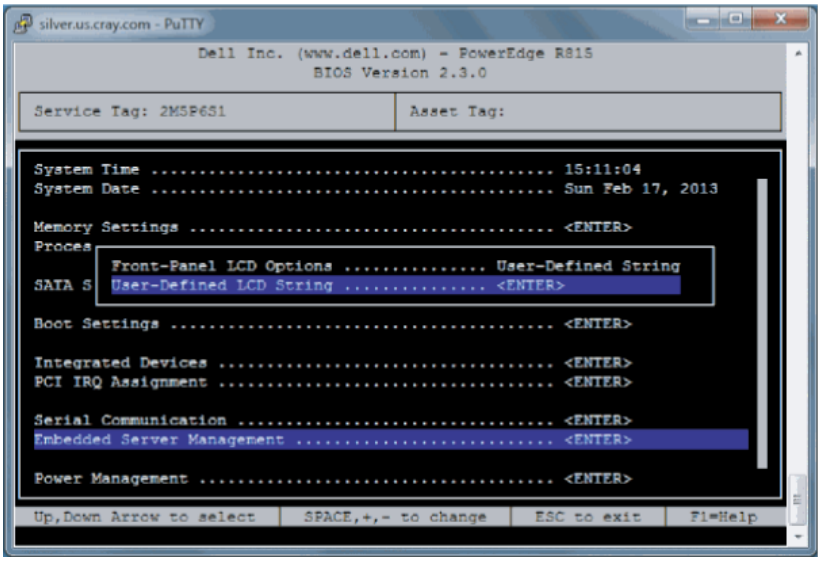
b. Confirm these settings in the **Serial Communication** menu.

- **Serial Communication** is set to **On with Console Redirection via COM2**
- **Serial Port Address** is set to **Serial Device1=COM2, Serial Device2=COM1**
- **External Serial Connector** is set to **Serial Device2**
- **Failsafe Baud Rate** is set to **115200**

c. Press **Esc** to exit the **Serial Communication** menu.

6. Select **Embedded Server Management** in the **System Setup** menu, then press **Enter**.

Figure 3. Dell R815 SMW Embedded Server Management Settings



a. Set **Front-Panel LCD Options** to **User-Defined LCD String** in the **Embedded Server Management** menu. Use the space key to cycle through the choices, then use the down-arrow key.

b. Set **User-Defined LCD String** to the login hostname (e.g., cray-drac), then press **Enter**.

c. Press **Esc** to exit the **Embedded Server Management** menu.

7. Insert base operating system DVD into SMW.

Insert the base operating system DVD labeled Cray-SMWbase12- into the DVD drive. (The DVD drive on the front of the SMW may be hidden by a removable decorative bezel.)

8. Save BIOS changes and exit.

a. Press **Esc** to exit the BIOS **System Setup** menu.

A menu with a list of exit options appears.

```
Save changes and exit
Discard changes and exit
Return to Setup
```

b. Ensure that **Save changes and exit** is selected, then press **Enter**.

The SMW resets automatically.

9. Enter BIOS boot manager.

a. When the BIOS POST process begins again, **quickly press the F11 key** within 5 seconds of when the following messages appear in the upper-right of the screen.

```
              F2 = System Setup
          F10 = System Services
        F11 = BIOS Boot Manager
                 F12 = PXE Boot
```

When the **F11** keypress is recognized, the **F11 = BIOS Boot Manager** line changes to **Entering BIOS Boot Manager**.

10. Change the integrated Dell Remote Access Controller (iDRAC) settings.

Watch the screen carefully as text scrolls until the **iDRAC6 Configuration Utility 1.57** line is visible. When the line **Press <Ctrl-E> for Remote Access Setup within 5 sec...** displays, press **Ctrl-E** within 5 seconds.

```
0 5 0 ATA WDC WD5000BPVT-0 1A01 465 GB
LSI Corporation MPT2 boot ROM successfully installed!
iDRAC6 Configuration Utility 1.57
Copyright 2010 Dell Inc. All Rights Reserved
iDRAC6 Firmware Revision version: 1.54.15
Primary Backplane Firmware Revision 1.07
-------------------------------
IPv6 Settings
-------------------------------
IPv6 Stack : Disabled
Address 1 : ::
Default Gateway : ::
-------------------------------
IPv4 Settings
-------------------------------
IPv4 Stack : Enabled
IP Address : 172. 31. 73.142
Subnet mask : 255.255.255. 0
Default Gateway : 172. 31. 73. 1
Press <Ctrl-E> for Remote Access Setup within 5 sec...
```

The **iDRAC6 Configuration Utility** menu appears.

11. Set **iDRAC LAN** to **ON**.

12. Configure the iDRAC LAN.

Select **LAN Parameters**, then press **Enter**.

a. Configure iDRAC6 name.

Use the arrow key to scroll down and select **iDRAC6 Name**, then press **Enter**. Enter a value for **Current DNS iDRAC6 Name** (e.g., smw-drac), then press **Enter**.

b. Configure domain name.

Use the arrow key to scroll down and select **Domain Name**, then press **Enter**. Enter a value for **Current Domain Name** (e.g., us.cray.com), then press **Enter**.

c. Configure hostname string.

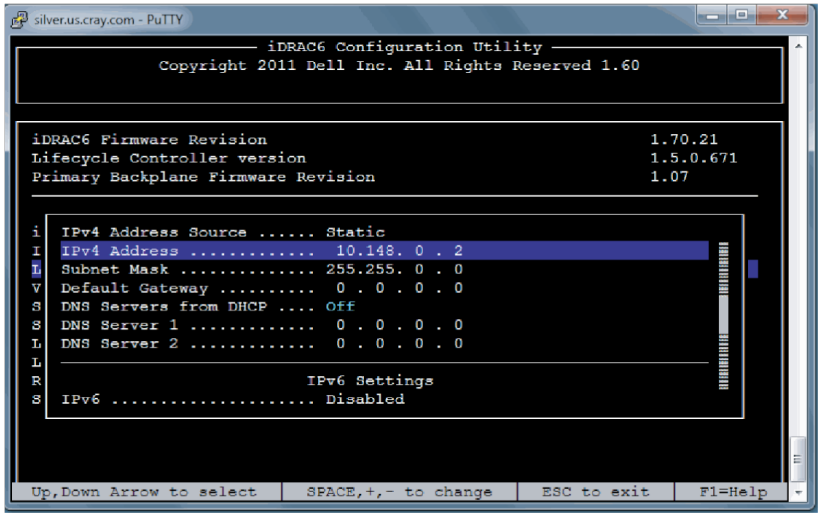
Use the arrow key to scroll down and select **Host Name String**, then press **Enter**. Enter a value for **Current Host Name String** (e.g., smw-drac), then press **Enter**.

d. Configure IPv4 settings.

Use the arrow key to scroll down into the **IPv4 Settings** group and confirm that the **IPv4 Address Source** is set to **static**. Then enter values for the following:

IPv4 Address (the SMW DRAC IP address)
Subnet Mask (the SMW iDRAC subnet mask)
Default Gateway (the SMW iDRAC default gateway)
DNS Server 1 (the first site DNS server)
DNS Server 2 (the second site DNS server)

Figure 4. Dell R815 SMW DRAC IPv4 Parameter Settings



e. Configure IPv6 settings.

Use the arrow key to scroll down into the **IPv6 Settings** group and ensure that **IPv6** is disabled.

f. Press **Esc** to exit **LAN Parameters** and return to the **iDRAC6 Configuration Utility** menu.

13. Configure iDRAC virtual media.

a. Select **Domain Name**, then press **Enter**.

b. Select **Virtual Media Configuration**, then press **Enter**.

c. Select the **Virtual Media** line and press the space key until it indicates **Detached**.

d. Press **Esc** to exit the **Virtual Media Configuration** menu.

14. Set the password for the iDRAC LAN root account.

Using the arrow keys, select **LAN User Configuration**, then press **Enter**. The following configuration is for both SSH and web browser access to the iDRAC.

a. Select **Account User Name** and enter the account name "root."

b. Select **Enter Password** and enter the intended password.

c. Select **Confirm Password** and enter the intended password again.

d. Press **Esc** to return to the **iDRAC6 Configuration Utility** menu.

15. Exit the iDRAC configuration utility.

a. Press **Esc** to exit the **iDRAC6 Configuration Utility** menu.

b. Select **Save Changes and Exit**.

The **BIOS Boot Manager** menu appears.

16. Choose to boot from SATA Optical Drive.

Using the arrow keys, select the **SATA Optical Drive** entry, then press **Enter**.

R630 SMW: Change the BIOS and iDRAC Settings

Prerequisites

This procedure assumes that the internal disk drives of the SMW have just been configured as RAID virtual disks and the system is rebooting. If the system is not rebooting, press **Ctrl-Alt-Delete** to reboot.

About this task

This procedure describes how to change the system setup for the SMW: the network connections, remote power control, and the remote console. This procedure includes detailed steps for the Dell R630 server. Depending on the server model and version of BIOS configuration utility, there could be minor differences in the steps to configure the system. For more information, refer to the documentation for the Dell server used at this site. Because Cray ships systems with most of the installation and configuration completed, some of the steps may have been done already.

For a Dell R815 server, see *R815 SMW: Change the BIOS and iDRAC Settings* on page 13.

Procedure

Watch as the system reboots and the BIOS power-on self-test (POST) process begins. Be prepared to press **F2**, when prompted, to change the system setup.

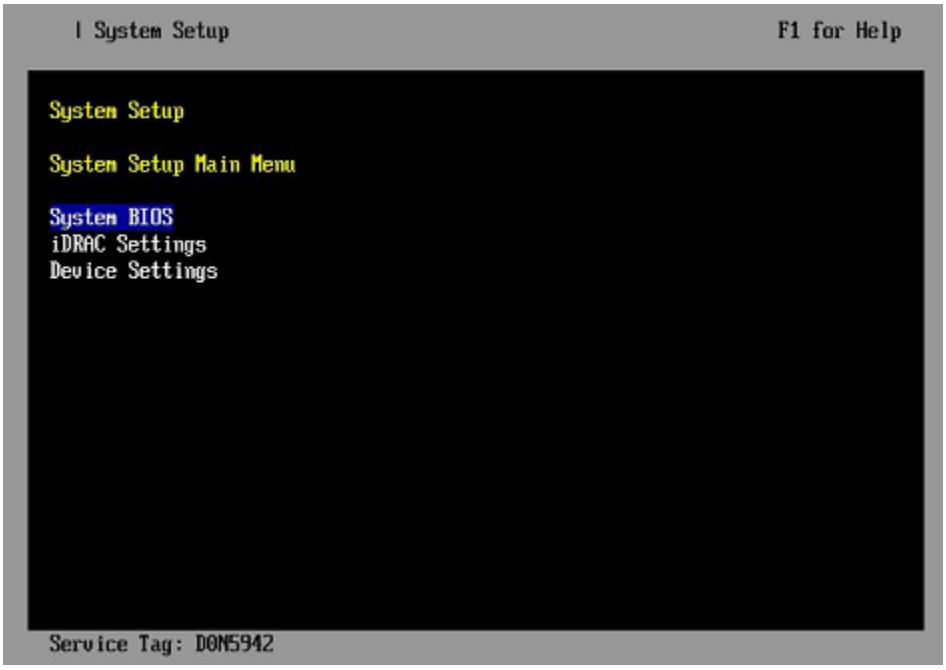
1. Press the **F2** key immediately after the following messages appear in the upper-rightupper-left of the screen:

```
F2 = System Setup
F10 = System Services
F11 = BIOS Boot Manager
F12 = PXE Boot
```

When the **F2** keypress is recognized, the **F2 = System Setup** line changes color from white-on-black to white-on-blue.

After the POST process completes and all disk and network controllers have been initialized, the Dell **System Setup** screen appears. The following submenus are available on the **System Setup Main Menu** and will be used in subsequent steps: **System BIOS**, **iDRAC Settings**, and **Device Settings**.

Figure 5. Dell R630 System Setup Main Menu



TIP: In system setup screens,

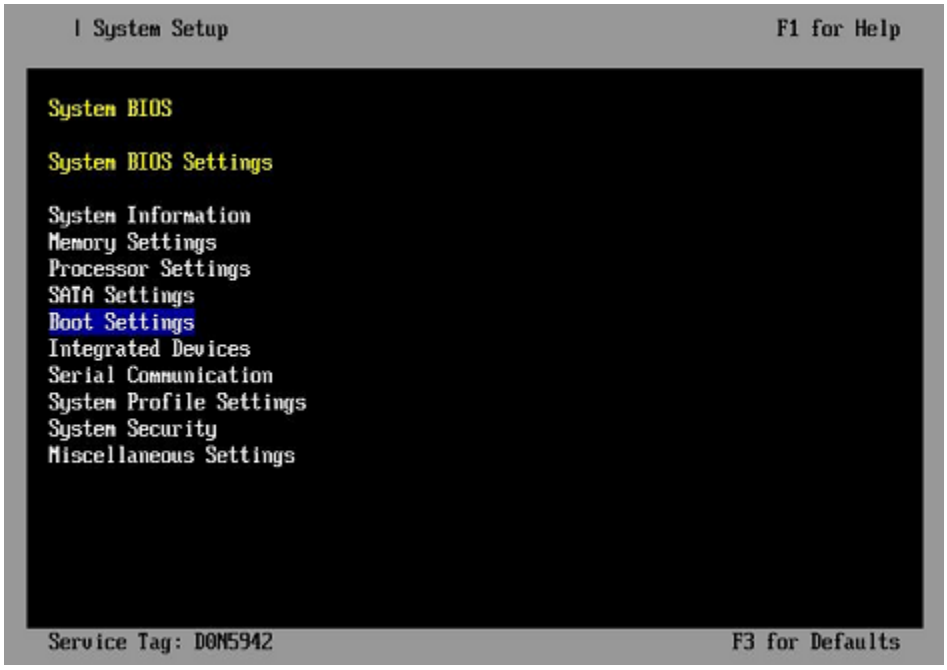
- Use the **Tab** key to move to different areas on the screen.
- Use the up-arrow and down-arrow keys to highlight or select an item in a list, then press the **Enter** key to enter or apply the item.
- Press the **Esc** key to exit a submenu and return to the previous screen.

2. Change the BIOS settings.

a. Select **System BIOS** on the **System Setup Main Menu**, then press **Enter**.

The **System BIOS Settings** screen appears.

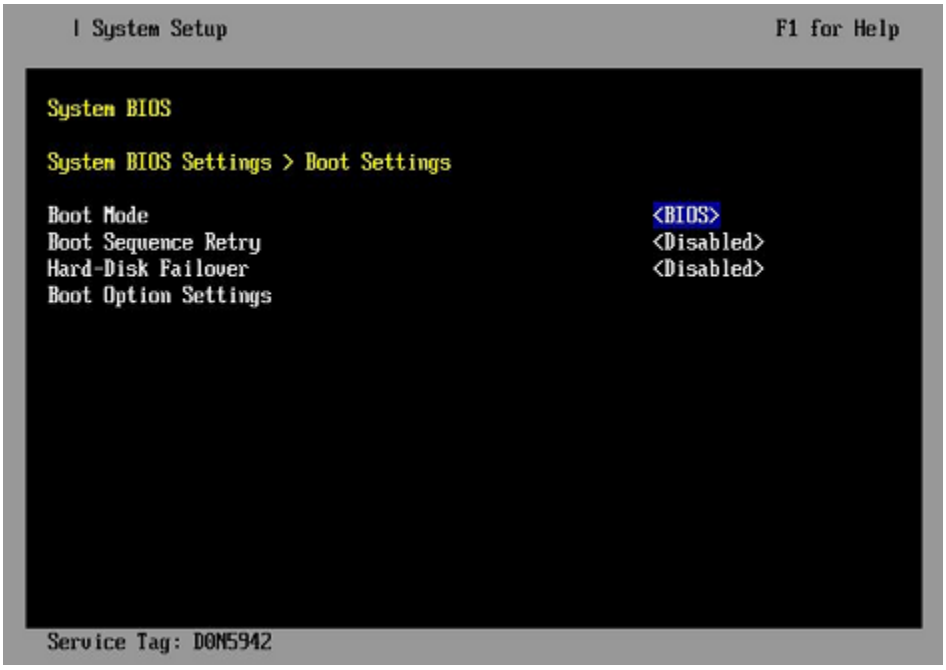
Figure 6. Dell R630 System BIOS Settings Screen



b. Change Boot Settings.

1. Select **Boot Settings** on the **System BIOS Settings** screen, then press **Enter**. The **Boot Settings** screen appears.

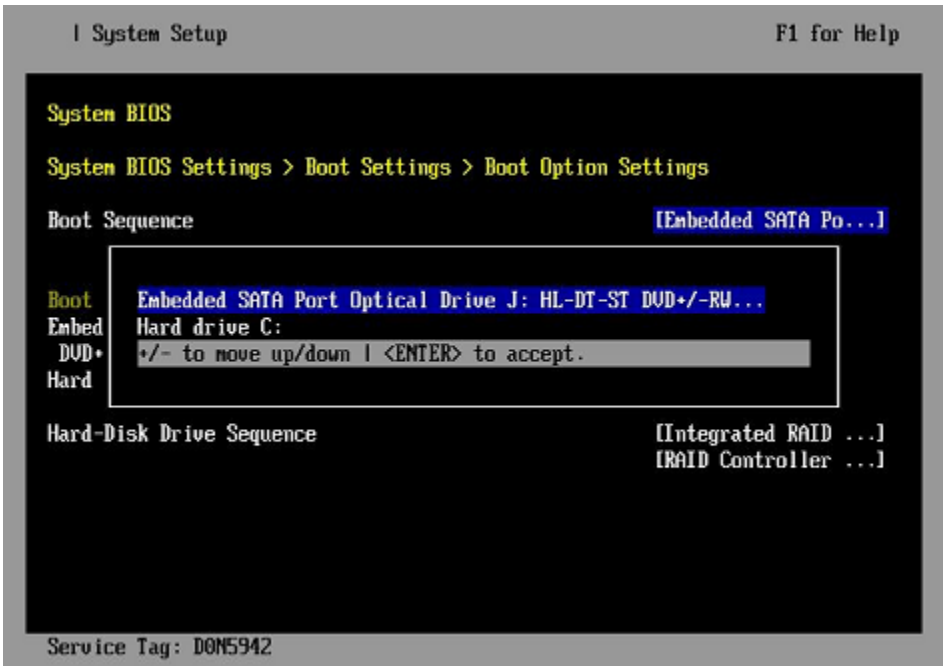
   Figure 7. Dell R630 Boot Settings Screen



2. Ensure that **Boot Mode** is **BIOS** and not **UEFI**.
3. Select **Boot Option Settings**, then press **Enter**.

4. Select **Boot Sequence** on the **Boot Option Settings** screen, then press **Enter** to view a pop-up window with the boot sequence.

   Figure 8. Dell R630 BIOS Boot Sequence



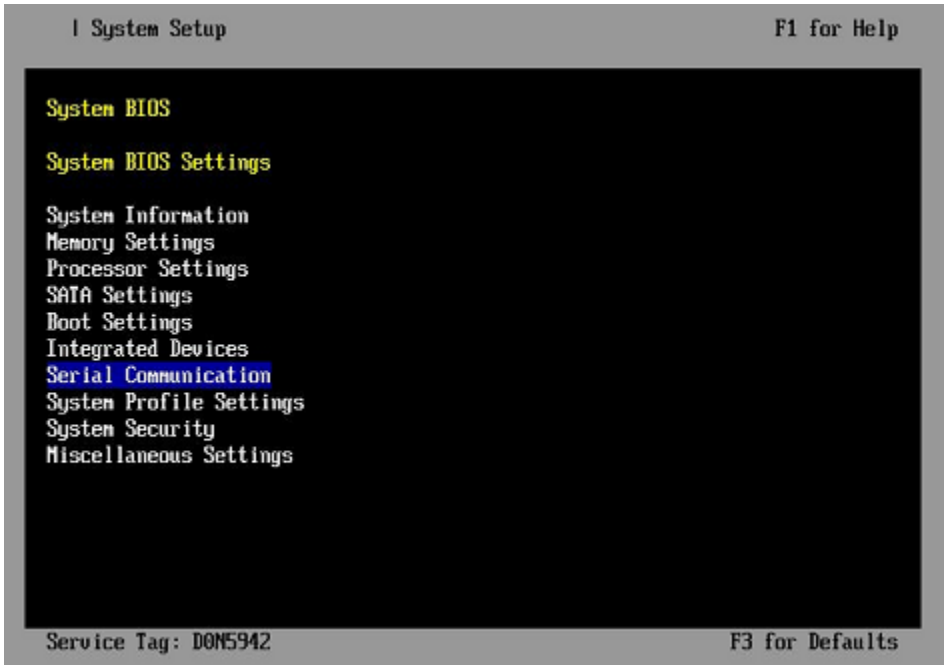
5. Change the boot order in the pop-up window so that the optical drive appears first, then the hard drive. If **Integrated NIC** appears in the list, it should end up below the optical drive and hard drive in the list.

   TIP: Use the up-arrow or down-arrow key to highlight or select an item, then use the **+** and **-** keys to move the item up or down.

6. Select **OK**, then press **Enter** to accept the change.
7. Click the box next to **Hard drive C:** under the **Boot Option/Enable/Disable** section to enable it. Do the same for the optical drive, if necessary.
8. Select **integrated NIC**, then press **Enter** to disable it.
9. Press **Esc** to exit **Boot Option Settings**.
10. Press **Esc** to exit **Boot Settings** and return to the **System BIOS Settings** screen.

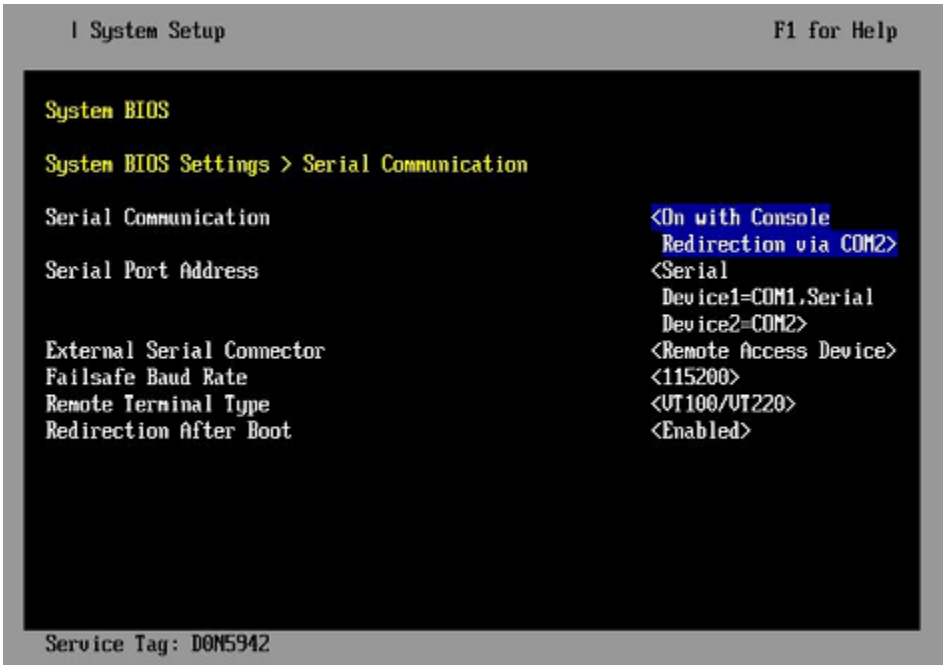
c. Change Serial Communication Settings.

Figure 9. Dell R630 System BIOS Settings: Serial Communication



1. Select **Serial Communication** on the **System BIOS Settings** screen. The **Serial Communication** screen appears.

   Figure 10. Dell R630 Serial Communication Screen



2. Select **Serial Communication** on the **Serial Communication** screen, then press **Enter**. A pop-up window displays the available options.
3. Select **On with Console Redirection via COM2** in the pop-up window, then press **Enter** to accept the change.

4. Select **Serial Port Address**, then select **Serial Device1=COM1, Serial Device2=COM2**, then press **Enter**.
5. Select **External Serial Connector**, then press **Enter**. A pop-up window displays the available options.
6. Select **Remote Access Device** in the pop-up window, then press **Enter** to return to the previous screen.
7. Select **Failsafe Baud Rate**, then press **Enter**. A pop-up window displays the available options.
8. Select **115200** in the pop-up window, then press **Enter** to return to the previous screen.
9. Press the **Esc** key to exit the **Serial Communication** screen.
10. Press **Esc** to exit the **System BIOS Settings** screen. A "Settings have changed" message appears.
11. Select **Yes** to save changes. A "Settings saved successfully" message appears.
12. Select **Ok**.

3. Change the iDRAC (Integrated Dell Remote Access Controller) settings.

   Select **iDRAC Settings** on the **System Setup Main Menu**, then press **Enter**.

   The **iDRAC Settings** screen appears.

   Figure 11. Dell R630 iDRAC Settings Screen

4. Change the iDRAC network.

   a. Select **Network** to display a long list of network settings.

   b. Change the DNS DRAC name.

      Use the arrow key to scroll down to **DNS DRAC Name**, then enter an iDRAC hostname that is similar to the SMW node hostname (e.g., cray-drac).

   c. Change the static DNS domain name.

Use the arrow key to scroll down to **Static DNS Domain Name**, then enter the DNS domain name and press **Enter**.

d. Change the IPv4 settings.

Use the arrow key to scroll down to the **IPV4 SETTINGS** list.

1. Ensure that IPv4 is enabled.
   a. If necessary, select **Enable IPV4**, then press **Enter**.
   b. Select **<Enabled>** in the pop-up window, then press **Enter** to return to the previous screen.
2. Ensure that DHCP is disabled.
   a. If necessary, select **Enable DHCP**, then press **Enter**.
   b. Select **<Disabled>** in the pop-up window, then press **Enter** to return to the previous screen.
3. Change the IP address.
   a. Select **Static IP Address**.
   b. Enter the IP address of the iDRAC interface (`ipmi0`) for the SMW, then press **Enter**.
4. Change the gateway.
   a. Select **Static Gateway**.
   b. Enter the appropriate value for the gateway of the network to which the iDRAC is connected, then press **Enter**.
5. Change the subnet mask.
   a. Select **Subnet Mask**.
   b. Enter the subnet mask for the network to which the iDRAC is connected (such as `255.255.255.0`), then press **Enter**.
6. Change the DNS server settings.
   a. Select **Static Preferred DNS Server**, enter the IP address of the primary DNS server, then press **Enter**.
   b. Select **Alternate DNS Server**, enter the IP address of the alternate DNS server, then press **Enter**.

e. Change the IPMI settings.

Change the IPMI settings to enable the Serial Over LAN (SOL) console.

1. Use the arrow key to scroll down to the **IPMI SETTINGS** list.
2. Ensure that **Enable IPMI over LAN** is selected.

   TIP: Use the left-arrow or right-arrow to switch between two settings.
3. Ensure that **Channel Privilege Level Limit** is set to **Administrator**.

f. Exit Network screen.

Press the **Esc** key to exit the **Network** screen and return to the **iDRAC Settings** screen.

5. Change hostname in iDRAC LCD display.

Change front panel security to show the hostname in LCD display.

a. Use the arrow key to scroll down and highlight **Front Panel Security** on the **iDRAC Settings** screen, then press **Enter**.

b. Select **Set LCD message**, then press **Enter**.

c. Select **User-Defined String**, then press **Enter**.

d. Select **User-Defined String**, then enter the SMW hostname and press **Enter**.

e. Press the **Esc** key to exit the **Front Panel Security** screen.

6. (Optional) Change the iDRAC **System Location** fields.

Change the **System Location** configuration on the **iDRAC Settings** screen to set any of these fields: **Data Center Name**, **Aisle Name**, **Rack Name**, and **Rack Slot**.

7. Set the password for the iDRAC root account.

a. Use the arrow key to highlight **User Configuration** on the **iDRAC Settings** screen, then press **Enter**.

b. Confirm that User Name is root. Select **User Name**, then enter the "root" user name.

c. Select **Change Password**, then enter a new password.

d. Reenter the new password in the next pop-up window to confirm it (the default password is "calvin").

e. Press the **Esc** key to exit the **User Configuration** screen.

8. Exit iDRAC settings.

a. Press the **Esc** key to exit the **iDRAC Settings** screen.

A "Settings have changed" message appears.

b. Select **Yes**, then press **Enter** to save the changes.

A "Success" message appears.

c. Select **Ok**, then press **Enter**.

The main screen (**System Setup Main Menu**) appears.

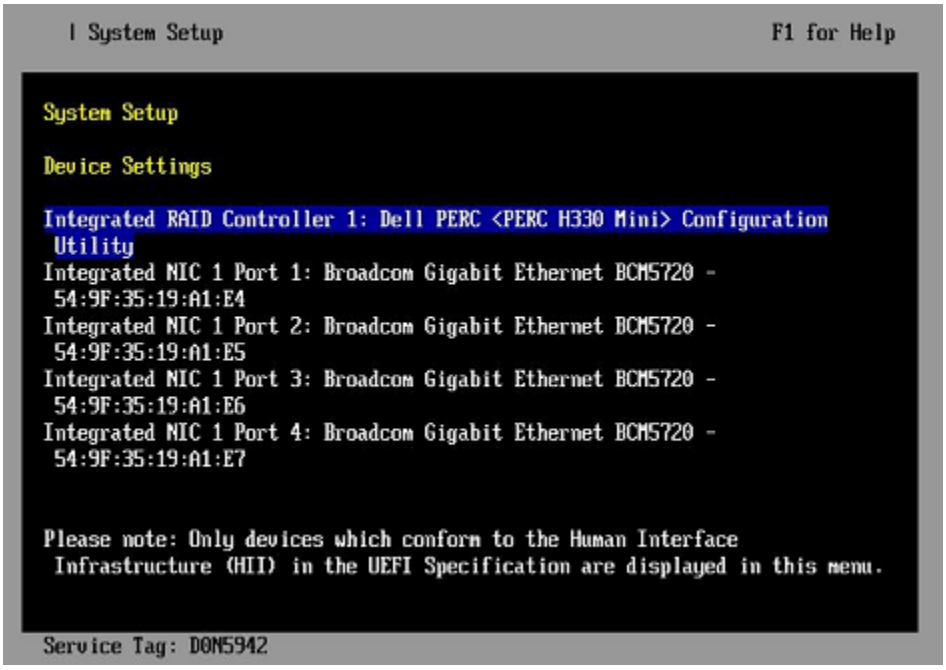
9. Change device settings.

These steps disable an integrated NIC device by changing the setting for the integrated NIC on a port from **PXE** to **None**.

a. Change Integrated NIC 1 Port 1

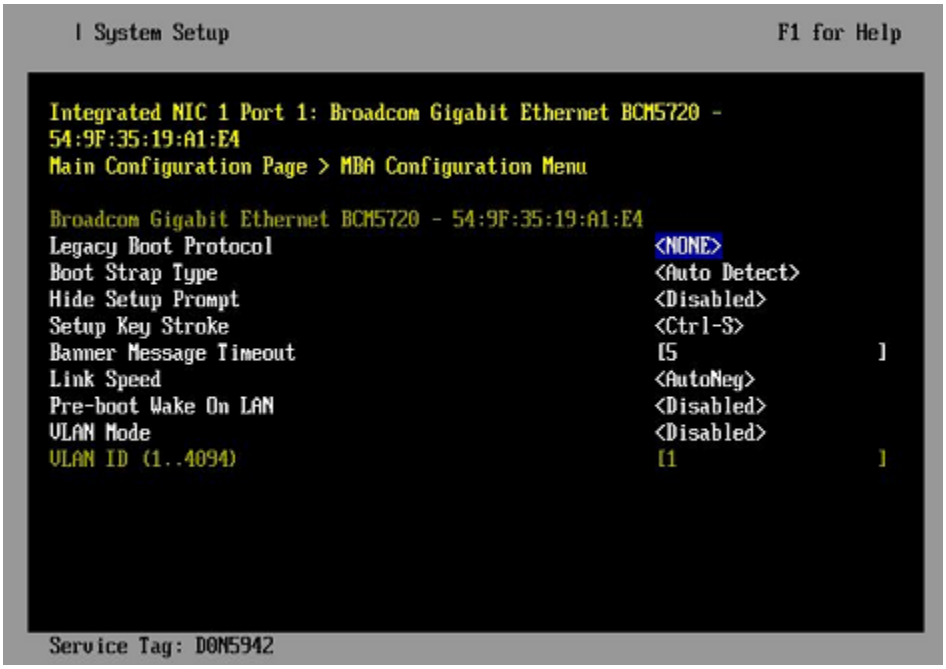
1. Select **Device Settings** on the **System Setup Main Menu**, then press **Enter**. The **Device Settings** screen appears.

Figure 12. Dell R630 Device Settings Screen



2. Select **Integrated NIC 1 Port 1: ...** on the **Device Settings** screen, then press **Enter**.
3. Select **MBA Configuration Menu** on the **Main Configuration Page** screen, then press **Enter**.

Figure 13. Dell R630 BIOS MBA Configuration Settings



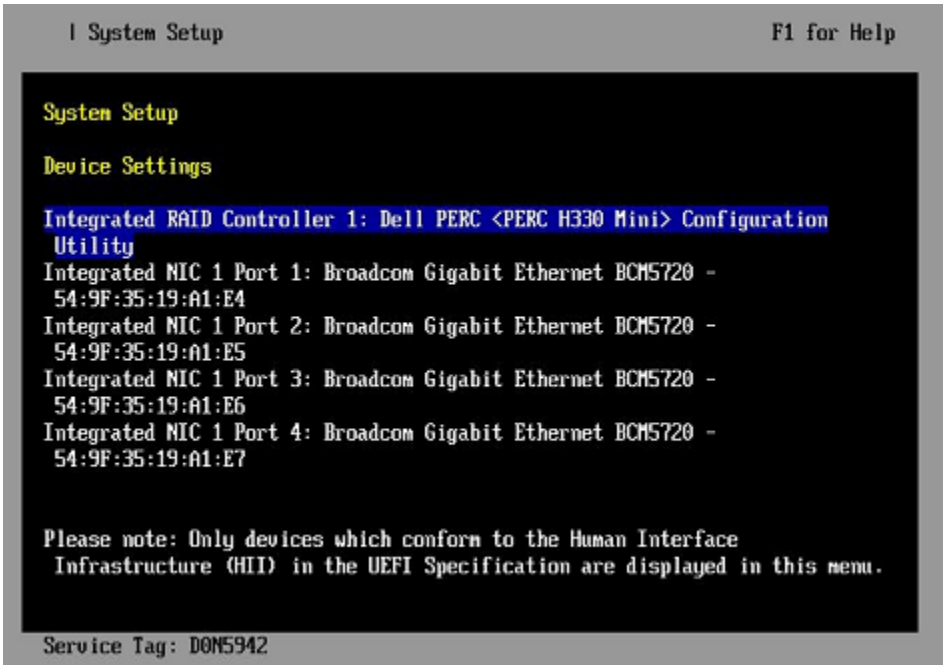
4. Select **Legacy Boot Protocol** on the **MBA Configuration Menu** screen, use the right-arrow or left-arrow key to highlight **None**, then press **Enter**.
5. Press the **Esc** key to exit the **MBA Configuration Menu** screen.

6. Press **Esc** to exit the **Main Configuration Page** screen. A "Warning Saving Changes" message appears.
7. Select **Yes**, then press **Enter** to save the changes. A "Success" message appears.
8. Select **OK**, then press **Enter**. The **Device Settings** screen appears.
9. Press **Esc** to exit the **Device Settings** screen. A "Settings have changed" message appears.
10. Select **Yes**, then press **Enter** to save the changes. A "Settings saved successfully" message appears.
11. Select **OK**, then press **Enter**. The main screen (**System Setup Main Menu**) appears.

b. Change Integrated NIC 1 Port 2

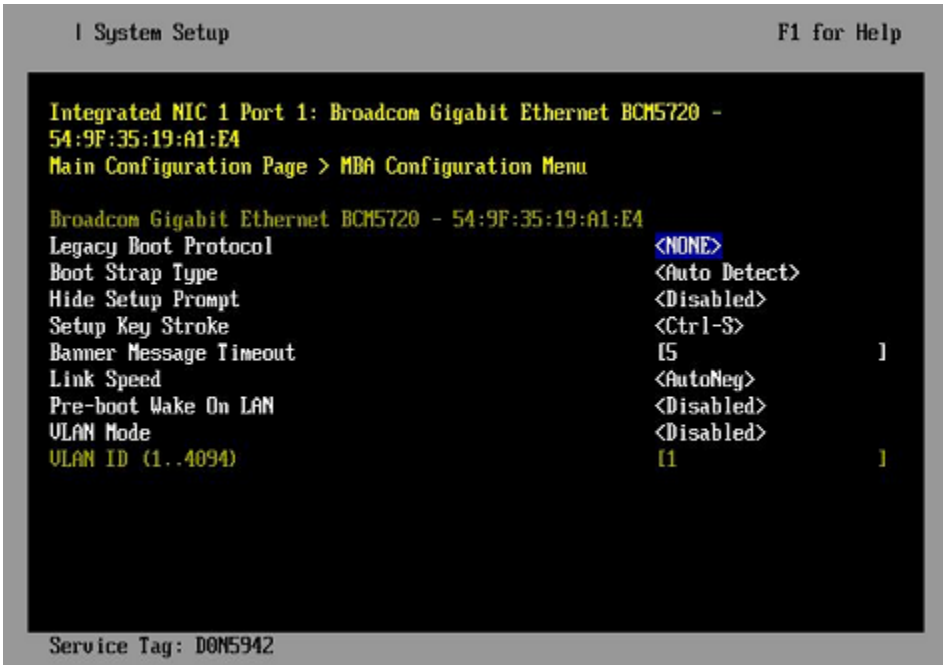
1. Select **Device Settings** on the **System Setup Main Menu**, then press **Enter**. The **Device Settings** screen appears.

Figure 14. Dell R630 Device Settings Screen



2. Select **Integrated NIC 1 Port 2: ...** on the **Device Settings** screen, then press **Enter**.
3. Select **MBA Configuration Menu** on the **Main Configuration Page** screen, then press **Enter**.

Figure 15. Dell R630 BIOS MBA Configuration Settings



4. Select **Legacy Boot Protocol** on the **MBA Configuration Menu** screen, use the right-arrow or left-arrow key to highlight **None**, then press **Enter**.
5. Press the **Esc** key to exit the **MBA Configuration Menu** screen.
6. Press **Esc** to exit the **Main Configuration Page** screen. A "Warning Saving Changes" message appears.
7. Select **Yes**, then press **Enter** to save the changes. A "Success" message appears.
8. Select **OK**, then press **Enter**. The **Device Settings** screen appears.
9. Press **Esc** to exit the **Device Settings** screen. A "Settings have changed" message appears.
10. Select **Yes**, then press **Enter** to save the changes. A "Settings saved successfully" message appears.
11. Select **OK**, then press **Enter**. The main screen (**System Setup Main Menu**) appears.

Use the iDRAC

Prerequisites

This procedure assumes an integrated Dell Remote Access Controller (iDRAC) has been set up for use with the SMW.

About this task

An iDRAC enables remote management of a Cray System Management Workstation (SMW). This procedure describes how to access the SMW console through the iDRAC.

Procedure

1. Bring up a web browser.

2. Go to: `https://cray-drac`, where *`cray-drac`* is the name assigned to the iDRAC during setup. The iDRAC login screen appears.

3. Enter the account user name and password set up in *Change the Default iDRAC Password* on page 13 or an iDRAC setup procedure.

   The **System Summary** window appears.

4. Select **Submit**.

5. To access the SMW console, select the **Console Media** tab.

   The **Virtual Console and Virtual Media** window appears.

6. Select **Launch Virtual Console**.

   TIP: By default, the console window has two cursors: one for the console and one for the administrator's window environment. To switch to single-cursor mode, select **Tools**, then **Single Cursor**. This single cursor will not move outside the console window. To exit single-cursor mode, press the **F9** key.

   TIP: To log out of the virtual console, kill the window or select **File**, then **Exit**. The web browser is still logged into the iDRAC.

   For detailed information, see the iDRAC documentation at: *http://www.dell.com/support*.

Boot the System

The `xtbootsys` command is used to manually boot the boot node, service nodes, and CNL compute nodes. An administrator can also boot the system using both user-defined and built-in procedures in automation files (e.g., `/opt/cray/hss/default/etc/auto.generic`).

```
crayadm@smw> xtbootsys -a auto.myautobootfile
```

Before modifying the `auto.generic` file, Cray recommends making a copy because it will be replaced by an SMW software upgrade. Avoid strict boot ordering of service nodes in an automated boot file. For related procedures, see .

The `xtbootsys` command prevents unintentional booting of currently booted partitions. If a boot automation file is being used, `xtbootsys` checks that file to determine if the string `shutdown` exists within any actions defined in the file. If it does, `xtbootsys` assumes that a shutdown is being done, and no further verification of operating on a booted partition occurs. If the partition is not being shut down and the boot node is in the `ready` state, `xtbootsys` announces this fact and queries for confirmation to proceed. By default, confirmation is enabled. To disable or enable confirmation when booting booted partitions, use the `xtbootsys config,confirm_booting_booted` and the `config,confirm_booting_booted_last_session` global TCL variables, the `--config` *`name=value`* on the `xtbootsys` command line, or the `XTBOOTSYS_CONFIRM_BOOTING_BOOTED` and `XTBOOTSYS_CONFIRM_BOOTING_BOOTED_LAST_SESSION` environment variables.

Run Tests After Boot is Complete

Prerequisites

This procedure assumes that the system has completed booting.

About this task

Log in to the login node as crayadm. This can be done from the SMW to the boot node to the login node or directly from another computer to the login node without passing through the SMW and boot node. Then perform these rudimentary functionality checks.

Procedure

1. Run `apstat` to get the number of nodes to use for the following commands.

```
crayadm@login> NUMNODES=$(($(apstat -v | grep XT | awk "{print \$3}")))
crayadm@login> echo NUMNODES is $NUMNODES
```

2. Verify that all nodes run (from `/tmp`).

```
crayadm@login> cd /tmp; aprun -b -n $NUMNODES -N 1 /bin/cat /proc/sys/kernel/
hostname
```

3. Verify that the home directory is working by running a job.

```
crayadm@login> cd ~; aprun -b -n $NUMNODES -N 1 /bin/cat /proc/sys/kernel/
hostname
```

4. Verify that the Lustre directory is working by running a job.

```
crayadm@login> cd /lustre_file_system
crayadm@login> aprun -b -n $NUMNODES -N 1 /bin/cat /proc/sys/kernel/hostname
```

5. Run `xtcheckssd` to ensure that SMW databases have the current state of compute node SSDs.

```
root@login# pcmd -r -n ALL_COMPUTE "/opt/cray/ssd/bin/xtcheckssd"
```

   This needs to be done after an initial installation, SSD hardware change, system update, and periodically (daily/weekly).

Manually Boot the Boot Node and Service Nodes

Prerequisites

The Lustre file system should start up before the compute nodes, and compute node Lustre clients should be unmounted before shutting down the Lustre file system.

About this task

If more than one boot image is set up to run, the administrator can check which image is set up to boot with the `xtcli boot_cfg show` or `xtcli part_cfg show p`*N* commands. To change which image is booting, see *Update the Boot Configuration*.

Procedure

1. Log on to the SMW as `crayadm`.

2. Invoke the `xtbootsys` command to boot the boot node. If the system is partitioned, invoke `xtbootsys` with the `--partition p`*N* option.

```
crayadm@smw:~> xtbootsys
```

The `xtbootsys` command prompts with a series of questions. Cray recommends answering **`yes`** by typing **`Y`** to each question.

```
Enter your boot choice:
       0) boot bootnode ...
       1) boot sdb ...
       2) boot compute ...
       3) boot service ...
       4) boot all (not supported) ...
       5) boot all_comp ...
       10) boot bootnode and wait ...
       11) boot sdb and wait ...
       12) boot compute and wait ...
       13) boot service and wait ...
       14) boot all and wait (not supported) ...
       15) boot all_comp and wait ...
       17) boot using a loadfile ...
       18) turn console flood control off ...
       19) turn console flood control on ...
       20) spawn off the network link recovery daemon (xtnlrd)...
       q) quit.
```

3. Select option 10 (boot bootnode and wait).

   A prompt to confirm the selection is displayed. Press the **`Enter`** key or type **`Y`** to each question to confirm.

   ```
   Do you want to boot the boot node ? [Yn] Y
   Do you want to send the ec_boot event ('no' means to only load memory) ? [Yn] Y
   ```

   After the boot node is booted, the process returns to the boot choice menu.

4. Select option 11 (boot sdb and wait).

   A prompt to confirm the selection is displayed. Press the **`Enter`** key or type **`Y`** to each question to confirm.

   ```
   Do you want to boot the sdb node ? [Yn] Y
   Do you want to send the ec_boot event ('no' means to only load memory) ? [Yn] Y
   ```

5. Select option 13 (boot service and wait).

   A prompt to enter a list of service nodes to be booted is displayed.

6. Type **`p0`** to boot the remaining service nodes in the entire system or **`pN`** (where *N* is the partition number) to boot a partition.

   ```
   Do you want to boot service p0 ? [Yn] Y
   Do you want to send the ec_boot event ('no' means to only load memory) ? [Yn] Y
   ```

   To confirm the selection, press the **`Enter`** key or type **`Y`** to each question.

7. Log on to any service nodes for which there are local configuration or startup scripts (such as starting Lustre) and run the scripts.

Manually Boot the Compute Nodes

Prerequisites

All service and login nodes are booted and Lustre, if configured at this time, has started.

Procedure

1. Invoke the `xtbootsys` command if it is not running.

```
crayadm@smw:~> xtbootsys
```

```
Enter your boot choice:
       0) boot bootnode ...
       1) boot sdb ...
       2) boot compute ...
       3) boot service ...
       4) boot all (not supported) ...
       5) boot all_comp ...
       10) boot bootnode and wait ...
       11) boot sdb and wait ...
       12) boot compute and wait ...
       13) boot service and wait ...
       14) boot all and wait (not supported) ...
       15) boot all_comp and wait ...
       17) boot using a loadfile ...
       18) turn console flood control off ...
       19) turn console flood control on ...
       20) spawn off the network link recovery daemon (xtnlrd)...
       q) quit.
```

2. Select option **17** (boot using a loadfile). A series of prompts are displayed. Type the responses indicated in the following example. For the `component list` prompt, type **p0** to boot the entire system, or **p***N* (where *N* is the partition number) to boot a partition. At the final three prompts, press the **Enter** key.

```
Enter your boot choice: 17
Enter a boot type string (or nothing to do nothing): CNL0
Enter a boot type option (or nothing to do nothing): compute
Enter a component list (or nothing to do nothing): p0
Enter 'any' to wait for any console output,
   or 'linux' to wait for a linux style boot,
   or anything else (or nothing) to not wait at all: Enter
Enter an alternative CPIO archive name (or nothing): Enter
Do you want to send the ec_boot event ('no' means to only load memory) ? [Yn]
Enter
```

3. Return to the `xtbootsys` menu after all compute nodes are booted. Type **q** to exit the `xtbootsys` program.

4. Remove the `/etc/nologin` file from all service nodes to permit a non-root account to log on.

```
smw:~# ssh root@boot
boot:~# xtunspec -r /rr/current -d /etc/nologin
```

Reboot a Single Compute Node

A system administrator can initiate a warm boot with the `xtbootsys` command's `--reboot` option. This operation performs minimal initialization followed by a boot of only the selected compute nodes. Unlike the

sequence that is used by the `xtbounce` command, there is no power cycling of the Cray ASICs or of the node itself; therefore, the high-speed network (HSN) routing information is preserved. Do not specify a session identifier (`-s` or `--session` option) because `--reboot` continues the last session and adds the selected components to it.

Reboot a single comput node

For this example, reboot node `c1-0c2s1n2`:

```
crayadm@smw:~> xtbootsys --reboot c1-0c2s1n2
```

Reboot Login or Network Nodes

Login or network nodes cannot be rebooted through a `shutdown` or reboot command issued on the node; they must be restarted through the HSS system using the `xtbootsys --reboot` *`idlist`* SMW command. The HSS must be used so that the proper kernel is pushed to the node.

IMPORTANT: Do not attempt to warm boot nodes running other services in this manner.

For additional information, see the `xtbootsys(8)` man page.

Reboot login or network nodes

```
crayadm@smw:~> xtbootsys --reboot idlist
```

Debug Ansible Failures During System Boot

Ansible runs in `init` and Ansible runs a second time after `systemd` completes the boot process. Ansible failures in `init` cause the affected node to drop into a debug shell for node access via `xtcon` for troubleshooting. When the debug shell is exited, Ansible is re-executed in `init`. A node's boot does not proceed until the first run of `cray-ansible` in `init` is successful.

The Ansible callback plugin captures any file changes made by Ansible file modules and stores a record of these changes in log files located at `/var/opt/cray/log/ansible/changelog`. The plugin provides detailed failure information, including the path to the task file being executed and any config set variable references in the task file.

Ansible logs under `/var/opt/cray/log/ansible` are collected via `cdump` and `xtdumpsys`. In addition, `xtdumpsys` collects the files from running nodes, changed by Ansible according to the changelog callback plugin. When possible, Ansible Cray-provided plays create a backup of files modify by a play to let the administrator to perform a diff of these files to see the changes made by Ansible. Administrators can use the `ansible_cfg_search` command to examine an image and a config set. This command outputs a list of variables and the Ansible files that accessed each variable.

Examine System Logs

Various logs receive entries during the boot process that can indicate boot problems.

`systemd` Journal

The `systemd` init system takes over the boot process after `initrd`. Use the `journalctl -a` to display all kernel messages and other information in the `systemd` journal. Using `journalctl -f` displays the most recent

journal entries and continuously prints new entries. `systemd` stores messages in a custom database, the `systemd` journal. The information available in the journal includes:

- `syslogd` messages
- Kernel log messages
- `initrd` messages
- Messages written to stdout/stderr for all services

HSS Daemon Logs

The HSS daemons and the `rsyslogd` daemon running on the SMW logs to files in the `/var/opt/cray/log` directory. These daemons include `nimsd`, `xtpmd`, `xtremoted`, `xtpowerd`, `xtsnmpd`, `xtdiagd`, `erfsd`, `state_manager`, `bootmanager`, `sedc_manager`, `nid_mgr`, `erdh`, and `erd`.

SMW Command Log

The `/var/opt/cray/log/commands` log lists the commands issued from the SMW console.

CLE Boot Logs

The output from booting CLE is in the `/var/opt/cray/log/p0-current` log. For more detailed information, go to the `p0-current` directory and examine these log files:

- `bootinfo.`*`timestamp`*

  Contains output from the `xtbootsys` command. Timing information for how long sections of the boot process take is listed at the bottom of this file.

- `console-`*`YYYYMMDD`*

  Contains the combined console output from every node. To find Ansible failures for a node during `init`,

Look Up Configuration Details

The `ansible_cfg_search` command line tool lets an administrator on the SMW specify a config set, an IMPS image root, and optionally, an Ansible play to query for config set lookups and template locations. The intent is to provide a general understanding of which configuration files are used at specific points in the boot process. The command uses the playbook structure to inspect the plays, roles, templates, and task files for patterns that appear to be config set variable lookups. For each lookup found in the Ansible content, the command lists a path to the configuration template that holds the variable.

Before using `ansible_cfg_search`, load the `system-config` module.

```
ansible_cfg_search [-h] [-p PLAYBOOK] [-s CONFIG_SETTING] [-e LOOKUP_EXPRESSION]
            [-q] config_set image
```

Required arguments:

config_set	The config set to search for config variables.
image	The IMPS image root containing ansible content to search. If necessary, use the `image list` command to find the IMPS image root.

Optional arguments:

-h, --help	Display help information.
-p *PLAYBOOK*, --playbook *PLAYBOOK*	The Ansible playbook file contained in the IMPS image to search for configuration lookups.
-s *CONFIG_SETTING*, --config-setting *CONFIG_SETTING*	List the configuration templates and Ansible files that contain the specified setting.

Example

Examine a config set to determine the settings that the baseopts.yaml play is looking up:

```
smw: # module load system-config
smw: # ansible_cfg_search p0 \
service-master_cle_6.1.DV00-build6.1.75_sles_12-created20160429 \
--play baseopts.yaml
```

Output:

```
/var/opt/cray/imps/image_roots/service-master_cle_6.1.DV00-build6.1.118DV_sles_12-
created20160510/etc/ansible/baseopts.yaml:

  - /var/opt/cray/imps/image_roots/service-master_cle_6.1.DV00-build6.1.118DV_sles_12-
created20160510/etc/ansible/roles/baseopts/tasks/smw.yaml:
    - /var/opt/cray/imps/config/sets/p0/config/cray_user_settings_config.yaml:
      - cray_user_settings.settings.default_modules.data.smw

  - /var/opt/cray/imps/image_roots/service-master_cle_6.1.DV00-build6.1.118DV_sles_12-
created20160510/etc/ansible/roles/baseopts/tasks/main.yaml:
    - /var/opt/cray/imps/config/sets/p0/config/cray_login_config.yaml:
      - cray_login.settings.login_nodes.data.members
    - /var/opt/cray/imps/config/sets/p0/config/cray_user_settings_config.yaml:
      - cray_user_settings.settings.default_modules.data.login
      - cray_user_settings.settings.default_modules.data.service
      - cray_user_settings.settings.default_modules.data.smw

  - /var/opt/cray/imps/image_roots/service-master_cle_6.1.DV00-build6.1.118DV_sles_12-
created20160510/etc/ansible/roles/baseopts/tasks/login.yaml:
    - /var/opt/cray/imps/config/sets/p0/config/cray_user_settings_config.yaml:
      - cray_user_settings.settings.default_modules.data.login

  - /var/opt/cray/imps/image_roots/service-master_cle_6.1.DV00-build6.1.118DV_sles_12-
created20160510/etc/ansible/roles/baseopts/tasks/service.yaml:
    - /var/opt/cray/imps/config/sets/p0/config/cray_user_settings_config.yaml:
      - cray_user_settings.default_modules
      - cray_user_settings.default_modules.login
      - cray_user_settings.default_modules.service
      - cray_user_settings.default_modules.smw
      - cray_user_settings.settings.default_modules.data.service
```

Examine Ansible Changelogs

The Ansible changelog provides information about files created, modified, and deleted by Ansible. Changelogs are created for `cray-ansible` when it first runs during the `init` phase and again when `cray-ansible` runs for the second time, during the booted phase. These logs are on the SMW in `/var/opt/cray/log/ansible`.

Logs created in the first phase (init):

sitelog-init	Contains Ansible play output from each task in executed plays.
file-changelog-init	Human-readable listing of each file changed by an Ansible play.

file-changelog-init.yaml	Machine-readable listing of each file changed by an Ansible play.

Logs created in the second phase (booted):

sitelog-booted	Contains Ansible play output from each task in executed plays.
file-changelog-booted	Human-readable listing of each file changed by an Ansible play.
file-changelog-booted.yaml	Machine-readable listing of each file changed by an Ansible play.

This `sitelog` entry shows that a task updated the message of the day (`motd`) file.

```
2016-01-17 12:15:27,671 TASK: [cle_motd | task motd, release]
*************************************
2016-01-17 12:15:27,671 changed: [localhost] => {"changed": true,
"cmd": "grep RELEASE /etc/opt/cray/release/cle-release | awk -F\\='{print $2}'",
"delta": "0:00:00.002536", "end": "2016-01-17 12:15:27.471384", "rc": 0,
"start": "2016-01-17 12:15:27.468848", "stderr": "", "stdout": "6.0.UP01",
"warnings": []}
```

The location of failing task can be found in plays:

```
boot# grep -Rn "task motd, release" /etc/ansible \
/etc/opt/cray/config/current/ansible
/etc/ansible/roles/cle_motd/tasks/motd.yaml:15:- name: task motd, release
```

The `file-changelog` files show the Ansible phase, each changed file, and the play that changed the file. This an entry from a `file-changelog-init` changelog:

```
Apr 05 2016 21:07:47 (init) template: file '/etc/nologin' changed by Ansible
task file '/etc/ansible/roles/early/tasks/nologin.yaml' with owner=root,
group=root, mode=0775
```

This an entry from a `file-changelog-booted` changelog:

```
May 16 2016 22:26:39 (booted) lineinfile: file '/etc/hosts' changed by Ansible
task file '/etc/ansible/roles/hosts/tasks/main.yaml' with owner=None,
group=None, mode=None
```

The same entry for the `/etc/hosts` edit in the `file-changelog-booted.yaml` changelog:

```
- backup_file_path: ''
  file_path: /etc/hosts
  group: null
  mode: null
  module: lineinfile
  owner: null
  phase: booted
  play: populate local hostfile
  state: null
  task_file: /etc/ansible/roles/hosts/tasks/main.yaml
  task_name: Add additional hosts to master file
  time: May 16 2016 22:26:39
```

The changelog entry fields are:

Field Name	Description
backup_file_path	Location of backup copy of file modified or deleted, if available.
file_path	Full path to the file which was modified.
group	Group given to the file if created or modified, or null if not specified.
mode	Permissions changed on the file if created or modified, or null if permissions were not changed.
module	Ansible module executed.
owner	Owner given to the file if created or modified, or null if not specified.
phase	Values are "booted" or "init".
play	Name of play making change.
state	Whether a line should be "present" or "absent".
task_file	Name of the task file which made the change.
task_name	Name of the task which made this change.
time	Format is "Month Day Year HH:mm:ss".

Debug Ansible Failures in `init`

About this task

Check the console log on the SMW to find out which nodes failed. Ansible failures in `init` drop a node into debug shell. The boot process is not allowed to continue until `cray-ansible` during `init` is successful on a node.

Procedure

1. Look for `cray-ansible` failures in the SMW console log.

```
crayadm@smw~> /var/opt/cray/log/p0-current> cat console-20160523 | grep
'completed in init - FAILED'
```

```
<158>1 2016-05-23T12:01:22.576591-05:00 c0-0c0s0n1 xtconsole 31798
p0-20160523t115109 [console@34] cray-ansible: /etc/ansible/site.yaml completed
in init - FAILED.
<158>1 2016-05-23T12:01:22.576634-05:00 c0-0c0s0n1 xtconsole 31798
p0-20160523t115109 [console@34] cray-ansible: /etc/ansible/site.yaml completed
in init - FAILED.
<158>1 2016-05-23T12:01:34.411653-05:00 c0-0c0s1n2 xtconsole 31798
p0-20160523t115109 [console@34] cray-ansible: /etc/ansible/site.yaml completed
in init - FAILED.
<158>1 2016-05-23T12:01:34.411699-05:00 c1-0c2s1n2 xtconsole 31798
p0-20160523t115109 [console@34] cray-ansible: /etc/ansible/site.yaml completed
in init - FAILED.
```

2. Access the debug shell with `xtcon` from the SMW.

```
smw# xtcon c1-0c2s1n2
```

```
nid00035#
```

3. Inspect Ansible logs on the node in `/var/opt/cray/log/ansible`, make a configuration change in the config set, or do some other corrective action. Exiting from the debug shell causes `cray-ansible` to run again in `init`.

Examine System Dumps

The `xtdumpsys` command collects and analyzes information from a Cray XC system that is failing or has failed, has crashed, or is hung. The dump file includes:

- Event log data, active heartbeat probing, voltages, temperatures, health faults, in-memory console buffers, and high-speed interconnection network errors.
- Config sets from the SMW.
- Ansible logs from nodes.
- Ansible changed files log from nodes can be collected.
- NIMS logs from SMW can be collected.

Include the files that Ansible changed by using the `ansible_changed_files` `xtdumpsys` plugin.

```
xtdumpsys --plugins-include=ansible_changed_files --reason="add changed files" -
add c0-0c0s3n2
```

Include the NIMS logs from the SMW by using the `nims_logs` `xtdumpsys` plugin. The NIMS logs are written to the `nims` directory in the dump.

```
xtdumpsys --plugins-include=nims_logs --reason="include NIMS logs"
```

Log on to the Boot Node

About this task

The standard Cray configuration has a gigabit Ethernet connection between the SMW and boot node. All other nodes on the Cray system are accessible from the boot node.

Procedure

1. Log on to the SMW as `crayadm`.
2. There are two methods to log on to the boot node: `ssh` to the boot node.
   - Use `ssh`:

     ```
     crayadm@smw:~>  ssh boot
     crayadm@boot:~>
     ```

   - Open an administrator window on the SMW:

     ```
     crayadm@smw:~> xterm -ls -vb -sb -sl 2049 6&
     ```

     After the window opens, use it to `ssh` to the boot node.

Display Boot Configuration Information

Use the `xtcli` command to display the configuration information for the primary and backup boot nodes, the primary and backup SDB nodes, and the `cpio` path.

Display boot configuration information for the entire system

```
crayadm@smw:~> xtcli boot_cfg show
Network topology: class 2
=== xtcli_boot_cfg ===
[boot]: c0-0c0s0n1:ready,c0-0c0s0n1:ready
[sdb]: c1-0c0s1n1:ready
[cpio_path]: /tmp/boot/kernel.cpio_5.2.14-wGPFS
```

Display boot configuration information for one partition in a system

```
crayadm@smw:~> xtcli part_cfg show pN
```

Where `pN` is the partition number. `p0` is always the whole system.

Update the Boot Configuration

The HSS `xtcli boot_cfg` command allows the administrator to specify the primary and backup boot nodes and the primary and backup SDB nodes for `s0` or `p0` (the entire system).

For a partitioned system, use `xtcli part_cfg` to manage boot configurations for partitions.

For more information, see the `xtcli_boot(8)` and `xtcli_part(8)` man pages.

For this example, update the boot configuration using the boot image `/bootimagedir/bootimage`, primary boot node (for example, `c0-0c0s0n1`), backup boot node, primary SDB node, and the backup SDB node:

```
crayadm@smw:~> xtcli boot_cfg update -b primaryboot_id,backupboot_id \
-d primarySDB_id,backupSDB_id -i /bootimagedir/bootimage
```

Display the Format of the SDB attributes Table

When the SDB boots, it reads the `/etc/opt/cray/sdb/attributes` file and loads it into the SDB `attributes` table.

To display the format of the `attributes` SDB table, use the `mysql` command:

```
crayadm@login:~> mysql -e "desc attributes;" -h sdb XTAdmin
+----------+------------------+------+-----+---------+-------+
| Field    | Type             | Null | Key | Default | Extra |
+----------+------------------+------+-----+---------+-------+
| nodeid   | int(32) unsigned | NO   | PRI | 0       |       |
| archtype | int(4) unsigned  | NO   |     | 2       |       |
```

```
| osclass   | int(4) unsigned   | NO   |     | 2        |        |
| coremask  | int(4) unsigned   | NO   |     | 1        |        |
| availmem  | int(32) unsigned  | NO   |     | 0        |        |
| pageszl2  | int(32) unsigned  | NO   |     | 12       |        |
| clockmhz  | int(32) unsigned  | YES  |     | NULL     |        |
| label0    | varchar(32)       | YES  |     | NULL     |        |
| label1    | varchar(32)       | YES  |     | NULL     |        |
| label2    | varchar(32)       | YES  |     | NULL     |        |
| label3    | varchar(32)       | YES  |     | NULL     |        |
| numcores  | int(4) unsigned   | NO   |     | 1        |        |
| sockets   | int(4) unsigned   | NO   |     | 1        |        |
| dies      | int(4) unsigned   | NO   |     | 1        |        |
+-----------+-------------------+------+-----+----------+--------+
```

The service database command pair `xtdb2attr` and `xtattr2db` enables the system administrator to update the `attributes` table in the SDB. For additional information about updating SDB tables using command pairs, see *Update SDB Tables* on page 41.

Update SDB Tables

The CLE command pairs shown enable the system administrator to update tables in the SDB. One command converts the data into an ASCII text file to edit; the other writes the data back into the database file.

Table 1. Service Database Update Commands

Get Command	Put Command	Table Accessed	Reason to Use	Default File
`xtdb2proc`	`xtproc2db`	`processor`	Updates the database when a node is taken out of service	`./processor`
`xtdb2attr`	`xtattr2db`	`attributes`	Updates the database when node attributes change	`./attribute`
`xtdb2nodeclasses`	`xtnodeclasses2db`	`service_processor`	Updates the database when a node's class changes	`./node_classes`
`xtdb2segment`	`xtsegment2db`	`segment`	For nodes with multiple NUMA nodes, updates the database when attribute information about node changes	`./segment`
`xtdb2servcmd`	`xtservcmd2db`	`service_cmd`	Updates the database when characteristics of a service change	`./serv_cmd`
`xtdb2servconfig`	`xtservconfig2db`	`service_config`	Updates the database when services change	`./serv_config`

Get Command	Put Command	Table Accessed	Reason to Use	Default File
xtdb2etchosts	none	processor	Manages IP mapping for service nodes	none
xtdb2lustrefailover	xtlustrefailover2db	lustre_failover	Updates the database when a node's Lustre failover state changes	./lustre_failover
xtdb2lustreserv	xtlustreserv2db	lustre_service	Updates the database when a file system's failover process is changed	./lustre_serv
xtdb2filesys	xtfilesys2db	filesystem	Updates the database when a file system's status changes	./filesys
xtdb2gpus	xtgpus2db	gpus	Updates the database when attributes about the accelerators change	./gpus
xtprocadmin	none	processor	Displays or sets the current value of processor flags and node attributes in the service database (SDB). The batch scheduler and ALPS are impacted by changes to these flags and attributes.	none
xtservconfig	none	service_config	Adds, removes, or modifies service configuration in the SDB service_config table	none

Change Nodes and Classes

The service_processor table tracks node IDs (NIDs) and their classes. The table is populated from the /etc/opt/cray/sdb/node_classes file on the boot node every time the system boots. Change this file to update the database when the classes of nodes change, for example, when adding login nodes. If changes are made to /etc/opt/cray/sdb/node_classes, the same changes must be made to the node class settings in CLEinstall.conf before performing an update or upgrade installation; otherwise, the install utility will complain about the inconsistency.

The `xtnodeclasses2db` command inserts the node-class list into the database. It does not make any changes to the shared root. To change the shared root, invoke the `xtnce` command.

For more information, see the `xtdb2nodeclasses(8)`, `xtnodeclasses2db(8)`, and `xtnce(8)` man pages.

Boot a Node or Set of Nodes Using the xtcli boot Command

To boot a specific image or load file on a given node or set of nodes, execute the HSS `xtcli boot` *`boot_type`* command, as shown in the following examples. When using a file for the boot image, the same file must be on both the SMW and the `bootroot` at the same path.

WARNING: Each system boot must be started with an `xtbootsys` session to establish a *`sessionid`*. Perform direct boot commands using the `xtcli boot` command only **after** a session has been established through `xtbootsys`.

Boot all service nodes with a specific image

For this example, the specific image is located at `/raw0`:

```
crayadm@smw:~> xtcli boot all_serv_img -i /raw0
```

Boot all compute nodes with a specific image

For this example, the specific image is located at */bootimagedir/bootimage*:

```
crayadm@smw:~> xtcli boot all_comp_img -i /bootimagedir/bootimage
```

Boot compute nodes using a load file

The following example boots all compute nodes in the system with using a load file name `CNL0`:

```
crayadm@smw:~> xtcli boot CNL0 -o compute s0
```

Increase the Boot Manager Timeout Value

On systems of 4,000 nodes or larger, the time that elapses until the boot manager receives all responses to the boot requests can be greater than the default 60-second time-out value. This is due, in large part, to the amount of other event traffic that occurs as each compute node generates its console output.

To avoid this problem, change the `boot_timeout` value in the `/opt/cray/hss/default/etc/bm.ini` file on the SMW to increase the default 60-second time-out value by 60 seconds for every 5,000 nodes; for example:

Increase the `boot_timeout` value

For systems of 5,000 to 10,000 nodes, change the `boot_timeout` line to:

```
boot_timeout 120
```

For systems of 10,000 to 15,000 nodes, change the `boot_timeout` line to:

```
boot_timeout 180
```

Reboot Controllers of a Cabinet or Blade

The `xtccreboot` command provides a means to reboot controllers. Options allow for rebooting all controllers of a specified type (cabinet or blade) or providing a list of controllers of a specified type to be rebooted.

For additional information, see the `xtccreboot(8)` man page.

Reboot cabinet controller c0-0, with verbose output

```
smw:~> xtccreboot -v -c c0-0
xtccreboot: /opt/cray-xt-pdsh/default/bin/pdsh -w "c0-0" /sbin/reboot
xtccreboot: reboot sent to specified CCs
```

Bounce Blades Repeatedly Until All Blades Succeed

About this task

IMPORTANT: This iterative `xtbounce` should typically be done in concert with an `xtbootsys` automation file where bounce and routing are turned off.

Procedure

1. Bounce the system.

   ```
   smw:~> xtbounce s0
   ```

2. Bounce any blades that failed the first bounce. Repeat as necessary.

3. Execute the following command, which copies route configuration files, based on the *idlist* (such as `s0`), to the blade controllers. This avoids having old, partial route configuration files left on the blades that were bounced earlier and ensures that the links are initialized correctly.

   ```
   smw:~> xtbounce --linkinit s0
   ```

4. Route and boot the system without executing `xtbounce` again. If using a `xtbootsys` automation file, specify `set data(config,xtbounce) 0`, or use the `xtbootsys --config xtbounce=0` command.

Request and Display System Routing

Use the HSS `rtr` command to request routing for the HSN, to verify current route configuration, or to display route information between nodes. Upon startup, `rtr` determines whether it is making a routing request or an information request.

For more information, see the `rtr(8)` man page.

Display routing information

The `--system-map` option to `rtr` writes the current routing information to `stdout` or to a specified file. This command can also be helpful for translating node IDs (NIDs) to physical ID names.

```
crayadm@smw:~> rtr --system-map
```

Route the entire system

The `rtr -R | --route-system` command sends a request to perform system routing. If no components are specified, the entire configuration is routed as a single routing domain based on the configuration information provided by the state manager. If a component list (*idlist*) is provided, routing is limited to the listed components. The state manager configuration further limits the routing domain to omit disabled blades, nodes, and links and empty blade slots.

```
crayadm@smw:~> rtr --route-system
```

Initiate a Network Discovery Process

Use the HSS `rtr --discover` command to initiate a network discovery process.

```
crayadm@smw:~> rtr --discover
```

IMPORTANT: The discovery process must be done on the system as a whole; it cannot be applied to individual partitions. Therefore, discovery will immediately fail if the system does not have partition `p0` enabled.

See the `rtr(8)` man page for additional information.

Configure IP Routes

Prerequisites

Configuring IP routes for compute nodes is not required on a CLE system.

About this task

An `/etc/routes` file can provide route entries for compute nodes. This provides a mechanism for administrators to configure routing access from compute nodes to login and network nodes, using external IP destinations without having to traverse RSIP tunnels. Careful consideration should be given before using this capability for general purpose routing.

The `/etc/routes` file will provide a route from the compute nodes to a gateway node (login or network). However, that gateway node must provide a connection to the network of interest (via IP forwarding, NAT, or something else). These instructions do not cover providing that connection.

Use the `simple_sync` functionality to make the `/etc/routes` file available on the compute nodes.

Procedure

Configure IP routes via `simple_sync`.

The new `/etc/routes` file is examined during startup. Non-comment, non-blank lines are passed to the route add command. The empty file contains comments describing the syntax.

To make the routes file available to the compute nodes, do the following on the SMW.

a. Edit a `routes` file with the desired compute node routes in a local directory.

```
smw# vi routes
```

b. Create the directory `etc` in the desired config set directory, `/var/opt/cray/imps/config/sets/<config set>/files/roles/simple_sync/classes/compute`. This will create an `/etc` directory on the compute nodes.

```
smw# mkdir -p /var/opt/cray/imps/config/sets/p0/files/roles/simple_sync/
classes/compute/etc
```

c. Copy the `routes` files from the local directory into the newly created `etc` directory. Then, this file will be available on all of the compute nodes when they boot.

```
smw# cp -p routes /var/opt/cray/imps/config/sets/p0/files/roles/simple_sync/
classes/compute/etc
```

Shut Down the System Using the auto.xtshutdown File

The preferred method to shut down the system is to use the `xtbootsys` command with the auto shutdown file as follows:

```
crayadm@smw:~> xtbootsys -s last -a auto.xtshutdown
```

Or, for a partitioned system with partition `pN`:

```
smw:~# xtbootsys --partition pN -s last -a auto.xtshutdown
```

This method shuts down the compute nodes (which are commonly also Lustre clients), then executes `xtshutdown` on service nodes, halting the nodes and then stopping processes on the SMW. A system administrator can shut down the system using both user-defined and built-in procedures in the `auto.xtshutdown` file, which is located on the SMW in the `/opt/cray/hss/default/etc` directory.

For related procedures, see *XC™ Series Software Initial Installation and Configuration Guide*. For more information about using automation files, see the `xtbootsys(8)` man page.

The xtshutdown Command

The `xtshutdown` command executes a series of commands locally on the boot node and service nodes to shut down the system in an orderly fashion. The sequence of shutdown steps and the nodes on which to execute them are defined by the system administrator in the `/etc/opt/cray/init-service/xtshutdown.conf` file or in the file specified by the environment variable `XTSHUTDOWN_CONF`.

Root user privileges are required to run `xtshutdown`. Passwordless `ssh` must be enabled for the root user from the boot node to all service nodes.

The `xtshutdown` command uses `pdsh` to invoke commands on the selected service nodes (i.e., boot node, SDB node, a class of nodes, or a single host). A system administrator can define functions to execute when the system is shut down. Place these functions in the `/etc/opt/cray/init-service/xt_shutdown_local` file or the file defined by the `XTSHUTDOWN_LOCAL` environment variable.

Shut Down the System or Part of the System Using the xtcli shutdown Command

The HSS `xtcli shutdown` command shuts down the system or a part of the system. To shut down compute nodes, execute the `xtcli shutdown` command. Under normal circumstances, for example to successfully disconnect from Lustre, invoking the `xtcli shutdown` command attempts to gracefully shut down the specified nodes.

For information, see the `xtcli(8)` man page.

Shut down all compute nodes

```
crayadm@smw:~> xtcli shutdown compute
```

Shut down specified compute nodes

For this example, shut down only compute nodes in cabinet `c13-2`:

```
crayadm@smw:~> xtcli shutdown c13-2
```

Shut down all nodes of a system

```
crayadm@smw:~> xtcli shutdown s0
```

Shut down a partition `pN` of a system

```
crayadm@smw:~> xtcli shutdown pN
```

Force nodes to shut down (immediate halt)

When all nodes of a system must be halted immediately, use the `-f` argument; nodes will not go through their normal shutdown process. Forced shutdown occurs even if the nodes have an alert status present.

```
crayadm@smw:~> xtcli shutdown -f s0
```

After the software on the nodes is shutdown, the system administrator can halt the hardware, reboot, or power down.

Shut Down Service Nodes

Prerequisites

`Root` user privileges are required to run `xtshutdown`. Passwordless `ssh` must be enabled for the `root` user from the boot node to all service nodes.

CAUTION: The `xtshutdown` command does not shut down compute nodes. To shut down the compute and service nodes, see *Shut Down the System or Part of the System Using the xtcli shutdown Command*.

About this task

For information about shutting down service nodes, see the `xtshutdown(8)` man page.

Procedure

1. Modify the `/etc/opt/cray/init-service/xtshutdown.conf` file or the file specified by the `XTSHUTDOWN_CONF` environment variable to define the sequence of shutdown steps and the nodes on which to execute them. The `/etc/opt/cray/init-service/xtshutdown.conf` file resides on the boot node.

2. If desired, define functions to execute when the system is shut down. Place these functions in the `/etc/opt/cray/init-service/xt_shutdown_local` file or the file defined by the `XTSHUTDOWN_LOCAL` environment variable.

3. Execute `xtshutdown`.

   ```
   boot:~ # xtshutdown
   ```

   After the software on the nodes is shutdown, the administrator can halt the hardware, reboot, or power down.

Stop System Components

When a system administrator removes, stops, or powers down components, any applications and compute processes that are running on those components are lost.

Reserve a Component

To allow applications and compute processes to complete before stopping components, use the HSS `xtcli set_reserve` *`idlist`* command to prevent the selected nodes from accepting new jobs.

A node running CNL and using ALPS is considered to be down by ALPS after it is reserved using the `xtcli set_reserve` command. The output from `apstat` will show the node as down (DN), even though there may be an application running on that node. This DN designation indicates that no other work will be placed on the node after the currently running application has terminated.

For more information, see the `xtcli_set(8)` man page.

Reserve a component

```
crayadm@smw:~> xtcli set_reserve idlist
```

Power Down Blades or Cabinets

WARNING: Power down the cabinets with software commands. Tripping the circuit breakers may result in damage to system components.

WARNING: Before powering down a blade or a cabinet, ensure the operating system is not running.

The `xtcli power down` command powers down the specified cabinet and/or blades within the specified partition, chassis or list of blades. Cabinets must be in the `READY` state to receive power commands. See *System Component States*. The `xtcli power down` command has the following form, where *`physIDlist`* is a comma-separated list of cabinets or blades present on the system.

```
xtcli power down physIDlist
```

The `xtcli power force_down` and `xtcli power down_slot` commands are aliases for the `xtcli power down` command. For information about disabling and enabling components, see *Disable Hardware Components*, and *Enable Hardware Components*, respectively.

WARNING: Although a blade is powered off, the HSS in the cabinet is live and has power.

For information about powering down a component, see the `xtcli_power(8)` man page.

Power down a specified blade

For this example, power down a blade with the ID `c0-0c0s7`:

```
crayadm@smw:~> xtcli power down c0-0c0s7
```

Halt Selected Nodes

Use the HSS `xtcli halt` command to halt selected nodes. For more information, see the `xtcli(8)` man page.

Halt a node

For this example, halt node `157`:

```
crayadm@smw:~> xtcli halt 157
```

Restart a Blade or Cabinet

IMPORTANT: Change the state of the hardware only when the operating system is not running or is shut down.

The `xtcli power up` command powers up the specified cabinet and/or blades within the specified partition, chassis or list of blades. Cabinets must be in the `READY` state (see *System Component States*) to receive power commands. The `xtcli power up` command does not attempt to power up network mezzanine cards or nodes that are handled by the `xtbounce` command during system boot.

The `xtcli power up_slot` command is an alias for the `xtcli power up` command.

The `xtcli power up` command has the following form, where *`physIDlist`* is a comma-separated list of cabinets or blades present on the system.

```
xtcli power up physIDlist
```

For more information, see the `xtcli_power(8)` man page.

Power up blades in `c0-0c0s7`

```
crayadm@smw:~> xtcli power up c0-0c0s7
```

Abort Active Sessions on the HSS Boot Manager

About this task

Use the HSS `xtcli session abort` command to abort sessions in the boot manager. A session corresponds to executing a specific command such as `xtcli power up` or `xtcli boot`.

For more information about manager sessions, see the `xtcli(8)` man page.

Procedure

1. Display all running sessions in the boot manager. Only the boot manager supports multiple simultaneous sessions.

   ```
   crayadm@smw:~> session show BM all
   ```

2. Abort the selected session, *`session_id`*.

   ```
   crayadm@smw:~>  xtcli session abort BM session_id
   ```

Display and Change Software System Status

The user command `xtnodestat` provides a display of the status of nodes: how they are allocated and to what jobs. The `xtnodestat` command provides current job and node status summary information, and it provides an interface to ALPS and jobs running on CNL compute nodes. ALPS must be running in order for `xtnodestat` to report job information.

For more information, see the `xtnodestat(1)` man page.

Configure Current System Timezone

Prerequisites

Start with the XC system booted.

About this task

Changing the timezone of a system can be done with a few configuration changes and then rebooting components.

Procedure

Check current timezone

1. Check timezone on SMW.

```
smw# date
```

2. Check timezone on cabinet and blade controllers.

```
smw# xtrsh -l root -s date
```

3. Check timezone on boot node.

```
smw# ssh boot date
```

4. Check timezone on SDB node. This command works from the SMW if the SDB node is a tier1 node with an Ethernet connection to the SMW.

```
smw# ssh sdb date
```

5. Check timezone on all service nodes.

```
smw# ssh sdb pcmd -r -n ALL_SERVICE_NOT_ME "date"
```

6. Check timezone on all compute nodes.

```
smw# ssh sdb pcmd -r -n ALL_COMPUTE "date"
```

Change SMW local timezone

7. Execute this command to change the default timezone. The default timezone on the SMW is "America/Chicago".

```
smw# yast2 timezone
```

The change on the SMW will be immediate, but users will need to logout and then login again to get the new environment.

This does not change the timezone for the CLE nodes or the cabinet and blade controllers. See below to make those changes.

Change timezone in global config set

8. Set `cray_time.settings.service.data.timezone` to be the desired timezone. A list of possible timezones is available on the SMW in `/usr/share/zoneinfo/zone1970.tab`.

```
smw# cfgset update -s cray_time -m interactive global
```

9. Validate the config set.

```
smw# cfgset validate global
```

Change timezone in CLE config set

If the CLE config set has `cray_time.inherit` set to true, then the timezone and other time settings from the global config set will be inherited by the CLE config set.

If the CLE config set has `cray_time.inherit` set to false, then use the following command to change the setting.

10. Set `cray_time.settings.service.data.timezone` to be the desired timezone. A list of possible timezones is available on the SMW in `/usr/share/zoneinfo/zone1970.tab`.

```
smw# cfgset update -s cray_time -m interactive p0
```

11. Validate the config set.

```
smw# cfgset validate p0
```

Reboot for new timezone

Follow these steps to set a new timezone for all components in the SMW and CLE system after the global and CLE config sets and SMW yast2 have been updated with the new setting.

12. Reboot SMW.

a. Shutdown CLE and reboot the SMW.

```
crayadm@smw> xtbootsys -s last -a auto.xtshutdown
crayadm@adm> su - root
smw# reboot
```

b. Check that the SMW has the desired timezone setting once the SMW reboots.

```
smw# date
```

13. Power down the system.

```
smw# xtcli power down s0
```

14. Reboot the cabinet controllers

```
smw# xtccreboot -c all
xtccreboot: reboot sent to specified CCs
smw# sleep 120
smw# xtalive -l cc
```

15. Power up the system.

```
smw# xtcli power up s0
```

16. Boot CLE nodes for new timezone.

```
crayadm@smw> xtbootsys -a auto.rhine
```

17. Check current timezone.

a. Check timezone on SMW.

```
smw# date
```

b. Check timezone on cabinet and blade controllers.

```
smw# xtrsh -l root -s date
```

c. Check timezone on boot node.

```
smw# ssh boot date
```

d. Check timezone on SDB node. This command works from the SMW if the SDB node is a tier1 node with an Ethernet connection to the SMW.

```
smw# ssh sdb date
```

e. Check timezone on all service nodes.

```
smw# ssh sdb pcmd -r -n ALL_SERVICE_NOT_ME "date"
```

f. Check timezone on all compute nodes.

```
smw# ssh sdb pcmd -r -n ALL_COMPUTE "date"
```

View and Change the Status of Nodes

Use the `xtprocadmin` command on a service node to view the status of components of a booted system in the `processor` table of the SDB. The command enables the system administrator to retrieve or set the processing mode (`interactive` or `batch`) of specified nodes. The administrator can display the state (`up`, `down`, `admindown`, `route`, or `unavailable`) of the selected components, if needed. The administrator can also allocate processor slots or set nodes to become unavailable at a particular time. The node is scheduled only if the status is `up`.

When the `xtprocadmin -ks` option is used, then the option can either a normal argument (`up`, `down`, etc.), or it can have a colon in it to represent a conditional option; for example, the option of the form `up:down` means "if state was up, mark down".

For more information, see the `xtprocadmin(8)` man page.

View node characteristics

```
login:~> xtprocadmin
   NID    (HEX)    NODENAME     TYPE     STATUS        MODE
     1      0x1  c0-0c0s0n1  service         up       batch
     2      0x2  c0-0c0s0n2  service         up       batch
     5      0x5  c0-0c0s1n1  service         up       batch
     6      0x6  c0-0c0s1n2  service         up       batch
     8      0x8  c0-0c0s2n0  compute         up       batch
     9      0x9  c0-0c0s2n1  compute         up       batch
    10      0xa  c0-0c0s2n2  compute         up       batch
    11      0xb  c0-0c0s2n3  compute         up       batch
```

View all node attributes

```
login:~> xtprocadmin -A
   NID    (HEX)    NODENAME     TYPE ARCH        OS CPUS  CU AVAILMEM   PAGESZ CLOCKMHZ GPU SOCKETS DIES C/
CU                       LABEL0                          LABEL1
LABEL2                     LABEL3
     1      0x1  c0-0c0s0n1  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     2      0x2  c0-0c0s0n2  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     5      0x5  c0-0c0s1n1  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     6      0x6  c0-0c0s1n2  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     8      0x8  c0-0c0s2n0  compute   xt       CNL   32  16    65536     4096     2600   0       2    2
2
     9      0x9  c0-0c0s2n1  compute   xt       CNL   32  16    65536     4096     2600   0       2    2
2
    10      0xa  c0-0c0s2n2  compute   xt       CNL   32  16    65536     4096     2600   0       2    2
2
```

View selected attributes of selected nodes

For this example, the `-a` option lists the selected attributes to display:

```
login:~> xtprocadmin -n 8 -a arch,clockmhz,os,cores
   NID    (HEX)    NODENAME     TYPE ARCH CLOCKMHZ        OS CPUS
     8      0x8  c0-0c0s2n0  compute   xt     2600       CNL   32
```

Disable a node

For this example, the `admindown` option disables node `c0-0c0s3n1` such that it cannot be allocated:

```
crayadm@nid00004:~> xtprocadmin -n c0-0c0s3n1 -k s admindown
```

Disable all processors

```
crayadm@nid00004:~> xtprocadmin -k s admindown
```

Perform Parallel Operations on Compute Nodes

The parallel command tool (`pcmd`) facilitates execution of the same commands on groups of compute nodes in parallel, similar to `pdsh`. Although `pcmd` is launched from a service node, it acts on compute nodes. It allows administrators and/or, if the site deems it feasible, other users to securely execute programs in parallel on compute nodes. The user can specify on which nodes to execute the command. Alternatively, the user can specify an application ID (*`apid`*) to execute the command on all the nodes available under that *`apid`*.

An unprivileged user must execute the command targeting nodes where the user is currently running an `aprun`. A `root` user is allowed to target any compute node, regardless of whether there are jobs running there or not. In either case, if the `aprun` exits and the associated applications are killed, any commands launched by `pcmd` will also exit.

By default, `pcmd` is installed as a `root`-only tool. It must be installed as `setuid root` in order for unprivileged users to use it.

The `pcmd` command is located in the `nodehealth` module. If the `nodehealth` module is not part of the default profile, load it by specifying:

```
module load nodehealth
```

For additional information, see the `pcmd(1)` man page.

Perform Parallel Operations on Service Nodes

Use `pdsh`, the CLE parallel remote shell utility for service nodes, to issue commands to groups of nodes in parallel. The system administrator can select the nodes on which to use the command, exclude nodes from the command, and limit the time the command is allowed to execute. Only user `root` can execute the `pdsh` command. The command has the following form:

```
pdsh [options] command
```

For more information, see the `pdsh(1)` man page.

Restart the NTP service

```
boot:~ # pdsh -w 'login[1-9]' /etc/init.d/ntp restart
```

Mark a Compute Node as a Service Node

Use the `xtcli mark_node` command to mark a node in a compute blade to have a role of `service` or `compute`; `compute` is the default. It is not permitted to change the role of a node on a service blade, which always has the `service` role.

Marking a node on a compute blade as `service` or `compute` allows the administrator to load the desired boot image at boot time. Compute nodes marked as `service` can run software-based services. A request to change the role of a running node (that is, the node is in the `ready` state and the operating system is running) will be denied.

For more information, see the `xtcli(8)` man page and *Check the Status of System Components* on page 143.

Find Node Information

Translate Between Physical ID Names and Integer NIDs

To translate between physical ID names (cnames) and integer NIDs, generate a system map on the System Management Workstation (SMW) and filter the output, enter the following command:

```
crayadm@smw:~> rtr --system-map | grep cname | awk '{ print $1 }'
```

To translate between physical ID names (rnames) and integer NIDs, generate a system map on the System Management Workstation (SMW) and filter the output, enter the following command:

```
crayadm@smw:~> rtr --system-map | grep rname | awk '{ print $1 }'
```

For more information, see the `rtr(8)` man page.

Find Node Information Using the `xtnid2str` Command

The `xtnid2str` command converts numeric node identification values to their physical names (cnames). This allows conversion of Node ID values, ASIC NIC address values, or ASIC ID values.

For additional information, see the `xtnid2str(8)` man page.

Find the physical ID for node 38

```
smw:~> xtnid2str 28
node id 0x26 = 'c0-0c0s1n2'
```

Find the physical ID for nodes 0, 1, 2, and 3

```
smw:~> xtnid2str 0 1 2 3
node id 0x0 = 'c0-0c0s0n0'
node id 0x1 = 'c0-0c0s0n1'
node id 0x2 = 'c0-0c0s1n0'
node id 0x3 = 'c0-0c0s1n1'
```

Find the physical IDs for Aries IDs 0-7

```
smw:~> xtnid2str -a 0-7
aries id 0x0 = 'c0-0c0s0a0'
aries id 0x1 = 'c0-0c0s1a0'
aries id 0x2 = 'c0-0c0s2a0'
aries id 0x3 = 'c0-0c0s3a0'
aries id 0x4 = 'c0-0c0s4a0'
aries id 0x5 = 'c0-0c0s5a0'
aries id 0x6 = 'c0-0c0s6a0'
aries id 0x7 = 'c0-0c0s7a0'
```

Find Node Information Using the `nid2nic` Command

The `nid2nic` command prints the *nid*-to-*nic* address mappings, *nic*-to-*nid* address mappings, and a specific *physical_location*-to-*nic* address and *nid* mappings.

For information about using the `nid2nic` command, see the `nid2nic(8)` man page.

Print the *nid*-to-*nic* address mappings for the node with NID 31

```
smw:~> nid2nic 31
NID:0x1f        NIC:0x21        c0-0c0s7n3
```

Print the *nid*-to-*nic* address mappings for the node with NID 31, but specify the NIC value in the command line

```
smw:~> nid2nic -n 0x21
NIC:0x21        NID:0x1f        c0-0c0s7n3
```

Display and Change Hardware System Status

A system administrator can execute commands that look at and change the status of the hardware.

CAUTION: Execute commands that change the status of hardware only when the operating system is shut down.

Generate HSS Physical IDs

The HSS `xtgenid` command generates HSS physical IDs, for example, to create a list of blade controller identifiers for input to the flash manager. Selection can be restricted to components of a particular type. Only user `root` can execute the `xtgenid` command.

For more information, see the `xtgenid(8)` man page.

Create a list of node identifiers that are not in the `DISABLE`, `EMPTY`, or `OFF` state

```
smw:~ # xtgenid -t node --strict
```

Disable Hardware Components

If links, nodes, or Cray ASICs have hardware problems, the system administrator can direct the system to ignore the components with the `xtcli disable` command.

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

The `xtcli disable` command has the following form, where *`idlist`* is a comma-separated list of components (in cname format) that the system is to ignore. The system disregards these links or nodes.

```
xtcli disable [{-t type [-a] } | -n] [-f] idlist
```

IMPORTANT: The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

Disabling of a cabinet, chassis, or blade will fail if any nodes under the component are in the `ready` state, unless the force option (`-f`) is used. An error message will indicate the reason for the failure.

Disabling of a node in the `ready` state will fail, unless the force option (`-f`) is used. An error message will indicate the reason for the failure.

The state of `empty` components will not change when using the `disable` command, unless the force option (`-f`) is used.

For detailed information about using the `xtcli disable` command, see the `xtcli(8)` man page.

Disable the Aries ASIC `c0-0c1s3a0`

1. Determine that the ASIC is in the `OFF` state.

   ```
   crayadm@smw:~> xtcli status -t aries c0-0c1s3a0
   ```

2. If the ASIC is not in the `OFF` state, power down the blade that contains the ASIC.

   ```
   crayadm@smw:~> xtcli power down c0-0c1s3
   ```

3. Disable the ASIC.

   ```
   crayadm@smw:~> xtcli disable c0-0c1s3a0
   ```

4. Power up the blade that contains the ASIC.

   ```
   crayadm@smw:~> xtcli power up c0-0c1s3
   ```

Enable Hardware Components

If links, nodes, or Cray ASICs that have been disabled are later fixed, the system administrator can add them back to the system with the `xtcli enable` command.

The `xtcli enable` command has the following form, where *`idlist`* is a comma-separated list of components (in cname format) for the system to recognize.

```
xtcli enable [{-t type [-a] } | -n] [-f] idlist
```

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

> **IMPORTANT:** The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

The state of `empty` components does not change when using the `xtcli enable` command, unless the force option (`-f`) is used.

The state of `off` means that a component is present on the system. If the component is a blade controller, node, or ASIC, then this will also mean that the component is powered off. If the administrator disables a component, the state shown becomes `disabled`. When the `xtcli enable` command is used to enable that component for use once again, its state switches from `disabled` to `off`. In the same manner, enabling an empty component means that its state switches from `empty` to `off`.

For more information, see the `xtcli(8)` man page.

Set Hardware Components to `EMPTY`

Use the `xtcli set_empty` command to set a selected component to the `EMPTY` state. HSS managers and the `xtcli` command ignore empty or disabled components.

Setting a selected component to the `EMPTY` state is typically done when a component, usually a blade, is physically removed. By setting it to `EMPTY`, the system ignores it and routes around it.

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

> **IMPORTANT:** The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

For more information, see the `xtcli(8)` man page.

Set a blade to the `EMPTY` state

```
crayadm@smw:~> xtcli set_empty -a c0-0c1s7
```

Lock Hardware Components

Components are automatically locked when a command that can change their state is running. As the command is started, the state manager locks these components so that nothing else can affect their state while the command executes. When the manager is finished with the command, it unlocks the components.

Use the HSS `xtcli lock` command to lock components. Locking a component prints out the state manager session ID.

For more information, see the `xtcli(8)` man page.

Lock cabinet `c0-0`

```
crayadm@smw:~> xtcli lock -l c0-0
```

Show all session (lock) data

```
crayadm@smw:~> xtcli lock show
```

Unlock Hardware Components

Use the HSS `xtcli lock` command to unlock components. This command is useful when an HSS manager fails to unlock some set of components.

The system administrator can manually check for locks with the `xtcli lock show` command and then unlock them. Unlocking a component does not print out the state manager session ID. The `-u` option must be used to unlock a component as follows:

```
crayadm@smw:~> xtcli lock -u lock_number
```

Where *`lock_number`* is the value given when initiating the lock; it is also indicated in the `xtcli lock show` query. Unlocking does nothing to the state of the component other than to release locks associated with it.

HSS daemons cannot affect components that are locked by a different session.

xtbounce Error Message Indicates Cabinet Controller and Its Blade Controllers Not in Sync

During the `gather_cab_pwr_states` phase of `xtbounce`, if the HSS software on a cabinet controller and any of its blade controllers is out of sync, error messages such as the following will be printed during the `xtbounce`.

```
***** gather_cab_pwr_states *****
18:28:42 - Beginning to wait for response(s)

ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
```

If this occurs, it indicates that the blade controller software is at a different revision than the cabinet controller software. `xtbounce` will print a list of cabinets for which this error has occurred. The message will be similar to the following:

```
ERROR: power state check error on 2 cabinet(s)
WARNING: unable to find c0-0 in err_cablist
WARNING: unable to find c0-2 in err_cablist
```

This error is an indication that when the HSS software was previously updated, the cabinet controllers and the blade controllers were not updated to the same version.

To correct this error, cancel out of `xtbounce` (with **Ctrl-C**), wait approximately five minutes for the `xtbounce` related activities on the blade controllers to finish, then reboot the cabinet controller(s) and their associated blade controllers to get the HSS software synchronized. Following this, the `xtbounce` may be executed once again.

Power-cycle a Component to Handle Bus Errors

About this task

Bus errors are caused by machine-check exceptions. If a bus error occurs, try power-cycling the component.

Procedure

1. Power down the components. The *physIDlist* is a comma-separated list of components present on the system.

   ```
   crayadm@smw:~> xtcli power down physIDlist
   ```

2. Power up the components.

   ```
   crayadm@smw:~> xtcli power up physIDlist
   ```

When a Component Fails

Components that fail are replaced as field replaceable units (FRUs). FRUs include compute blade components, service blade components, and power and cooling components.

When a field replaceable unit (FRU) problem arises, contact a Customer Service Representative to schedule a repair.

Dump and Reboot Nodes Automatically

The SMW daemon `dumpd` initiates automatic dump and reboot of nodes when requested by the Node Health Checker (NHC).

CAUTION: The `dumpd` daemon is invoked automatically by `xtbootsys` on system (or partition) boot. In most cases, system administrators do not need to use this daemon directly.

A system administrator can set global variables in the `/etc/opt/cray/nodehealth/nodehealth.conf` configuration file to control the interaction of NHC and `dumpd`. For more information about NHC and the `nodehealth.conf` configuration file, see *Configure the Node Health Checker (NHC)*.

Variables can also be set in the `/etc/opt/cray-xt-dumpd/dumpd.conf` configuration file on the SMW to control how `dumpd` behaves on the system.

Each CLE release package also includes an example `dumpd` configuration file, `/etc/opt/cray-xt-dumpd/dumpd.conf.example`. The `dumpd.conf.example` file is a copy of the `/etc/opt/cray-xt-dumpd/dumpd.conf` file provided for an initial installation.

> **IMPORTANT:** The `/etc/opt/cray-xt-dumpd/dumpd.conf` file is not overwritten during a CLE upgrade if the file already exists. This preserves the site-specific modifications previously made to the file. Cray recommends comparing the site's `/etc/opt/cray-xt-dumpd/dumpd.conf` file content with the `/etc/opt/cray-xt-dumpd/dumpd.conf.example` file provided with each release to identify any changes and then update the site's `/etc/opt/cray-xt-dumpd/dumpd.conf` file accordingly.
>
> If the `/etc/opt/cray-xt-dumpd/dumpd.conf` file does not exist, then the `/etc/opt/cray-xt-dumpd/dumpd.conf.example` file is copied to the `/etc/opt/cray-xt-dumpd/dumpd.conf` file.

The CLE installation and upgrade processes automatically install `dumpd` software, but it must be explicitly enabled.

Collect Debug Information From Hung Nodes Using the xtnmi Command

CAUTION: This is not a harmless tool to use to repeatedly get information from a node at various times; only use this command when debugging data from nodes that are in trouble is needed. The `xtnmi` command output may be used to determine problems such as a core hang. `xtnmi` will stop a running node. It is best used when a node is not running correctly and debugging information is needed, or to stop a node that is running incorrectly.

The sole purpose of the `xtnmi` command is to collect debug information from unresponsive nodes. As soon as that debug information is displayed to the console, the node panics.

For additional information, see the `xtnmi(8)` man page.

Modify BIOS Parameters

There are a few, rare circumstances where it may be necessary to modify BIOS parameters, for example, in order to troubleshoot a problem, or if there is a need to test a new BIOS version on a small set of nodes before implementing the change across an entire system.

The `xtbiosconf` command allows administrators to specify BIOS parameters at the node, blade, chassis, or cabinet level. BIOS parameters can be associated with a BIOS revision, numeric parameter offset or parameter name, and target nodes. BIOS revision wildcards are supported. The BIOS parameter data is saved in a database

on the SMW, and made available automatically to blade controllers via the ERFS file system. In most cases a cold reboot of the affected nodes is needed to apply the new settings.

CAUTION: *Do not* attempt to use this command except under guidance by Cray support personnel, who will provide all the steps for shutting down the nodes, changing the settings, and bringing the nodes back up. Improper use of this command can damage a system.

The following command displays the current BIOS Parameter settings for the entire system:

```
smw~> xtbiosconf --show s0
==============|======|==================================
              | BIOS | BIOS
Node          | REV  | Parameter
==============|======|==================================
c0-1c0s0n1    | 4030 | numlock=1
c0-1c0s0n1    | 4030 | acpiauto=0
==============|======|==================================
c0-1c0s0n2    | 4030 | numlock=1
c0-1c0s0n2    | 4030 | acpiauto=0
==============|======|==================================
```

For more information see the `xtbiosconf` man page.

Increase File System Size

About this task

When a `btrfs` or `xfs` file system on the boot RAID needs to be increased, both the `cray_bootraid_config.yaml` file needs to be changed for the new size and the commands to grow the file system need to be done.

Procedure

1. Edit the `cray_bootraid_config.yaml` file to increase the size for the filesystem which needs to grow.

   ```
   smw# vi /var/opt/cray/imps/config/sets/global/config/cray_bootraid_config.yaml
   ```

   For example, to increase the size of the `/var/opt/cray/imps` file system on the SMW, locate the "`smwdefault`" storage set, the `smw_node_vg` volume group, and the "`home`" volume within that storage set. Change the "`fs_size`" for imps from 600 to 800.

   Increase the size of the `/home` file system on the SMW, in the "`smwdefault`" storage set, the "`smw_node_vg`" volume group, and the "`home`" volume within that storage set. Change the "`fs_size`" for imps from 50 to 100.

   ```
            -   key: smwdefault
                volume_groups:
                -   key: smw_node_vg
                    owner: smw
                    devices:
                    - /dev/disk/by-id/wwn-0x600a0980006b47b7000000e5561260a7
                    volumes:
                    -   key: home
                        description: LVM volume for user home directories on the
   ```

```
SMW.
                        type: lvm
                        fs_type: xfs
                        fs_size: 50
                        fs_mount_point: /home
                        snapshot: false
                        mount_options:
                    -   key: imps
                        description: LVM Volume for storage of IMPS
configuration.
                        type: lvm
                        fs_type: btrfs
                        fs_size: 600
                        fs_mount_point: /var/opt/cray/imps
                        snapshot: false
                        mount_options:
```

2. Extend an LVM volume.

   a. Extend the "`home`" volume in the "`smw_node_vg`" LVM volume from the existing size to 100GB.

   ```
   smw# lvextend -L100G /dev/mapper/smw_node_vg-home
   ```

   b. Extend the "`imps`" volume in the "`smw_node_vg`" LVM volume from the existing size to 800GB.

   ```
   smw# lvextend -L800G /dev/mapper/smw_node_vg-imps
   ```

3. Grow a `btrfs` file system.

   ```
   smw# btrfs filesystem resize max /var/opt/crayimps
   ```

4. Grow an `xfs` file system.

   ```
   smw# xfs_growfs /home
   ```

Add New Hardware to a System

About this task

Whether adding a single compute blade or a single service blade or several components in a full cabinet or several cabinets, the process is similar.

Procedure

1. Add new components to system partition.

   a. If the system is partitioned, then add the new components to the specific partition. If the system is not partitioned, then this step can be skipped.

   ```
   crayadm@smw> xtcli part_cfg show p2
   crayadm@smw> xtcli part_cfg deactivate p2
   ```

   b. Update the members of the partition with the old components and the new components.

```
crayadm@smw> xtcli part_cfg update p2 -m
c2-0c0s0,c2-0c0s1,c2-0c0s7,c0-0c0s9,c2-0c0s11,c2-0c0s13,c2-0c0s15,c2-0c0s3
crayadm@smw> xtcli part_cfg activate p2
```

2. Ensure new components are not disabled and are assigned to the desired partition. If they are disabled, they will not be discovered. If they are not assigned to a partition, they will not be bounced during the `xtdiscover` process, and therefore will not be properly discovered.

   Full system:

   ```
   crayadm@smw> xtcli status s0
   ```

   Partitioned system:

   ```
   crayadm@smw> xtcli status p1
   crayadm@smw> xtcli status p2
   ```

3. Discover the new hardware.

   ```
   crayadm@smw> su -
   smw# xtdiscover
   smw# exit
   ```

   a. Run `rtr --discover` if there is a significant change modifying the routing configuration.

      Full system:

      ```
      crayadm@smw> rtr --discover
      ```

      Partitioned system:

      ```
      crayadm@smw> xtcli part_cfg deactivate p1
      crayadm@smw> xtcli part_cfg deactivate p2
      crayadm@smw> xtcli part_cfg activate p0
      crayadm@smw> rtr --discover
      crayadm@smw> xtcli part_cfg deactivate p0
      crayadm@smw> xtcli part_cfg activate p1
      crayadm@smw> xtcli part_cfg activate p2
      ```

   b. Confirm the new components are now seen.

      ```
      crayadm@smw> xtcli status s0
      ```

      If the new components do not show up properly in the status output, do not continue. Power cycle the whole system, try the `xtdiscover` again. If they still are not showing, there may be a problem with the new hardware components.

4. Update firmware on new components. Check whether any firmware needs to be updated on the various controllers.

   ```
   crayadm@smw> xtzap -r -v s0
   ```

   If any are out of date, output like the following from the `xtzap` command will be seen and the firmware needs to be updated.

   ```
   Individual Revision Mismatches:

   Type       ID                Expected   Installed
   ---------- ----------------- ---------- ----------------------------------------
   ```

```
cc_bios    c0-0              0013       0012
bc_bios    c0-0c0s0          0013       0012
bc_bios    c0-0c0s1          0013       0012
bc_bios    c0-0c0s2          0013       0012
bc_bios    c0-0c0s3          0013       0012
```

a. Update firmware, if not all current.

   CAUTION: The `xtzap` command is normally intended for use by Cray Service personnel only. Improper use of this restricted command can cause serious damage to the computer system.

   If the output of `xtzap` includes a "Revision Mismatches" section, then some firmware is out of date and needs to be reflashed. To update, run `xtzap` with one or more of the options described in the next paragraph.

   While the `xtzap -a` command can be used to update all components with a single command, it may be faster to use the `xtzap -blade` command when only blade types need to be updated, or the `xtzap -t` command when only a single type needs to be updated. On larger systems, this can save significant time.

   This is the list of all cabinet level components:

   ```
   cc_mc (CC Microcontroller)
   cc_bios (CC Tolapai BIOS)
   cc_fpga (CC FPGA)
   chia_fpga (CHIA FPGA)
   ```

   This is a list of all blade level components:

   ```
   cbb_mc (CBB BC Microcontroller)
   ibb_mc (IBB BC Microcontroller)
   anc_mc (ANC BC Microcontroller)
   bc_bios (BC Tolapai BIOS)
   lod_fpga (LOD FPGA)
   node_bios (Node BIOS)
   loc_fpga (LOC FPGA)
   qloc_fpga (QLOC FPGA)
   ```

   If the output of the `xtzap` command shows that only a specific type needs to be updated, then use the `-t` option with that type (this example uses the `node_bios` type).

   ```
   crayadm@smw:~> xtzap -t node_bios s0
   ```

   If the output of the `xtzap` command shows that only blade component types need to be updated, then use the `-b` option:

   ```
   crayadm@smw:~> xtzap -b s0
   ```

   If the output of the `xtzap` command shows that only cabinet component types need to be updated, then use the `-c` option:

   ```
   crayadm@smw:~> xtzap -c s0
   ```

   If the output of the `xtzap` command shows that both blade- and cabinet-level component types need to be updated, or if unsure of what needs to be updated, then use the `-a` option:

   ```
   crayadm@smw:~> xtzap -a s0
   ```

b. Perform `xtbounce --linktune`, if not all current. Force `xtbounce` to do a linktune on the full system before checking firmware again.

```
crayadm@smw> xtbounce --linktune=all s0
```

c. Check firmware, after update and linktune. After updating them, confirm that they were all updated.

```
crayadm@smw> xtzap -r -v s0
```

5. Check routing configuration of the system.

The `rtr -R` command produces no output unless there is a routing problem.

Full system:

```
crayadm@smw> rtr -R s0
```

Partitioned system:

```
crayadm@smw> rtr -R p1
crayadm@smw> rtr -R p2
```

6. Update NIMS for new components. Now that the new components have been added, and the firmware is up to date, several NIMS commands are needed.

a. See what settings are for already existing similar nodes.

```
crayadm@smw> cnode list -p p0
```

b. If this blade was swapped out and replaced with a different type (that is, was compute, swapped for service), remove it from the old group.

```
crayadm@smw> cnode update --partition p1 -c p1 -G netroot_compute
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3
```

c. Assign the nodes to the correct config set, group (`compute`, `netroot_compute`, `service`, `login`, `dal`, etc.) and image.

```
crayadm@smw> cnode update --partition p1 -c p1 -g service -i /var/opt/cray/
imps/boot_images/service_XXX.cpio c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3
```

d. If this is a `netroot_compute` node, assign the key for `netroot` (can be combined with the config set, group and image assignment in above command).

```
crayadm@smw> cnode update --partition p1 -s netroot=compute-large_cle_XXX
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3
```

e. If this was a `netroot_compute` and is not anymore, remove the key.

```
crayadm@smw> cnode update --partition p1 -K netroot
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3
```

f. If this was a compute node, and is now a service, remove the rest of the extraneous keys.

```
crayadm@smw> cnode update --partition p1 -c p1 -K hsn_ipv4_mask
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3'
crayadm@smw> cnode update --partition p1 -c p1 -K hsn_ipv4_net
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3'
crayadm@smw> cnode update --partition p1 -c p1 -K sdbnodeip
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3'
crayadm@smw> cnode update --partition p1 -c p1 -K bootnodeip
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3'
```

7. Update config sets with the new components. This will generate a new `/etc/hosts` file for the CLE nodes.

   Full system:

   ```
   crayadm@smw> su -
   smw# cfgset update p0
   smw# exit
   ```

   Partitioned system:

   ```
   crayadm@smw> su -
   smw# cfgset update p1
   smw# cfgset update p2
   smw# exit
   ```

8. Update any workload manager configuration as specified in their documentation. For internal systems running native slurm, see *http://oskernel/wiki/Workload_Managers_Rhine_Redwood#Slurm*.

9. Boot the system using the standard boot procedure.

Add a New Disk to a Volume Group in a Storage Set

About this task

When more disk space is needed in an LVM volume group, add another physical volume to the `cray_bootraid_config.yaml` file and rerun `cray-ansible` for the node which owns the storage.

Procedure

1. Edit the `cray_bootraid_config.yaml` file to add another physical device to the list of devices in the volume group.

   ```
   smw# vi /var/opt/cray/imps/config/sets/global/config/cray_bootraid_config.yaml
   ```

   For example, to add a new disk device called "`/dev/disk/by-id/wwn-0x600a0980006b47b7000000e756127f9d`" to the "`smw_node_vg`" volume group, add the new disk device. Change this entry:

   ```
           -   key: smwdefault
               volume_groups:
               -   key: smw_node_vg
                   owner: smw
                   devices:
                   - /dev/disk/by-id/wwn-0x600a0980006b47b7000000e5561260a7
   ```

   To be this:

   ```
           -   key: smwdefault
               volume_groups:
               -   key: smw_node_vg
                   owner: smw
                   devices:
                   - /dev/disk/by-id/wwn-0x600a0980006b47b7000000e5561260a7
                   - /dev/disk/by-id/wwn-0x600a0980006b47b7000000e756127f9d
   ```

2. Run `cray-ansible` on the node.

   If the storage was added to the SMW volume group.

   ```
   smw# /media/SMW/SMWinstall --mode=provision-storage
   ```

   If the storage was added to the boot node volume group.

   ```
   boot# /etc/init.d/cray-ansible start
   ```

   If the storage was added to the SDB node volume group.

   ```
   sdb# /etc/init.d/cray-ansible start
   ```

Reboot Controllers of a Cabinet or Blade

The `xtccreboot` command provides a means to reboot controllers. Options allow for rebooting all controllers of a specified type (cabinet or blade) or providing a list of controllers of a specified type to be rebooted.

For additional information, see the `xtccreboot(8)` man page.

Reboot cabinet controller c0-0, with verbose output

```
smw:~> xtccreboot -v -c c0-0
xtccreboot: /opt/cray-xt-pdsh/default/bin/pdsh -w "c0-0" /sbin/reboot
xtccreboot: reboot sent to specified CCs
```

Bounce Blades Repeatedly Until All Blades Succeed

About this task

IMPORTANT: This iterative `xtbounce` should typically be done in concert with an `xtbootsys` automation file where bounce and routing are turned off.

Procedure

1. Bounce the system.

   ```
   smw:~> xtbounce s0
   ```

2. Bounce any blades that failed the first bounce. Repeat as necessary.

3. Execute the following command, which copies route configuration files, based on the *`idlist`* (such as `s0`), to the blade controllers. This avoids having old, partial route configuration files left on the blades that were bounced earlier and ensures that the links are initialized correctly.

   ```
   smw:~> xtbounce --linkinit s0
   ```

4. Route and boot the system without executing `xtbounce` again. If using a `xtbootsys` automation file, specify `set data(config,xtbounce) 0`, or use the `xtbootsys --config xtbounce=0` command.

Shut Down the System Using the auto.xtshutdown File

The preferred method to shut down the system is to use the `xtbootsys` command with the auto shutdown file as follows:

```
crayadm@smw:~> xtbootsys -s last -a auto.xtshutdown
```

Or, for a partitioned system with partition `pN`:

```
smw:~# xtbootsys --partition pN -s last -a auto.xtshutdown
```

This method shuts down the compute nodes (which are commonly also Lustre clients), then executes `xtshutdown` on service nodes, halting the nodes and then stopping processes on the SMW. A system administrator can shut down the system using both user-defined and built-in procedures in the `auto.xtshutdown` file, which is located on the SMW in the `/opt/cray/hss/default/etc` directory.

For related procedures, see *XC™ Series Software Initial Installation and Configuration Guide*. For more information about using automation files, see the `xtbootsys(8)` man page.

The xtshutdown Command

The `xtshutdown` command executes a series of commands locally on the boot node and service nodes to shut down the system in an orderly fashion. The sequence of shutdown steps and the nodes on which to execute them are defined by the system administrator in the `/etc/opt/cray/init-service/xtshutdown.conf` file or in the file specified by the environment variable `XTSHUTDOWN_CONF`.

Root user privileges are required to run `xtshutdown`. Passwordless `ssh` must be enabled for the root user from the boot node to all service nodes.

The `xtshutdown` command uses `pdsh` to invoke commands on the selected service nodes (i.e., boot node, SDB node, a class of nodes, or a single host). A system administrator can define functions to execute when the system is shut down. Place these functions in the `/etc/opt/cray/init-service/xt_shutdown_local` file or the file defined by the `XTSHUTDOWN_LOCAL` environment variable.

Shut Down Service Nodes

Prerequisites

`Root` user privileges are required to run `xtshutdown`. Passwordless `ssh` must be enabled for the `root` user from the boot node to all service nodes.

CAUTION: The `xtshutdown` command does not shut down compute nodes. To shut down the compute and service nodes, see *Shut Down the System or Part of the System Using the xtcli shutdown Command*.

About this task

For information about shutting down service nodes, see the `xtshutdown(8)` man page.

Procedure

1. Modify the `/etc/opt/cray/init-service/xtshutdown.conf` file or the file specified by the `XTSHUTDOWN_CONF` environment variable to define the sequence of shutdown steps and the nodes on which to execute them. The `/etc/opt/cray/init-service/xtshutdown.conf` file resides on the boot node.

2. If desired, define functions to execute when the system is shut down. Place these functions in the `/etc/opt/cray/init-service/xt_shutdown_local` file or the file defined by the `XTSHUTDOWN_LOCAL` environment variable.

3. Execute `xtshutdown`.

   ```
   boot:~ # xtshutdown
   ```

   After the software on the nodes is shutdown, the administrator can halt the hardware, reboot, or power down.

Shut Down the System or Part of the System Using the xtcli shutdown Command

The HSS `xtcli shutdown` command shuts down the system or a part of the system. To shut down compute nodes, execute the `xtcli shutdown` command. Under normal circumstances, for example to successfully disconnect from Lustre, invoking the `xtcli shutdown` command attempts to gracefully shut down the specified nodes.

For information, see the `xtcli(8)` man page.

Shut down all compute nodes

```
crayadm@smw:~> xtcli shutdown compute
```

Shut down specified compute nodes

For this example, shut down only compute nodes in cabinet `c13-2`:

```
crayadm@smw:~> xtcli shutdown c13-2
```

Shut down all nodes of a system

```
crayadm@smw:~> xtcli shutdown s0
```

Shut down a partition *pN* of a system

```
crayadm@smw:~> xtcli shutdown pN
```

Force nodes to shut down (immediate halt)

When all nodes of a system must be halted immediately, use the `-f` argument; nodes will not go through their normal shutdown process. Forced shutdown occurs even if the nodes have an alert status present.

```
crayadm@smw:~> xtcli shutdown -f s0
```

After the software on the nodes is shutdown, the system administrator can halt the hardware, reboot, or power down.

Stop System Components

When a system administrator removes, stops, or powers down components, any applications and compute processes that are running on those components are lost.

Reserve a Component

To allow applications and compute processes to complete before stopping components, use the HSS `xtcli set_reserve` *`idlist`* command to prevent the selected nodes from accepting new jobs.

A node running CNL and using ALPS is considered to be down by ALPS after it is reserved using the `xtcli set_reserve` command. The output from `apstat` will show the node as down (DN), even though there may be an application running on that node. This DN designation indicates that no other work will be placed on the node after the currently running application has terminated.

For more information, see the `xtcli_set(8)` man page.

Reserve a component

```
crayadm@smw:~> xtcli set_reserve idlist
```

Power Down Blades or Cabinets

WARNING: Power down the cabinets with software commands. Tripping the circuit breakers may result in damage to system components.

WARNING: Before powering down a blade or a cabinet, ensure the operating system is not running.

The `xtcli power down` command powers down the specified cabinet and/or blades within the specified partition, chassis or list of blades. Cabinets must be in the `READY` state to receive power commands. See *System Component States*. The `xtcli power down` command has the following form, where *`physIDlist`* is a comma-separated list of cabinets or blades present on the system.

```
xtcli power down physIDlist
```

The `xtcli power force_down` and `xtcli power down_slot` commands are aliases for the `xtcli power down` command. For information about disabling and enabling components, see *Disable Hardware Components*, and *Enable Hardware Components*, respectively.

WARNING: Although a blade is powered off, the HSS in the cabinet is live and has power.

For information about powering down a component, see the `xtcli_power(8)` man page.

Power down a specified blade

For this example, power down a blade with the ID `c0-0c0s7`:

```
crayadm@smw:~> xtcli power down c0-0c0s7
```

Halt Selected Nodes

Use the HSS `xtcli halt` command to halt selected nodes. For more information, see the `xtcli(8)` man page.

Halt a node

For this example, halt node `157`:

```
crayadm@smw:~> xtcli halt 157
```

Restart a Blade or Cabinet

IMPORTANT: Change the state of the hardware only when the operating system is not running or is shut down.

The `xtcli power up` command powers up the specified cabinet and/or blades within the specified partition, chassis or list of blades. Cabinets must be in the `READY` state (see *System Component States*) to receive power commands. The `xtcli power up` command does not attempt to power up network mezzanine cards or nodes that are handled by the `xtbounce` command during system boot.

The `xtcli power up_slot` command is an alias for the `xtcli power up` command.

The `xtcli power up` command has the following form, where *`physIDlist`* is a comma-separated list of cabinets or blades present on the system.

```
xtcli power up physIDlist
```

For more information, see the `xtcli_power(8)` man page.

Power up blades in `c0-0c0s7`

```
crayadm@smw:~> xtcli power up c0-0c0s7
```

Abort Active Sessions on the HSS Boot Manager

About this task

Use the HSS `xtcli session abort` command to abort sessions in the boot manager. A session corresponds to executing a specific command such as `xtcli power up` or `xtcli boot`.

For more information about manager sessions, see the `xtcli(8)` man page.

Procedure

1. Display all running sessions in the boot manager. Only the boot manager supports multiple simultaneous sessions.

   ```
   crayadm@smw:~> session show BM all
   ```

2. Abort the selected session, *session_id*.

   ```
   crayadm@smw:~>  xtcli session abort BM session_id
   ```

Display and Change Software System Status

The user command `xtnodestat` provides a display of the status of nodes: how they are allocated and to what jobs. The `xtnodestat` command provides current job and node status summary information, and it provides an interface to ALPS and jobs running on CNL compute nodes. ALPS must be running in order for `xtnodestat` to report job information.

For more information, see the `xtnodestat(1)` man page.

View and Change the Status of Nodes

Use the `xtprocadmin` command on a service node to view the status of components of a booted system in the `processor` table of the SDB. The command enables the system administrator to retrieve or set the processing mode (`interactive` or `batch`) of specified nodes. The administrator can display the state (`up`, `down`, `admindown`, `route`, or `unavailable`) of the selected components, if needed. The administrator can also allocate processor slots or set nodes to become unavailable at a particular time. The node is scheduled only if the status is `up`.

When the `xtprocadmin -ks` option is used, then the option can either a normal argument (`up`, `down`, etc.), or it can have a colon in it to represent a conditional option; for example, the option of the form `up:down` means "if state was up, mark down".

For more information, see the `xtprocadmin(8)` man page.

View node characteristics

```
login:~> xtprocadmin
   NID    (HEX)    NODENAME     TYPE    STATUS        MODE
     1      0x1  c0-0c0s0n1  service        up       batch
     2      0x2  c0-0c0s0n2  service        up       batch
     5      0x5  c0-0c0s1n1  service        up       batch
     6      0x6  c0-0c0s1n2  service        up       batch
```

```
      8       0x8   c0-0c0s2n0   compute          up        batch
      9       0x9   c0-0c0s2n1   compute          up        batch
     10       0xa   c0-0c0s2n2   compute          up        batch
     11       0xb   c0-0c0s2n3   compute          up        batch
```

View all node attributes

```
login:~> xtprocadmin -A
   NID    (HEX)    NODENAME     TYPE ARCH        OS CPUS  CU AVAILMEM   PAGESZ CLOCKMHZ GPU SOCKETS DIES C/
CU                      LABEL0                            LABEL1
LABEL2                     LABEL3
     1      0x1  c0-0c0s0n1  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     2      0x2  c0-0c0s0n2  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     5      0x5  c0-0c0s1n1  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     6      0x6  c0-0c0s1n2  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     8      0x8  c0-0c0s2n0  compute   xt       CNL   32  16    65536     4096     2600   0       2    2
2
     9      0x9  c0-0c0s2n1  compute   xt       CNL   32  16    65536     4096     2600   0       2    2
2
    10      0xa  c0-0c0s2n2  compute   xt       CNL   32  16    65536     4096     2600   0       2    2
2
```

View selected attributes of selected nodes

For this example, the `-a` option lists the selected attributes to display:

```
login:~> xtprocadmin -n 8 -a arch,clockmhz,os,cores
   NID    (HEX)    NODENAME     TYPE ARCH CLOCKMHZ        OS CPUS
     8       0x8   c0-0c0s2n0   compute   xt     2600       CNL   32
```

Disable a node

For this example, the `admindown` option disables node `c0-0c0s3n1` such that it cannot be allocated:

```
crayadm@nid00004:~> xtprocadmin -n c0-0c0s3n1 -k s admindown
```

Disable all processors

```
crayadm@nid00004:~> xtprocadmin -k s admindown
```

Mark a Compute Node as a Service Node

Use the `xtcli mark_node` command to mark a node in a compute blade to have a role of `service` or `compute`; `compute` is the default. It is not permitted to change the role of a node on a service blade, which always has the `service` role.

Marking a node on a compute blade as `service` or `compute` allows the administrator to load the desired boot image at boot time. Compute nodes marked as `service` can run software-based services. A request to change the role of a running node (that is, the node is in the `ready` state and the operating system is running) will be denied.

For more information, see the `xtcli(8)` man page and *Check the Status of System Components* on page 143.

Find Node Information

Translate Between Physical ID Names and Integer NIDs

To translate between physical ID names (cnames) and integer NIDs, generate a system map on the System Management Workstation (SMW) and filter the output, enter the following command:

```
crayadm@smw:~> rtr --system-map | grep cname | awk '{ print $1 }'
```

To translate between physical ID names (rnames) and integer NIDs, generate a system map on the System Management Workstation (SMW) and filter the output, enter the following command:

```
crayadm@smw:~> rtr --system-map | grep rname | awk '{ print $1 }'
```

For more information, see the `rtr(8)` man page.

Find Node Information Using the `xtnid2str` Command

The `xtnid2str` command converts numeric node identification values to their physical names (cnames). This allows conversion of Node ID values, ASIC NIC address values, or ASIC ID values.

For additional information, see the `xtnid2str(8)` man page.

Find the physical ID for node 38

```
smw:~> xtnid2str 28
node id 0x26 = 'c0-0c0s1n2'
```

Find the physical ID for nodes 0, 1, 2, and 3

```
smw:~> xtnid2str 0 1 2 3
node id 0x0 = 'c0-0c0s0n0'
node id 0x1 = 'c0-0c0s0n1'
node id 0x2 = 'c0-0c0s1n0'
node id 0x3 = 'c0-0c0s1n1'
```

Find the physical IDs for Aries IDs 0-7

```
smw:~> xtnid2str -a 0-7
aries id 0x0 = 'c0-0c0s0a0'
aries id 0x1 = 'c0-0c0s1a0'
aries id 0x2 = 'c0-0c0s2a0'
aries id 0x3 = 'c0-0c0s3a0'
aries id 0x4 = 'c0-0c0s4a0'
aries id 0x5 = 'c0-0c0s5a0'
aries id 0x6 = 'c0-0c0s6a0'
aries id 0x7 = 'c0-0c0s7a0'
```

Find Node Information Using the `nid2nic` Command

The `nid2nic` command prints the *nid*-to-*nic* address mappings, *nic*-to-*nid* address mappings, and a specific *physical_location*-to-*nic* address and *nid* mappings.

For information about using the `nid2nic` command, see the `nid2nic(8)` man page.

Print the *nid*-to-*nic* address mappings for the node with NID 31

```
smw:~> nid2nic 31
NID:0x1f         NIC:0x21         c0-0c0s7n3
```

Print the *nid*-to-*nic* address mappings for the node with NID 31, but specify the NIC value in the command line

```
smw:~> nid2nic -n 0x21
NIC:0x21         NID:0x1f         c0-0c0s7n3
```

Display and Change Hardware System Status

A system administrator can execute commands that look at and change the status of the hardware.

CAUTION: Execute commands that change the status of hardware only when the operating system is shut down.

Generate HSS Physical IDs

The HSS `xtgenid` command generates HSS physical IDs, for example, to create a list of blade controller identifiers for input to the flash manager. Selection can be restricted to components of a particular type. Only user `root` can execute the `xtgenid` command.

For more information, see the `xtgenid(8)` man page.

Create a list of node identifiers that are not in the DISABLE, EMPTY, or OFF state

```
smw:~ # xtgenid -t node --strict
```

Disable Hardware Components

If links, nodes, or Cray ASICs have hardware problems, the system administrator can direct the system to ignore the components with the `xtcli disable` command.

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

The `xtcli disable` command has the following form, where *`idlist`* is a comma-separated list of components (in cname format) that the system is to ignore. The system disregards these links or nodes.

```
xtcli disable [{-t type [-a] } | -n] [-f] idlist
```

IMPORTANT: The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

Disabling of a cabinet, chassis, or blade will fail if any nodes under the component are in the `ready` state, unless the force option (`-f`) is used. An error message will indicate the reason for the failure.

Disabling of a node in the `ready` state will fail, unless the force option (`-f`) is used. An error message will indicate the reason for the failure.

The state of `empty` components will not change when using the `disable` command, unless the force option (`-f`) is used.

For detailed information about using the `xtcli disable` command, see the `xtcli(8)` man page.

Disable the Aries ASIC `c0-0c1s3a0`

1. Determine that the ASIC is in the `OFF` state.

   ```
   crayadm@smw:~> xtcli status -t aries c0-0c1s3a0
   ```

2. If the ASIC is not in the `OFF` state, power down the blade that contains the ASIC.

   ```
   crayadm@smw:~> xtcli power down c0-0c1s3
   ```

3. Disable the ASIC.

   ```
   crayadm@smw:~> xtcli disable c0-0c1s3a0
   ```

4. Power up the blade that contains the ASIC.

   ```
   crayadm@smw:~> xtcli power up c0-0c1s3
   ```

Enable Hardware Components

If links, nodes, or Cray ASICs that have been disabled are later fixed, the system administrator can add them back to the system with the `xtcli enable` command.

The `xtcli enable` command has the following form, where *`idlist`* is a comma-separated list of components (in cname format) for the system to recognize.

```
xtcli enable [{-t type [-a] } | -n] [-f] idlist
```

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

> **IMPORTANT:** The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

The state of `empty` components does not change when using the `xtcli enable` command, unless the force option (`-f`) is used.

The state of `off` means that a component is present on the system. If the component is a blade controller, node, or ASIC, then this will also mean that the component is powered off. If the administrator disables a component, the state shown becomes `disabled`. When the `xtcli enable` command is used to enable that component for use once again, its state switches from `disabled` to `off`. In the same manner, enabling an empty component means that its state switches from `empty` to `off`.

For more information, see the `xtcli(8)` man page.

Set Hardware Components to `EMPTY`

Use the `xtcli set_empty` command to set a selected component to the `EMPTY` state. HSS managers and the `xtcli` command ignore empty or disabled components.

Setting a selected component to the `EMPTY` state is typically done when a component, usually a blade, is physically removed. By setting it to `EMPTY`, the system ignores it and routes around it.

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

> **IMPORTANT:** The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

For more information, see the `xtcli(8)` man page.

Set a blade to the `EMPTY` state

```
crayadm@smw:~> xtcli set_empty -a c0-0c1s7
```

Lock Hardware Components

Components are automatically locked when a command that can change their state is running. As the command is started, the state manager locks these components so that nothing else can affect their state while the command executes. When the manager is finished with the command, it unlocks the components.

Use the HSS `xtcli lock` command to lock components. Locking a component prints out the state manager session ID.

For more information, see the `xtcli(8)` man page.

Lock cabinet `c0-0`

```
crayadm@smw:~> xtcli lock -l c0-0
```

Show all session (lock) data

```
crayadm@smw:~> xtcli lock show
```

Unlock Hardware Components

Use the HSS `xtcli lock` command to unlock components. This command is useful when an HSS manager fails to unlock some set of components.

The system administrator can manually check for locks with the `xtcli lock show` command and then unlock them. Unlocking a component does not print out the state manager session ID. The `-u` option must be used to unlock a component as follows:

```
crayadm@smw:~> xtcli lock -u lock_number
```

Where *`lock_number`* is the value given when initiating the lock; it is also indicated in the `xtcli lock show` query. Unlocking does nothing to the state of the component other than to release locks associated with it.

HSS daemons cannot affect components that are locked by a different session.

Set the Turbo Boost Limit

The Intel® Xeon Phi™ processors do not support turbo boost limiting.

Because Intel Ivy Bridge and Haswell processors have a high degree of variability in the amount of turbo boost each processor can supply, limiting the amount of turbo boost can reduce performance variability and reduce power consumption. The limit applies only when a high number of cores are active. On an N-core processor, the limit is in effect when the active core count is N, N-1, N-2, or N-3. On a 12-core processor, the limit is in effect when 12, 11, 10, or 9 cores are active. Set the turbo_boost_limit parameter to `100`, `200`, or `999`. The default setting is `999`. When `100` is specified, 100 MHz is the limit. A value of `200` limits turbo boost to 200 MHz. A value of `999` implies no turbo boost limit is applied.

Set the turbo boost limit using one of the following methods:

- Add the following action to a boot automation file:

  ```
  lappend actions { crms_boot_loadfile CNL0 compute $data(idlist)
  turbo_boost_limit=value }
  ```

- Warm boot the compute nodes using this command:

  ```
  xtbootsys --reboot -L CNL0 --compute-boot-params turbo_boost_limit=value
  idlist
  ```

  where *`idlist`* is a comma-separated list of compute nodes (in cname format) to be booted. **This configuration change is not persisted.**

Perform Parallel Operations on Service Nodes

Use `pdsh`, the CLE parallel remote shell utility for service nodes, to issue commands to groups of nodes in parallel. The system administrator can select the nodes on which to use the command, exclude nodes from the command, and limit the time the command is allowed to execute. Only user `root` can execute the `pdsh` command. The command has the following form:

```
pdsh [options] command
```

For more information, see the `pdsh(1)` man page.

Restart the NTP service

```
boot:~ # pdsh -w 'login[1-9]' /etc/init.d/ntp restart
```

Perform Parallel Operations on Compute Nodes

The parallel command tool (`pcmd`) facilitates execution of the same commands on groups of compute nodes in parallel, similar to `pdsh`. Although `pcmd` is launched from a service node, it acts on compute nodes. It allows administrators and/or, if the site deems it feasible, other users to securely execute programs in parallel on compute nodes. The user can specify on which nodes to execute the command. Alternatively, the user can specify an application ID (*`apid`*) to execute the command on all the nodes available under that *`apid`*.

An unprivileged user must execute the command targeting nodes where the user is currently running an `aprun`. A `root` user is allowed to target any compute node, regardless of whether there are jobs running there or not. In either case, if the `aprun` exits and the associated applications are killed, any commands launched by `pcmd` will also exit.

By default, `pcmd` is installed as a `root`-only tool. It must be installed as `setuid root` in order for unprivileged users to use it.

The `pcmd` command is located in the `nodehealth` module. If the `nodehealth` module is not part of the default profile, load it by specifying:

```
module load nodehealth
```

For additional information, see the `pcmd(1)` man page.

xtbounce Error Message Indicates Cabinet Controller and Its Blade Controllers Not in Sync

During the `gather_cab_pwr_states` phase of `xtbounce`, if the HSS software on a cabinet controller and any of its blade controllers is out of sync, error messages such as the following will be printed during the `xtbounce`.

```
***** gather_cab_pwr_states *****
18:28:42 - Beginning to wait for response(s)

ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
```

If this occurs, it indicates that the blade controller software is at a different revision than the cabinet controller software. `xtbounce` will print a list of cabinets for which this error has occurred. The message will be similar to the following:

```
ERROR: power state check error on 2 cabinet(s)
WARNING: unable to find c0-0 in err_cablist
WARNING: unable to find c0-2 in err_cablist
```

This error is an indication that when the HSS software was previously updated, the cabinet controllers and the blade controllers were not updated to the same version.

To correct this error, cancel out of `xtbounce` (with **Ctrl-C**), wait approximately five minutes for the `xtbounce` related activities on the blade controllers to finish, then reboot the cabinet controller(s) and their associated blade controllers to get the HSS software synchronized. Following this, the `xtbounce` may be executed once again.

Reduce Impact to SMW Performance of Btrfs Periodic Maintenance

About this task

Btrfs (B-tree file system) runs periodic maintenance. The weekly and monthly maintenance scripts, which include balance, trim, and scrub actions, can consume large amounts of compute resource. This can impact a site's ability to use the SMW for normal operations, even using SSH to log into nodes. This procedure explains how to reduce impact to the SMW by controlling when these scripts are run.

Procedure

1. Create a file `/etc/cron.d/cray_btrfs.cron`. Set ownership to root,root with permissions 644.

   The new cron file needs to be in `/etc/cron.d` because the Btrfs RPM installs links to maintenance scripts into the `/etc/cron.{weekly,monthly}` directories.

2. Add these lines to the new file. Adjust as needed for this site.

```
# Control when btrfs maintenance scripts run by deleting the corresponding
# 'lastrun' files at a predetermined time.  Caveat, this affects all of the
# scripts in the corresponding cron directories (/etc/cron.{weekly,monthy})

# Run weekly on Saturday at 2 AM as root
0 2 * * 6 root rm -f /var/spool/cron/lastrun/cron.weekly
# Run monthly on the first Sunday of the month at 2 AM as root
0 2 * * 0 root [ $(date +%d) -le 07 ] && rm -f /var/spool/cron/lastrun/
cron.monthly
```

Power-cycle a Component to Handle Bus Errors

About this task

Bus errors are caused by machine-check exceptions. If a bus error occurs, try power-cycling the component.

Procedure

1. Power down the components. The *physIDlist* is a comma-separated list of components present on the system.

```
crayadm@smw:~> xtcli power down physIDlist
```

2. Power up the components.

```
crayadm@smw:~> xtcli power up physIDlist
```

When a Component Fails

Components that fail are replaced as field replaceable units (FRUs). FRUs include compute blade components, service blade components, and power and cooling components.

When a field replaceable unit (FRU) problem arises, contact a Customer Service Representative to schedule a repair.

Capture and Analyze System-level and Node-level Dumps

The `xtdumpsys` command collects and analyzes information from a Cray system that is failing or has failed, has crashed, or is hung. Analysis is performed on, for example, event log data, active heartbeat probing, voltages, temperatures, health faults, in-memory console buffers, and high-speed interconnection network errors. When failed components are found, detailed information is gathered from them.

To collect similar information for components that have not failed, invoke the `xtdumpsys` command with the `--add` option and name the components from which to collect data. The HSS `xtdumpsys` command saves dump information in `/var/opt/cray/dump/timestamp` by default.

> **NOTE:** When using the `--add` option to add multiple components, separate components with spaces, not commas.

Dump information about a working component

For this example, dump the entire system and collect detailed information from all blade controllers in chassis `0` of cabinet `0`:

```
crayadm@smw:~> xtdumpsys --add c0-0c0s0
```

The `xtdumpsys` command is written in Python and supports plug-ins written in Python. A number of plug-in scripts are included in the software release. Call `xtdumpsys --list` to view a list of included plug-ins and their respective directories. The `xtdumpsys` command also now supports the use of configuration files to specify `xtdumpsys` presets, rather than entering them via the command line.

For more information, see the `xtdumpsys(8)` man page.

cdump and crash Utilities for Node Memory Dump and Analysis

The `cdump` and `crash` utilities may be used to analyze the memory on any Cray service node or CNL compute node. The `cdump` command is used to dump node memory to a file. After `cdump` completes, the `crash` utility can be used on the dump file generated by `cdump`.

Cray recommends executing the `cdump` utility only if a node has panicked or is hung, or if a dump is requested by Cray.

To select the desired access method for reading node memory, use the `cdump -r access` option. Valid access methods are:

xt-bhs The `xt-bhs` method uses a basic hardware system server that runs on the SMW to access and read node memory. `xt-bhs` is the default access method for these systems.

xt-hsn The `xt-hsn` method utilizes a proxy that reads node memory through the High-speed Network (HSN). The `xt-hsn` method is faster than the `xt-bhs` method, but there are situations where it will not work (for example, if the ASIC is not functional). However, the `xt-hsn` method is preferable because the dump completes in a short amount of time and the node can be returned to service sooner.

xt-file The `xt-file` method is used for memory dump file created by the `-z` option. The compressed memory dump file must be uncompressed prior to executing this command. Use the file name for *node-id*.

xc-knc The `xc-knc` method is used to dump Intel Xeon Phi nodes. Use this method when dumping only the Xeon Phi coprocessor without dumping the host node. When dumping the host node, do not use `xc-knc`. A host node dump automatically includes dumping the Xeon Phi coprocessors unless they are suppressed by specifying the `-n` option.

To dump Cray node memory, *access* takes the following form:

```
method[@host]
```

For additional information, see the `cdump(8)` and `crash(8)` man pages.

Dump and Reboot Nodes Automatically

The SMW daemon `dumpd` initiates automatic dump and reboot of nodes when requested by the Node Health Checker (NHC).

CAUTION: The `dumpd` daemon is invoked automatically by `xtbootsys` on system (or partition) boot. In most cases, system administrators do not need to use this daemon directly.

A system administrator can set global variables in the `/etc/opt/cray/nodehealth/nodehealth.conf` configuration file to control the interaction of NHC and `dumpd`. For more information about NHC and the `nodehealth.conf` configuration file, see *Configure the Node Health Checker (NHC)*.

Variables can also be set in the `/etc/opt/cray-xt-dumpd/dumpd.conf` configuration file on the SMW to control how `dumpd` behaves on the system.

Each CLE release package also includes an example `dumpd` configuration file, `/etc/opt/cray-xt-dumpd/dumpd.conf.example`. The `dumpd.conf.example` file is a copy of the `/etc/opt/cray-xt-dumpd/dumpd.conf` file provided for an initial installation.

> **IMPORTANT:** The `/etc/opt/cray-xt-dumpd/dumpd.conf` file is not overwritten during a CLE upgrade if the file already exists. This preserves the site-specific modifications previously made to the file. Cray recommends comparing the site's `/etc/opt/cray-xt-dumpd/dumpd.conf` file content with the `/etc/opt/cray-xt-dumpd/dumpd.conf.example` file provided with each release to identify any changes and then update the site's `/etc/opt/cray-xt-dumpd/dumpd.conf` file accordingly.
>
> If the `/etc/opt/cray-xt-dumpd/dumpd.conf` file does not exist, then the `/etc/opt/cray-xt-dumpd/dumpd.conf.example` file is copied to the `/etc/opt/cray-xt-dumpd/dumpd.conf` file.

The CLE installation and upgrade processes automatically install `dumpd` software, but it must be explicitly enabled.

The /etc/opt/cray-xt-dumpd/dumpd.conf Configuration File

The `dumpd` configuration file, `/etc/opt/cray-xt-dumpd/dumpd.conf`, is located on the SMW. There is no need to change any installation configuration parameters, but a system administrator can edit the `/etc/opt/cray-xt-dumpd/dumpd.conf` file to customize how `dumpd` behaves on the system using the following configuration variables.

`enable: yes|no`

Provides a quick on/off switch for all `dumpd` functionality.

Default is `no`.

`partitions: number`

Specifies whether or not `dumpd` acts on specific partitions or ranges of partitions. Placing `!` in front of a partition or range disables it.

For example, specifying

```
partitions: 1-10,!2-4
```

enables partitions 1, 5, 6, 7, 8, 9, and 10 but not 2, 3, or 4. Partitions must be explicitly enabled. Leaving this option blank disables all partitions.

`disabled_action: ignore|queue`

Specifies what to do when requests come in for a disabled partition. If `ignore` is specified, requests are removed from the database and not acted upon. If `queue` is

specified, requests continue to build while `dumpd` is disabled on a partition. When the partition is reenabled, the requests will be acted on. Specifying `queue` is not recommended if `dumpd` will be disabled for long periods of time, as it can cause SMW stress and database problems.

save_output: always|errors|never

Indicates when to save `stdout` and `stderr` from `dumpd` commands that are executed. If `save_output` is set to `always`, all output is saved. If `errors` is specified, output is saved only when the command exits with a nonzero exit code. If `never` is specified, output is never saved.

The default is to save output on errors.

command_output: *directory*

Specifies where to save output of `dumpd` commands, per the `save_output` variable. The command output is put in the file *`action.pid.timestamp`*`.out` in the directory specified by this option.

Default directory is `/var/opt/cray/dump`.

dump_dir: *directory*

Specifies the directory in which to save dumps.

Default directory is `/var/opt/cray/dump`.

max_disk: *nnnMB*|unlimited

Specifies the amount of disk space beyond which no new dumps will be created. This is not a hard limit; if `dumpd` sees that this directory has less than this amount of space, it starts a new dump, even if that dump subsequently uses enough space to exceed the `max_disk` limit.

The default value is `max_disk: unlimited`.

no_space_action: *action*

Specifies a command to be executed if the directory specified by the variable `dump_dir` does not have enough space free, as specified by `max_disk`. For example:

Deletes the oldest dump in the dump directory:

```
no_space_action: rm -rf $dump_dir/$(ls -rt $dump_dir | head
-1)
```

Moves the oldest dump somewhere useful:

```
no_space_action: mv $dump_dir/$(ls -t $dump_dir|head
-1) /some/dump/archive
```

Sends E-mail to an administrator at *admin@fictionalcraysite.com*:

```
no_space_action: echo "" | mail -s "Not Enough Space in
$dump_dir" \
admin@fictionalcraysite.com
```

The dumpd-dbadmin Tool

The `dumpd` daemon sits and waits for requests from NHC (or some other entity using the `dumpd-request` tool on the shared root.) When `dumpd` gets a request, it creates a database entry in the `mznhc` database for the

request, and calls the script `/opt/cray-xt-dumpd/default/bin/executor` to read the `dumpd.conf` configuration file and perform the requested actions.

Use the `dumpd-dbadmin` tool to view or delete entries in the `mznhc` database in a convenient manner.

The dumpd-request Tool

Use the `dumpd-request` tool to send dump and reboot requests to `dumpd` from the SMW or the shared root. A request includes a comma-separated list of actions to perform, and the node or nodes on which to perform the actions.

A typical request from NHC looks like this:

```
cname: c0-0c1s4n0 actions: halt,dump,reboot
```

A system administrator can define additional actions in the `dumpd.conf` configuration file. To use, execute the `dumpd-request` tool located on the shared root or the SMW. A typical call would be:

```
dumpd-request -a halt,dump,reboot -c c0-0c1s4n0
```

Or

```
dumpd-request -a myaction1,myaction2 -c
c1-0c0s0n0,c1-0c0s0n1,c1-0c0s0n2,c1-0c0s0n3
```

For this example to work, `myaction1` and `myaction2` must be defined in the `dumpd.conf` file. See the examples in the configuration file for more detail.

Collect Debug Information From Hung Nodes Using the xtnmi Command

CAUTION: This is not a harmless tool to use to repeatedly get information from a node at various times; only use this command when debugging data from nodes that are in trouble is needed. The `xtnmi` command output may be used to determine problems such as a core hang. `xtnmi` will stop a running node. It is best used when a node is not running correctly and debugging information is needed, or to stop a node that is running incorrectly.

The sole purpose of the `xtnmi` command is to collect debug information from unresponsive nodes. As soon as that debug information is displayed to the console, the node panics.

For additional information, see the `xtnmi(8)` man page.

Modify BIOS Parameters

There are a few, rare circumstances where it may be necessary to modify BIOS parameters, for example, in order to troubleshoot a problem, or if there is a need to test a new BIOS version on a small set of nodes before implementing the change across an entire system.

The `xtbiosconf` command allows administrators to specify BIOS parameters at the node, blade, chassis, or cabinet level. BIOS parameters can be associated with a BIOS revision, numeric parameter offset or parameter name, and target nodes. BIOS revision wildcards are supported. The BIOS parameter data is saved in a database

on the SMW, and made available automatically to blade controllers via the ERFS file system. In most cases a cold reboot of the affected nodes is needed to apply the new settings.

CAUTION: *Do not* attempt to use this command except under guidance by Cray support personnel, who will provide all the steps for shutting down the nodes, changing the settings, and bringing the nodes back up. Improper use of this command can damage a system.

The following command displays the current BIOS Parameter settings for the entire system:

```
smw~> xtbiosconf --show s0
==============|======|==================================
              | BIOS | BIOS
Node          | REV  | Parameter
==============|======|==================================
c0-1c0s0n1    | 4030 | numlock=1
c0-1c0s0n1    | 4030 | acpiauto=0
==============|======|==================================
c0-1c0s0n2    | 4030 | numlock=1
c0-1c0s0n2    | 4030 | acpiauto=0
==============|======|==================================
```

For more information see the `xtbiosconf` man page.

Set or Change the HSS Data Store (MariaDB) Root Password

About this task

The method for setting or changing the HSS data store (database) root password has changed with the release of CLE 6.0.

Old The HSS database was implemented with MySQL. After initial installation, its root password was changed from the initial default empty string to a user-defined value by the `SMWconfig` script, which was run after `SMWinstall` and the initial discovery of the system.

New The HSS database is now implemented with MariaDB, a MySQL work-alike database with identically named commands. As before, the initial default root password is the empty string; however, the `SMWconfig` script is no longer used to set it after installation. The administrator must use the following procedure to set the root password to a user-defined value.

Once the MariaDB root password has been set, it must be placed in `/root/.my.cnf`, a file readable only by root that has the format shown in step 2. This file serves as the mechanism by which the installer and the `snaputil` command obtain the root password when they access MariaDB as root. If the file does not exist or it has no `password=` line, the system will attempt to access MariaDB using the default empty-string password, which will fail once the password has been changed. The first time the root password is set to a user-defined value, `/root/.my.cnf` must be created. Afterwards, whenever the MariaDB root password is changed, `/root/.my.cnf` must be updated to match.

IMPORTANT: For an SMW HA system, record the new MySQL root password. It will need to be changed on the second SMW later (by editing `/root/.my.cnf`). After the SMW HA cluster has been configured, the MySQL root password needs to be reset with `mysqladmin` on only one SMW, because the MySQL database is shared between both SMWs in the HA cluster.

Procedure

1. Set or change the MariaDB root password.

   ```
   smw# mysqladmin -uroot password -p
   ```

   Do one of the following at the prompt:

   - To **set** the root password for fresh installs or after the database has been reinitialized, press **Enter** to enter an empty string, the default initial password.

     ```
     Enter password: <cr>
     ```

   - To **change** the root password, enter the existing password.

     ```
     Enter password: existing_password
     ```

   At these prompts, enter the new root password, and then enter it again.

   ```
   New password:
   Confirm new password:
   ```

2. Ensure that the root password in the `/root/.my.cnf` file matches the new root password.

   If this file does not yet exist, create it and add the lines shown in the example, substituting the new password for the placeholder *MariaDB-password*.

   ```
   smw# vi /root/.my.cnf
   [client]
   user=root
   password=MariaDB-password
   ```

   Ensure that only root can see or write to this file.

   ```
   smw# chmod 600 /root/.my.cnf
   ```

Recover from a Corrupt or Missing SMW MariaDB Database

About this task

If the HSS MariaDB (formerly MySQL) database has been damaged or is missing, there are three possible courses of action:

- Repair.

  If the database has become corrupt, MariaDB automatically attempts to repair damaged tables. Look in the log file (default `/var/lib/mysql/machine.err`) for suggested manual recovery steps, if any, and try those first. Repairing the database is the best option when possible.

- Restore and regenerate.

  If there are no suggestions or the suggested steps fail to repair the database, use the procedure *Restore the HSS MariaDB Database from a Backup* on page 88. Restoring the database from the most recent backup (provided a more recent manual backup is not available) will restore the database to its state just prior to the last `xtdiscover` or `warmswap add` operation. An incremental discovery to the present system state will usually be faster than one made from a fresh database, and it will not require administrative state changes made prior to the backup (such as marking compute nodes as 'service') to be performed again.

TIP: To minimize needed discovery, make more frequent backups:

```
/usr/bin/mysqldump --add-drop-database --routines -uhssds -phssds hssds
 > /home/crayadm/hss_db_backup/my-new-hssds-backup.sql
```

The HSS MariaDB database could be backed up after every successful warmswap (**xtdiscover --warmswap**), regular **xtdiscover**, and any administrative state change (e.g., **xtcli disable/enable/set_empty/mark_node**). Because these actions are all logged in the commands log, they could be used to automatically trigger backups.

- Regenerate from scratch.

  If all else fails, use the procedure *Regenerate the HSS MariaDB Database from Scratch* on page 89. In this case, the database and the database root password are wiped out, and discovery is used to regenerate the database.

Restore the HSS MariaDB Database from a Backup

About this task

If the HSS MariaDB database becomes corrupt or is missing, and automated attempts to repair damaged tables have failed, use this procedure to do a partial restoration from backup.

Procedure

1. Stop the HSS daemons (by stopping RSMS) and the MariaDB service.

   ```
   crayadm@smw> sudo /usr/bin/systemctl stop rsms.service
   crayadm@smw> sudo /usr/bin/systemctl stop mysql.service
   ```

2. Move the damaged database files out of the database directory.

   ```
   crayadm@smw> mkdir /tmp/backup12
   crayadm@smw> cd /var/lib/mysql
   crayadm@smw> sudo mv ibdata1 ib_logfile0 ib_logfile1 hssds /tmp/backup12
   ```

   This procedure assumes that the old database files cannot be repaired; however, this step retains those old database files (just in case) and clears out the database directory.

3. Restart MariaDB.

   ```
   crayadm@smw> sudo /usr/bin/systemctl start mysql.service
   ```

4. Ensure the database is gone.

   ```
   crayadm@smw> mysql -uhssds -phssds -e "drop database hssds"
   ```

   If the database is gone, the following error message appears:

   ```
   ERROR 1008 (HY000) at line 1: Can't drop database 'hssds'; database doesn't
   exist
   ```

5. Load the most recent MariaDB backup (from `/home/crayadm/hss_db_backup/`).

   ```
   crayadm@smw> mysql -uhssds -phssds < db_backup.11-17-2014.1120.sql
   ```

The backups in `/home/crayadm/hss_db_backup/` are from past runs of `xtdiscover` and `xtwarmswap --add` and were taken *before* the state of the database was updated.

6. Restart the HSS daemons (important!)

```
crayadm@smw> sudo /usr/bin/systemctl start rsms.service
```

7. Run `xtdiscover` to pick up any changes to the system since the backup was taken (or all of the database, if a backup was not loaded in the previous step).

```
crayadm@smw> sudo xtdiscover
```

Regenerate the HSS MariaDB Database from Scratch

About this task

If the HSS MariaDB database becomes corrupt or is missing, and all attempts to repair or restore it have failed, use this procedure to regenerate the database from scratch. Deleting the contents of `/var/lib/mysql` removes everything that stores MariaDB state, including the password (hence the need to re-create it). When MariaDB is restarted and its directory is empty, `/var/lib/mysql` will be re-initialized.

Procedure

1. Stop the HSS daemons (by stopping RSMS) and the MariaDB service.

```
crayadm@smw> sudo /usr/bin/systemctl stop rsms.service
crayadm@smw> sudo /usr/bin/systemctl stop mysql.service
```

2. Remove the damaged database.

```
crayadm@smw> sudo mkdir /var/lib/mysql.bad
crayadm@smw> sudo mv /var/lib/mysql/* /var/lib/mysql.bad
crayadm@smw> sudo mv /var/lib/mysql/.??* /var/lib/mysql.bad
```

The `/var/lib/mysql` directory is the mount point for a file system from the boot RAID, so it cannot simply be removed. However, its contents can be removed (moved). The `/var/lib/mysql` directory will be newly initialized when the MariaDB service is restarted.

3. Restart MariaDB.

```
crayadm@smw> sudo /usr/bin/systemctl start mysql.service
```

The database directory is reinitialized, and the default password is set to the empty string.

4. Reset the MariaDB root password and update the `/root/.my.cnf` file.

   a. Reset the MariaDB root password to its former value.

```
smw# mysqladmin -uroot password -p
```

   Do one of the following at the prompt:

   - To **set** the root password for fresh installs or after the database has been reinitialized, press **Enter** to enter an empty string, the default initial password.

```
Enter password: <cr>
```

- To **change** the root password, enter the existing password.

```
Enter password: existing_password
```

At these prompts, enter the new root password, and then enter it again.

```
New password:
Confirm new password:
```

b. Ensure that the root password in the `/root/.my.cnf` file matches the new root password.

```
smw# vi /root/.my.cnf
[client]
user=root
password=MariaDB-password
```

If this file does not yet exist, create it and add the lines shown in the example, substituting the new password for the placeholder *MariaDB-password*.

Ensure that only root can see or write to this file.

```
smw# chmod 600 /root/.my.cnf
```

5. Initialize the HSS database tables and restore user permission tables.

```
crayadm@smw> hssds_init
crayadm@smw> dbgrant
```

The system will prompt for a password after each of the above two commands. Give the newly reset MariaDB root password each time.

6. Restart the HSS daemons (important!).

```
crayadm@smw> sudo /usr/bin/systemctl start rsms.service
```

7. Run `xtdiscover` twice (first with the `--bootstrap` option) to regenerate the database.

```
crayadm@smw> sudo xtdiscover --bootstrap
crayadm@smw> sudo xtdiscover
```

Troubleshoot Temperature Warnings Reported in an End Cabinet

About this task

If the consumer log or `xtcheckhss` reports temperature warnings in an end-of-row cabinet of a liquid-cooled system, the current `hss.ini` file may not have the necessary temperature set point defined, or the set point value may not be appropriate for the site. Use this procedure to ensure that this temperature set point is defined and is set to an appropriate value.

Details	In a liquid-cooled cabinet with chassis (cages) that are unevenly populated, the exit temperatures in each cage will be very different. In a normal cabinet, the water valve is controlled by the average temperature of the hottest temperature strip. By contrast, the water valve in an end-of-row cabinet is

controlled by the average temperature of all temperature strips. This may lead to inadequate cooling of a populated cage if the other two cages are not populated or have minimal heat load.

To avoid problems arising from inadequate cooling, the exit air temperatures of the end-of-row cabinet can be independently controlled. This is achieved through an entry in the `hss.ini` file that sets the end-of-row cabinet exit temperature lower than that of other cabinets. The default value is 22°C; however this should be adjusted to meet site-specific requirements. If the end cabinet exit air temperature is not defined in the `hss.ini` file, the air temperature will default to the setting defined for the other cabinets in the cooling row.

What to look for

The consumer log may show entries similar to the example below:

```
Mon Jul 28 05:59:47 2014 - rs_event_t at 0x7f5bc0000920
ev_id = 0x080040ed (ec_l1_failed)
ev_src = ::c1-0
ev_gen = ::c0-0c0s0n0
ev_flag = 0x00000002 ev_priority = 0 ev_len = 158 ev_seqnum = 0x00000000
ev_stp = 53d5e6d3.0000176d [Mon Jul 28 05:59:47 2014]
svcid 0: ::c1-0 = svid_inst=0x0/svid_type=0x0/svid_node=c1-0[rsn_node=0x0/
rsn_type=0x3/rsn_state=0x6], err code 65914
- Cabinet Controller Temperature Fault
ev_data...
00000000: 01 00 00 00 00 00 00 00 00 00 00 0c 06 00 00 *................*
00000010: 04 00 00 00 00 00 00 00 01 00 00 00 7a 01 01 00 *............z...*
00000020: 7a 00 00 00 30 39 34 7c 57 41 52 4e 7c 54 45 4d *z...094|WARN|TEM*
00000030: 50 7c 2f 64 61 74 61 2f 63 6f 6d 70 75 74 65 5f *P|/data/compute_*
00000040: 63 61 62 69 6e 65 74 2f 61 69 72 5f 73 65 6e 73 *cabinet/air_sens*
00000050: 6f 72 73 2f 63 68 32 2f 61 69 72 5f 74 65 6d 70 *ors/ch2/air_temp*
00000060: 32 3a 64 65 67 63 2a 31 30 30 7c 4d 61 78 69 6d *2:degc*100|Maxim*
00000070: 75 6d 20 73 6f 66 74 20 6c 69 6d 69 74 20 65 78 *um soft limit ex*
00000080: 63 65 65 64 65 64 21 7c 44 61 74 61 3d 33 30 30 *ceeded!|Data=300*
00000090: 32 7c 4c 69 6d 69 74 3d 33 30 30 30 2e 00       *2|Limit=3000....*
```

With `xtcheckhss`, the problem may look like this:

```
No Version Mismatches Found!
============================================
========== Sensor Warnings =================
============================================
Component Module Sensor HMIN SMIN DATA UNIT SMAX HMAX
--------- ------------ ------------ ---- ---- ---- -------- ---- ----
c2-0 compute_cabinet ambient_temp0 30 50 324 degc*10 300 350
c2-0 compute_cabinet ambient_temp1 30 50 306 degc*10 300 350
c2-0 compute_cabinet ch0_air_temp0 0 1000 3486 degc*100 3000 3500
c2-0 compute_cabinet ch0_air_temp1 0 1000 3355 degc*100 3000 3500
c2-0 compute_cabinet ch0_air_temp2 0 1000 3338 degc*100 3000 3500
c2-0 compute_cabinet ch0_air_temp3 0 1000 3486 degc*100 3000 3500

No SEEP Errors Found!
No ITP Errors Found!
No NTP Time Sync Errors Found!
No Control Errors Found!
```

Procedure

1. Edit `hss.ini`.

   Open the `/opt/tftpboot/ccrd/hss.ini` and look for the following entry.

   ```
   crayadm@smw> vi /opt/tftpboot/ccrd/hss.ini
   ```

```
## ----------------- END CABINET -----------------
# This group is used to define the attributes that are only applied to the end
cabinet
# of a row. The attributes defined here will override the same attributes in
group [ccrd]
# above. If no attributes are defined in this group the end cabinet will be
configured
# using the attributes of group [ccrd].
[endcabinet]
#define the temperature setpoint for the last cabinet in a row
temp_setpoint=22
```

2. Adjust the value of `temp_setpoint` as appropriate for the installation site.

   To determine an appropriate value, consider the following:

   - The inlet water temperature, which should be below the exit air temperature setting.
   - The facility room environment.

Recover from SMW R630 Boot Disk Hardware RAIDS Failure

If one of the disks in the SMW R630, which is part of the hardware RAID5, fails, the hot spare will take over and the data will be rebuilt using the remaining drives. The bad drive should be removed. When a new disk is inserted into the SMW, the hardware RAID will begin the process of adding it back into the RAID5 set of drives.

This procedure does not apply to the SMW R815 which has software RAID1 for the boot disk.

Recover from SMW R815 Boot Disk Software RAID1 Failure

About this task

If one of the disks in the SMW R815, part of the software RAID1 mirror, fails, corrective action should be taken.

This procedure does not apply to the SMW R630 which has hardware RAID5 for the boot disk.

Procedure

1. Check status of RAID1 filesystems.

   a. Confirm that all RAID1 filesystems are fully synced.

   ```
   smw# cat /proc/mdstat
   ```

   b. Get detailed information on RAID1 devices. `swap` is on `/dev/md125`, `/boot` is on `/dev/md126`, and `/` is on `/dev/md127`.

   ```
   smw# mdadm -D /dev/md125
   smw# mdadm -D /dev/md126
   smw# mdadm -D /dev/md127
   ```

2. Replace the failed disk drive in slot 0 on the SMW.

   a. Shutdown CLE if still booted before the next step of shutting down and booting the SMW itself.

   ```
   crayadm@smw> xtbootsys -s last -a auto.xtshutdown
   ```

b. Shutdown SMW.

```
smw# shutdown -h now
```

c. Remove the failed disk drive in slot 0 of the SMW so that drive 1 will become the bootable disk.

d. Boot SMW from drive 1. System boots from drive 1, but calls it `/dev/sda` since it is the first drive found and there is no drive in slot 0.

e. Remove failed drive from RAID1 configuration.

```
smw# mdadm --manage /dev/md127 --fail /dev/sda1
smw# mdadm --manage /dev/md127 --remove /dev/sda1
smw# mdadm --manage /dev/md126 --fail /dev/sda3
smw# mdadm --manage /dev/md126 --remove /dev/sda3
smw# mdadm --manage /dev/md125 --fail /dev/sda2
smw# mdadm --manage /dev/md125 --remove /dev/sda2
```

f. Replace drive 0. The system still runs.

g. Reboot the SMW.

```
smw# reboot
```

h. Check RAID1 status.

System boots and immediately will use `/dev/md125` (`swap`) as shown by this command with [`UU`], however, `md126` and `md127` show [`_U`] indicating a degraded state.

```
smw# cat /proc/mdstat
```

`mdadm` shows active sync for both drives in `/dev/md125` (`/dev/sda2` and `/dev/sdb2`).

```
smw# mdadm -D /dev/md125
```

`mdadm` shows removed for drive 0 but active sync for `/dev/sdb1` in `/dev/md127` and `/dev/sdb3` in `/dev/md126`.

```
smw# mdadm -D /dev/md126
smw# mdadm -D /dev/md127
```

i. Partition new drive correctly using `sfdisk` or `fdisk` so it matches drive 1.

```
smw# sfdisk -d /dev/sdb | sfdisk --force /dev/sda
```

j. Add drive 0 back to RAID1 configuration to reconstruct degraded RAID1.

```
smw# mdadm -v --manage /dev/md126 --add /dev/sda3
smw# mdadm -v --manage /dev/md127 --add /dev/sda1
```

k. Check status of RAID1 rebuild with these commands.

```
smw# mdadm -D /dev/md126
smw# mdadm -D /dev/md127
```

Checking `mdstat` will display the percentage of recovery and an estimate of when it will complete for each device being reconstructed.

```
smw# cat /proc/mdstat
```

When all reconstruction is complete, `mdstat` will display the percentage of recovery and an estimate of when it will complete for each device being reconstructed.

```
smw# cat /proc/mdstat
```

3. Replace the failed disk drive in slot 1 of the SMW. If drive 1 is removed, then the process is similar to drive 0 above, but there are differences.

   a. Confirm that all RAID1 filesystems are fully synced.

   ```
   smw# cat /proc/mdstat
   ```

   b. Get detailed information on RAID1 devices.

   ```
   smw# mdadm -D /dev/md125
   smw# mdadm -D /dev/md126
   smw# mdadm -D /dev/md127
   ```

   c. Shutdown CLE, if CLE is still booted, before the next step of shutting down and booting the SMW itself.

   d. Shutdown SMW.

   ```
   smw# shutdown -h now
   ```

   e. Remove the failed disk drive in slot 1 of the SMW so that drive 0 will become the bootable disk.

   ```
   smw# mdadm --manage /dev/md127 --fail /dev/sdb1
   smw# mdadm --manage /dev/md127 --remove /dev/sdb1
   smw# mdadm --manage /dev/md126 --fail /dev/sdb3
   smw# mdadm --manage /dev/md126 --remove /dev/sdb3
   smw# mdadm --manage /dev/md125 --fail /dev/sdb2
   smw# mdadm --manage /dev/md125 --remove /dev/sdb2
   ```

   f. Boot SMW from drive 0.

   g. Replace drive 1. The SMW still runs, but in degraded mode for RAID1 devices. One of the other disks (local to SMW or in boot RAID) will be called `/dev/sdb`.

   h. Reboot SMW so that drive 1 will appear as `/dev/sdb`.

   ```
   smw# reboot
   ```

   i. Check RAID 1 status. System boots and, unlike with disk 0 above, will not immediately use `/dev/md125` (`swap`) as shown by this command with [`U_`], also, `md126` and `md127` show [`U_`] indicating a degraded state.

   ```
   smw# cat /proc/mdstat
   ```

   `mdadm` shows removed for drive 1 but active sync for `/dev/sda1` in `/dev/md127` and `/dev/sda3` in `/dev/md/126` and `/dev/sda2` in `/dev/md125`.

   ```
   smw# mdadm -D /dev/md125
   smw# mdadm -D /dev/md126
   smw# mdadm -D /dev/md127
   ```

   j. Partition new drive correctly using `sfdisk` or `fdisk` so it matches drive 1.

   ```
   smw# sfdisk -d /dev/sda | sfdisk --force /dev/sdb
   ```

k. Add Drive 1 back to RAID1 configuration.

```
smw# mdadm -v --manage /dev/md125 --add /dev/sdb2
smw# mdadm -v --manage /dev/md126 --add /dev/sdb3
smw# mdadm -v --manage /dev/md127 --add /dev/sdb1
```

l. Check status of RAID1 rebuild with these commands.

```
smw# mdadm -v --manage /dev/md125 --add /dev/sdb2
smw# mdadm -v --manage /dev/md126 --add /dev/sdb3
smw# mdadm -v --manage /dev/md127 --add /dev/sdb1
```

Checking `mdstat` will display the percentage of recovery and an estimate of when it will complete for each device being reconstructed.

```
smw# cat /proc/mdstat
```

When all reconstruction is complete, `mdstat` should show all drives as [`UU`].

```
smw# cat /proc/mdstat
```

About X.509 Certificates and How to Redistribute Them

Some features of Cray XC systems, such as Cray Advanced Platform Monitoring and Control (CAPMC), use X.509 certificate authority files (certificates) for access authorization. These certificates are generated and managed using the `xtmake_ca` tool. The certificate authority (CA) resides on the SMW and is typically generated during the SMW software installation process; however, there may be occasion to rebuild the CA from scratch. The `xtmake_ca` man page describes how to do this, but it does not provide details about what certificates are used, where they are used, and how to redistribute them after rebuilding a CA from scratch. This topic fills that gap.

Here is a summary; details follow.

What uses certs	Certs used	Where	How redistributed

CAPMC API service	`certificate_authority.crt`
	`certificate_authority.crl`
	`hosts/host.crt`
	`hosts/host.key`
	`client/xtremoted.crt`
	`client/xtremoted.key`
		SMW	reconfigure and restart CAPMC API service

CAPMC SDB node service	`certificate_authority.crt`
	`host/sdb-p0.crt`
	`host/sdb-p0.key`
		SDB node	update and apply config set

DataWarp service	`certificate_authority.crt`
	`/etc/opt/cray/dws/`*`$dw_node_name`*`.crt`
	`/etc/opt/cray/dws/`*`$dw_node_name`*`.key`
		DataWarp service nodes	update and apply config set

What uses certs	Certs used	Where	How redistributed
capmc	certificate_authority.crt client/client.crt client/client.key	SMW	move aside existing capmc configuration directory and rerun xtremoted_setup

In the default set of certificates that follows, file paths are specified relative to the certificate authority directory: /var/opt/cray/certificate_authority.

Certificate Authority

Certificates used to maintain the CA include:

certificate_authority.crt This is the root certificate in which the SMW CA is based. It is used to validate the authenticity of all other certificates created by the SMW private CA. It must be distributed to all clients and services that use certificates generated by the SMW CA.

certificate_authority.key This is the CA private key file, which must be kept private at all times. It must never be distributed to any system.

certificate_authority.crl This is an optional certificate revocation list. It is a PEM-encoded certificate containing a list of serial numbers that identify any client access or host certificates that have been revoked. certificate_authority.crl is rebuilt each time xtmake_ca buildcrl is invoked.

CAPMC API Service

The CAPMC API service runs on the SMW. It is implemented by nginx, a standard HTTP server that provides encrypted communications and client authorization, and xtremoted, which handles client requests that have been authorized by nginx.

Certificates used The following certificates are used by the HTTP server (nginx) on the SMW.

certificate_authority.crt nginx uses this certificate to validate that the client access certificate, presented when a client first connects, was issued by the SMW CA. If the certificate was not issued by the local SMW CA, the client is denied access.

certificate_authority.crl If this file exists, the HTTP server checks it for client access certificates that have been revoked. Any client with a revoked certificate is denied access.

hosts/host.crt This is the host certificate used by the HTTP server to enable encrypted communications. It is generated auotmatically at the time of SMW installation, or when a system administrator takes an explicit action to regenerate them using xtmake_ca. The Common Name (CN) field of the certificate subject line should match the DNS hostname associated with the SMW. This certificate implements the

	X509v3 Subject Alternative Name extension, which uses a list of DNS attribute values to specify additional host names that a client should consider valid. The default list of DNS attribute values includes these two elements: • the fully qualified domain name (FQDN) of the SMW • the text string literal "`smw`"
`hosts/host.key`	This is the private key associated with the SMW host certificate.
`client/xtremoted.crt`	This is the client access certificate used by `xtremoted` to identify itself to remote procedure call handlers. This is needed because some API calls require `xtremoted` to forward a client's request to another server running on the target partition's system database (SDB) node (see *CAPMC SDB Node Service* below).
`client/xtremoted.key`	This is the private key associated with the client access certificate.

How to redistribute

If the CA has been rebuilt from scratch, `certificate_authority.crl` has been rebuilt, or `hosts/host.crt` has been modified, reconfigure and restart the CAPMC API service (as root):

```
smw# xtremoted_setup
```

This command restarts the CAPMC API service and copies relevant files, with appropriate permissions, into a directory owned by that `xtremoted` userid (`/opt/cray/hss/default/etc/xtremoted`). This copy is necessary because the userid that the `xtremoted` process is running under does not have read access to files located within the `certificate_authority` directory.

CAPMC SDB Node Service

The CAPMC SDB node service handles remote procedure call requests issued from the CAPMC API service running on the SMW. It is implemented by `nginx`, a front-end HTTP server that performs encryption and client access authorization, and `xtremoted-agent`, a remote procedure call handler that handles the specific request.

Certificates used

The following certificates are used by the HTTP server (`nginx`) on the SDB node.

`certificate_authority.crt`	`nginx` running on the SDB node uses this certificate to validate that the clie certificate, presented when `xtremoted` issues a remote procedure call requ issued by the SMW CA. If the certificate was not issued by the local SMW C is denied. In addition, the CN field of the client access certificate subject line the string "`xtremoted`" for the request to be accepted.
`hosts/sdb-p0.crt`	This is the host certificate for the SDB node and config set `p0`.
`hosts/sdb-p0.key`	This is the private key associated with the SDB node host certificate and cor

How to redistribute

If the CA has been rebuilt from scratch, update the config set and apply it.

1. Update the current configuration set (as root):

```
smw# cfgset update -m auto p0
```

   When the config set is updated, the config set gets the new certificates by means of the `xremoted_agent` configuration callback script (`/opt/cray/imps_config/system-config/default/configurator/callbacks/post/xtremot` which updates the certificates from the `/var/opt/cray/certificate_authority` location to the config updated.

2. Reboot the system. When the node boots, the config set certificate files are copied from the config set to the Ansible play.

3. After the Ansible play has run, verify that the certificates have been distributed.

```
smw> ls -la /var/opt/cray/imps/config/sets/p0/files/roles/common/etc/opt/cray/x
agent
total 12
drwxr-xr-x 1 root root   90 Dec  7 15:39 .
drwxr-xr-x 1 root root   42 Dec  7 15:39 ..
-rw------- 1 root root  956 Dec  9 11:18 certificate_authority.crt
-rw------- 1 root root 3002 Dec  9 11:18 sdb-p0.crt
-rw------- 1 root root  916 Dec  9 11:18 sdb-p0.key
```

DataWarp Service Nodes

DataWarp service nodes (and elogin and compute nodes as well) use the SSL certificates only to connect to the HTTP API. The client certificates are not essential because they can be regenerated. What is essential is that the CA on the SMW is trusted on the remote nodes.

Certificates used

The following certificates are used primarily at the login node and any elogin node. Copies of the cert chain are made so that client compute nodes and service nodes are able to run tools that interact with the DataWarp API with no problems.

certificate_authority.crt	This file is synced with the certificate on the DataWarp service.
hosts/$dw_node_name.crt	This file is synced with the certificate on the DataWarp service.
hosts/$dw_node_name.key	This file is synced with the certificate on the DataWarp service.
/etc/opt/cray/dws/$dw_node_name.crt	This is the certificate on the DataWarp service.
/etc/opt/cray/dws/$dw_node_name.key	This is the private key on the DataWarp service.

How to redistribute

Certificates are deployed initially by means of the configurator and Ansible plays when the DataWarp service is set up. The Ansible plays generate the certificates using `xtmake_ca` and synchronize the certificate authority to the remote nodes as needed. If the CA has been rebuilt from scratch, update the config set and apply it.

1. Update the current configuration set (as root):

```
smw# cfgset update -m auto p0
```

When the config set is updated, the config set gets the new certificates by means of a post-configuration callback script, which updates the certificates from the `/var/opt/cray/certificate_authority` location to the config set being updated.

2. Reboot the system. When the node boots, the config set certificate files are copied from the config set to the node using an Ansible play.

Troubleshooting

Problem:

- `capmc` outputs a hostname mismatch error.

```
smw:/etc/opt/cray/capmc # capmc node_rules
    Error - Certificate identity does not match the target hostname
```

Possible Causes:

- The `capmc` client configuration, (`/etc/opt/cray/capmc/capmc.json`) `os_service_url`, setting is invalid.

  When capmc is being executed from the SMW on an internal Cray service node running the Cray Linux Environment, the `os_service_url` setting should configured as follows:

```
https://smw:8443
```

  When `capmc` is being executed from an external system, the `os_service_url` setting should include the fully qualified domain name of the SMW as follows:

```
https://my-smw.my-domain.com:8443
```

  - Resolution:
    - Reconfigure the `os_service_url` parameter.
- The SMW capmc API server host certificate contains an incorrect list of acceptable DNS names.

  Verify the "Subject Alternative Name" DNS name list contains the SMW FQDN and short hosname `smw`:

```
smw:/etc/opt/cray/capmc # openssl x509 -text -noout \
                -in /var/opt/cray/certificate_authority/hosts/host.crt | \
                grep -A 1 "Subject Alternative Name"

X509v3 Subject Alternative Name:
                DNS:my-smw.my-domain.com, DNS:smw
```

  - Resolution:
    - Regenerate the SMW host server certificate.

Problem:

- `capmc` outputs a certificate verification error.

```
smw:/etc/opt/cray/capmc # capmc node_rules
    Error - url(https://smw:8443/capmc/get_node_rules) \
    [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:581)
```

Possible Causes:

- The client's copy of the CA certificate is not from the actual certificate authority that generated the SMW CAPMC API server certificate.
  - Resolution:
    - Redistribute the certificate_authority.crt file from the SMW to the client system.
- The SMW CAPMC API server was not restarted after regenerating the certificate authority from scratch.
  - Resolution:
    - Reconfigure the capmc API server by invoking `xtremoted_setup`.

Problem

- Capmc client connection times out. IP connectivity is non-functional between the `capmc` client system and the SMW.

```
smw:/etc/opt/cray/capmc # capmc node_rules
    Error - url(https://smw:8443/capmc/get_node_rules) \
    [Errno 113] No route to host
```

Possible Causes:

- `capmc` client `os_service_url` is configured incorrectly.
  - Resolution:
    - For use on internal Cray service nodes, reconfigure the `os_service_url` to `https://smw:8443`.
    - For use on external nodes, reconfigure the `os_service_url` to be the SMW's fully qualified domain name and verify that a valid IP connectivity path is established.
- When using `capmc` from an internal Cray service node, the IP path taken is over the high speed network, to the boot node, and on the SMW. IP Routing tables may be misconfigured on the SMW, boot node, or internal service node.
  - Resolution:
    - Verify the boot node has IP forwarding enabled.

      ```
      boot-p0:~ # sysctl net.ipv4.ip_forward
                    net.ipv4.ip_forward = 1
      ```

    - Verify the boot node firewall has TCP port 8443 open.

      ```
      boot-p0:~ # iptables -L
                    ...
      ```

    - Verify the SMW has a return route on an internal interface to the high speed network via the boot node.

      For example:

      ```
      smw:/etc/opt/cray/capmc # netstat -rn
                    Kernel IP routing table
                    Destination Gateway    Genmask              Iface
                    ...
                    10.128.0.0  10.3.1.254 255.255.0.0 UG 0 0 0 eth3
                    ...
      ```

- Verify the internal Cray service node has a route to the SMW's internal interface via the boot node.

  For example:

```
svc-node:~ # netstat -rn
                Kernel IP routing table
                Destination Gateway         Genmask ...
Iface
                ...
                10.3.1.1    10.128.255.254  255.255.255.255 UGH 0 0 0
ipogif0
                ...
```

Update X.509 Host Certificate After SMW Hostname Change

About this task

Whenever the SMW hostname changes, the previously generated x509 SMW host certificate `host.crt` file will need to be updated. Failure to perform this step will prevent the `capmc` client from connecting to the SMW, due to a host name certificate validation error.

Procedure

1. Create a backup copy of the `certificate_authority` directory.

```
smw:~# cd /var/opt/cray
smw:~# cp -a certificate_authority certificate_authority.backup
```

2. Run the host validation.

```
smw:~# xtmake_ca validate
```

   You will be notified if the `host.crt` file has a common name that does not match the current hostname.

```
..
 - CN in SMW host file matches current hostname (my-smw.example.com)
Alternate names: my-smw.example.com, smw  - SMW host certificate file
validation succeeded.
..
```

3. Generate the new certificate.

   - If the SMW was only renamed, rebuild the host certificate using the new hostname.

```
smw:~# xtmake_ca update
```

   - If a specific SMW hostname or list of alternate names must be specified, manually revoke the SMW host server certificate and recreate it with a list of appropriate hostnames.

```
smw:~# xtmake_ca revoke \
/var/opt/cray/certificate_authority/hosts/host.crt
smw:~# xtmake_ca CN=my-smw.example.com,my-smw.local,my-smw
```

     NOTE: This does not require remaking or redistributing existing certificates. `xtmake_ca` will recreate only missing certificates. In this case, the only missing certificate should be SMW host

certificate which was intentionally revoked. Any services, such as `nginx`, running on the SMW which are using the rebuilt host certificate should be restarted.

This step generates a new host certificate with the currently assigned hostname is listed in the **CN** field, as well as a list of additional DNS names which `capmc` should consider valid.

4. Run the host validation again.

```
smw:~# xtmake_ca validate
```

5. Reconfigure and restart `nginx` on the SMW.

```
smw:~# xtremoted_setup
```

6. View the contents of the newly generated SMW host server certificate.

```
smw:~# openssl x509 -noout -text -in
/var/opt/cray/certificate_authority/hosts/host.crt
```

Manage System Access

Change Account Passwords on the SMW

About this task

The SMW contains its own `/etc/passwd` and `/etc/shadow` files that are separate from the files for the rest of the CLE system.

Procedure

Execute the following commands to change the passwords on the SMW for the following Linux accounts.

```
smw# passwd root
smw# passwd crayadm
smw# passwd mysql
```

Change Account Passwords on CLE Nodes

About this task

Use this procedure to change a password for an account that is local to the CLE nodes, such as `root` and `crayadm`.

For LDAP or other authentication services, passwords are changed through those services.

Procedure

1. Update passwords in `cray_local_users`.

   a. Update the CLE config set to change passwords for `root` (`cray_local_users.settings.users.data.root.crypt`) and `crayadm` (`cray_local_users.settings.users.data.crayadm.crypt`).

      Full system:

      ```
      smw# cfgset update -s cray_local_users -l advanced -m interactive p0
      ```

      Partitioned system (update a config set for each partition):

      ```
      smw# cfgset update -s cray_local_users -l advanced -m interactive p1
      smw# cfgset update -s cray_local_users -l advanced -m interactive p2
      ```

2. Validate config set.

   Full system:

```
smw# cfgset validate p0
```

Partitioned system:

```
smw# cfgset validate p1
smw# cfgset validate p2
```

3. Activate new passwords for local accounts. The password changes can be made immediately on the CLE nodes or can take effect at the next boot of the nodes.

   a. Activate new passwords immediately on nodes. Doing so immediately does not require a reboot of the node, merely running `cray-ansible` again.

      On the boot node:

      ```
      boot# /etc/init.d/cray-ansible start
      ```

      On the SDB node:

      ```
      sdb# /etc/init.d/cray-ansible start
      ```

      On all service nodes:

      ```
      sdb# pcmd -r -n ALL_SERVICE_NOT_ME "/etc/init.d/cray-ansible start"
      ```

      On all compute nodes:

      ```
      sdb# pcmd -r -n ALL_COMPUTE "/etc/init.d/cray-ansible start"
      ```

4. Activate new passwords by rebooting nodes. Either a full system reboot or warm booting individual nodes will cause `cray-ansible` to activate these new passwords on the CLE nodes.

Configure the System

Cray XC System Configuration

To configure Cray XC systems and manage configuration content, system administrators use the Cray configuration management framework (CMF). The CMF comprises configuration data, the tools to manage and distribute that data, and software to apply the configuration data to the running image at boot time. Its major components include configuration service packages, config sets, the IMPS (Image Management and Provisioning System) distribution service (IDS), the configurator, and Ansible.

Configuration Starts with Configuration Service Packages

Configuration content (data and software) is installed as configuration service packages on the management node of Cray XC systems (in `/opt/cray/imps_config/<service package>/default/configurator` by default). Each service package delivers configuration content for one or more system services. The contents of each service package reside in the following subdirectories:

`ansible`	Drop zone for Cray-provided Ansible play content.
`callbacks`	Pre- and post-configuration scripts.
`dist`	Drop zone for other Cray-provided content, such as static files required for the configuration of a service.
`template`	Configuration templates that define the configuration settings to be set and provide some default values. These templates are never modified by administrators or other users.

Configuration service packages are installed for system upgrades and updates as well as for initial installation.

Configuration Information is Stored in Config Sets

Administrators use the `cfgset` command to manage configuration information. It takes configuration content delivered in service packages and invokes the *configurator* tool to combine that content with site-specific configuration content gathered from administrators either interactively or through bulk import. The results are used by `cfgset` to create a configuration set or *config set*. A config set is a central repository that stores all configuration information necessary to operate the system. Config sets reside on the management node (e.g., the SMW) in `/var/opt/cray/imps/config/sets` by default. The contents of each config set reside in the following subdirectories:

`ansible`	Drop zone for local site-provided Ansible play content to be distributed with the config set. When the config set is created, `cfgset` copies Ansible content from service packages to this location. Whenever the config set is updated, `cfgset` copies Ansible content from service packages again, overwriting the previous service-package Ansible content and leaving the site-provided content unchanged.
`changelog`	YAML change logs from previous sessions with the configurator.

config	Configuration templates containing configuration information. When the config set is created, the configurator copies service package templates to this location. Administrators can modify the content of these templates using `cfgset` and the configurator. Whenever the config set is updated, the configurator merges service package templates with the templates in this location.
dist	Drop zone for other site-provided content, such as static files required for the configuration of a service. When the config set is created, `cfgset` copies dist content from service packages to this location. Whenever the config set is updated, `cfgset` copies dist content from service packages again, overwriting the previous service-package dist content and leaving the site-provided content unchanged.
files	Files necessary for system configuration that are generated by configuration callback scripts or manually and distributed with the config set (e.g., `/etc/hosts`).
worksheets	Configuration worksheets generated by the configurator using data stored in the configuration templates in the `config` subdirectory of the config set. Administrators copy these worksheets to a location outside the config set, edit them with site-specific configuration data, and then import them to create a new config set or update an existing one.

An administrator may create multiple config sets to support partitions or alternate configurations. Typically a config set of type `cle` is created for each partition to store partition- and CLE-specific content, and another config set of type `global` is created to store management node and global configuration data.

IDS Distributes Config Sets to Nodes

IDS, a read-only network share of content from the management node to the rest of the system, distributes config sets to every node in the system. All config sets are shared throughout the system, but only one `cle` config set is active on a given node at a time (in addition to an active `global` config set, which is applies to the entire system). Currently, IDS leverages the 9P network file system and the Linux automounter facility as its distribution mechanism; however, the content and use of the config sets is independent of the distribution mechanism.

Ansible Plays Apply Configuration during System Boot

Prior to booting the system, each node will have an image, a `global` config set, and a `cle` config set. When the system boots, each node boots an unconfigured software image. Then Ansible plays, which can be located in both the image and the config set (config set is the preferred location for site-supplied Ansible plays), apply configuration to that image, bringing up the services pertinent to each node.

Administrators Configure/Reconfigure the System on an Ongoing Basis

Configuration happens at times other than initial installation. New configuration service packages can be installed during system upgrades and updates, sites can decide to enable a new service or change the configuration of an existing service, and so forth. In all of these scenarios, an administrator uses the `cfgset` command to manage config sets and the `cray-ansible` script to apply any configuration changes. The `cfgset` command and its associated subcommands and options enable administrators to perform a variety of operations on config sets in addition to create and update, such as search, diff, list, show, validate, push, and remove. See the `cfgset` man page for a description of its subcommands and options and some examples of each.

About the Configurator

The configurator plays a major role in Cray XC system configuration. The configurator gathers configuration data from several sources (including the user, with helpful prompts and default values), merges and validates it, and

stores it in a central location on the management node, where it is used during boot to configure the entire system. The configurator is invoked by the `cfgset` command to:

- handle all configuration template and worksheet operations
- perform steps 4, 5, and 6 of the *Config Set Create/Update Process*, including providing a user interface to gather and modify configuration data interactively or through the import of configuration worksheets

The configurator is invoked with the `cfgset` subcommands `create` (except when the `--clone` option used) and `update`. It is invoked also with the `search` subcommand, because that involves searching data stored in the configuration templates, but no changes are made to the config set using `search`. The options selected for the `create` and `update` subcommands determine the mode in which the configurator is run (with or without user interaction), which settings can be viewed and set by a user, and whether callback scripts are run before and after the configurator session. The configurator is not involved when the remaining `cfgset` subcommands are used: `diff`, `list`, `push`, `remove`, `show`, and `validate`. See the `cfgset` man page for a description of its subcommands and options and some examples of each, or use `cfgset` *`SUBCOMMAND`* `-h` to see information about just one of the subcommands.

Choose How to Interact with the Configurator: Modes

The `mode` option of the `cfgset` command determines how the configurator interacts with a user. Mode can be specified only with subcommands `create` and `update`.

`--mode` | `-m` Possible values: `auto` (default), `interactive`, `prepare`

`auto` The configurator searches through all available configuration templates in the config set and automatically presents all configuration settings that meet state and level filtering criteria. It presents the configuration settings in a certain order (taking into account dependencies among services) one at a time until all have been presented to the user, and then it automatically ends the session and saves the config set.

`interactive` The configurator searches through templates as with auto mode, but in interactive mode, it presents a menu of all available services (or a menu of all available settings, when a service has been selected) that meet state and level filtering criteria. This mode enables the user to navigate through the services and settings to view and modify the settings as needed. The configuration session ends when the user exits the session. The user chooses whether to save any changes to the config set upon exit.

`prepare` The configurator prepares configuration worksheets, one for each service. Each worksheet contains all configuration settings (unfiltered) for that service, and the worksheet can be edited offline and then imported later to create or update a config set. In this mode, the configurator does not open an interactive session with the user.

Choose What to See with the Configurator: Filters

Two `cfgset` command options act as filters to determine which settings are available to view and set or update. These options can be specified only with subcommands `create`, `update`, and `search`.

`--state` | `-S` Possible values: `unset` (default), `set`, `all`

`--level` | `-l` Possible values: `required`, `basic` (default), `advanced`

required Settings that must be set or the system will not function. The config set will not validate if any required settings are skipped (i.e., left unset). Specify level `required` in a `cfgset` command to filter for required settings only.

basic Settings that are likely to be used by most sites. If a `basic` setting is left unset, the template-provided default is used. Specify level `basic` in a `cfgset` command to filter for both basic and required settings.

advanced Settings that are likely to be used only by advanced users to tune a service. If an `advanced` setting is left unset, the template-provided default is used. Specify level `advanced` in a `cfgset` command to filter for all settings: advanced, basic, and required.

Create a Config Set

Choosing the best strategy for creating a config set depends on the circumstances ("when to use"):

Strategy	When to use	Rationale
Create a Config Set from Configuration Worksheets	when performing fresh installs, major upgrades, or any time there is a large amount of configuration data to set up	Worksheets can be generated, filled out offline with site-specific data by the appropriate staff, and then imported when needed.
Create a Config Set by Cloning	when there is already a config set with site-specific data and additional config sets are needed with minor variations (for partitions, alternate configurations, etc.), or when manually backing up a config set	Cloning is quick, and it is easy to interactively update the clone with needed variations.
Create a Config Set without Callbacks	when no hardware is attached to the XC system, as in some testing scenarios	Pre- and post-configuration callback scripts may invoke utilities that query hardware in order to provide additional config set content.
Create a Config Set Interactively	when configuring a smaller system with little configuration data to change	Setting all configuration values one at a time in response to a series of prompts or when selected from a menu can be very time-consuming.

These strategies all use the `cfgset` command. Use `cfgset create -h` for information about the create subcommand. See *Config Set Create/Update Process* on page 116 for an outline of the process followed by `cfgset` each time the `create` or `update` subcommand is used.

Note that when the `create` subcommand is used in any of these strategies (except cloning), it is necessary to specify the config set type for any type other than the default `cle`. Most of the following `create` procedures omit `--type` because they are for config sets of type `cle`.

REMEMBER: Run `cfgset` as root.

Create Backup Config Sets Automatically

If the `auto_clone` option in the IMPS configuration file (`/etc/opt/cray/imps/imps.json`) is enabled, the `cfgset create` and `cfgset update` commands will automatically clone a config set as a backup upon successful creation/update of the original config set. A failed operation will not create a backup.

The `autosave_limit` parameter in the IMPS configuration file determines how many clones will be retained. Config set backups are rotated with the oldest backup removed as a new backup is generated. Config set backups are saved with names of the form *`CONFIGSET`*`-autosave-`*`YYYY-MM-DDTHH:mm:SS`*, where *`CONFIGSET`* is the name of the original config set.

Create a Config Set from Configuration Worksheets

Prerequisites

This procedure has no prerequisites.

About this task

Use this procedure when performing fresh installs, major upgrades, or any time there is a large amount of configuration data to set up. To create a config set from configuration worksheets, use this process:

1. Generate the worksheets.
2. Copy the worksheets to a new location on the management node.
3. Edit the worksheets.
4. Import the worksheets.

The detailed steps of this procedure show an example of how to create config set `p0` of type `cle` (default) from configuration worksheets.

Note that the `cfgset` command is run as root.

Procedure

1. Generate new worksheets from configuration service packages installed on the system.

   ```
   smw# cfgset create --mode prepare p0
   ```

2. Locate the newly generated worksheets and copy them to a new location.

   ```
   smw# cfgset show --fields path p0
   p0:
     path: /var/opt/cray/imps/config/sets/p0

   smw# cp /var/opt/cray/imps/config/sets/p0/worksheets/* /some/edit/location
   ```

3. Edit the worksheets to customize them for this site.

   The system administrator typically distributes them to site staff members with knowledge about the services being configured so that they can edit the worksheets and enter appropriate values. Each worksheet is a YAML file that contains instructions on how to edit it; the basic idea is to locate the settings of interest, uncomment them, and either retain or change the default setting (if provided).

4. Import the completed worksheets using `cfgset update` or `cfgset create`.

Import the completed worksheets by updating the config set created when the worksheets were generated originally or by creating an entirely new config set. The argument to the `--worksheet-path` option is a file glob to allow multiple worksheets to be imported in a single create/update operation. Full paths to single worksheets can also be used.

- Import to the config set created with `--mode prepare` in step 1.

```
smw# cfgset update --worksheet-path '/some/edit/location/*_worksheet.yaml' p0
```

- Import to a new config set.

```
smw# cfgset create --worksheet-path '/some/edit/location/*_worksheet.yaml' \
 p0-new
```

REMEMBER: When using `cfgset` with the `--worksheet-path` option to import worksheets,

- Always add single quote marks around the worksheet path.
- Do not add mode, state, level, or service options; the configurator ignores them for worksheet import.
- The type of the config set must match the type of the worksheets being imported.

Create a Config Set by Cloning

Prerequisites

This procedure assumes that the config set to be cloned (the original) already exists.

About this task

Use this procedure when there is already a config set with site-specific data and additional config sets are needed with minor variations (for partitions, alternate configurations, etc.), or when manually backing up a config set. This procedure shows an example of creating config set `p0-new` by cloning it from existing config set `p0`. No callback scripts or configurator sessions occur when cloning a config set. The clone will have the same config set type as the original.

Note that the `cfgset` command is run as root.

Procedure

Create a clone using the `--clone` option.

```
smw# cfgset create --clone p0 p0-new
```

The configurator is not invoked when the `--clone` option is used, so no configurator session occurs, and no changes are made to the configuration data in the original config set.

Create a Config Set without Callbacks

Prerequisites

This procedure has no prerequisites.

About this task

Pre- and post-configuration callback scripts may invoke utilities that query hardware in order to provide additional config set content. Use this procedure when no hardware is attached to the XC system, as in some testing scenarios. This procedure shows an example of creating config set `global0` of type `global` from worksheets

while skipping all callback scripts. The `--no-scripts` option can also be used when creating a config set interactively.

Note that the `cfgset` command is run as root.

Procedure

Create a config set without callbacks.

```
smw# cfgset create --no-scripts --worksheet-path \
'/some/edit/location/*_worksheet.yaml' --type global global0
```

CAUTION: Skipping callback script processing invalidates a config set. A config set cannot be considered validated unless it is updated successfully without the `--no-scripts` option. Update all config sets to run the callback scripts before using the config set with the system.

Create a Config Set Interactively

Prerequisites

This procedure has no prerequisites.

About this task

This procedure shows examples of creating config set `p0` of type `cle` interactively. For additional examples, use `cfgset create -h`.

Note that the `cfgset` command is run as root.

Procedure

Invoke the configurator in auto mode (default) or interactive mode.

- **Auto mode.**

  To be presented with all settings with state `unset` (default) and level `basic` (default) in all services in config set `p0`:

  ```
  smw# cfgset create p0
  ```

  To be presented with all settings (any state and any level) in all services in config set `p0`:

  ```
  smw# cfgset create --state all --level advanced p0
  ```

- **Interactive mode.**

  To display a menu of services in config set `p0` that have configuration settings with state `unset` (default) and level `basic` (default):

  ```
  smw# cfgset create --mode interactive p0
  ```

  To display a menu of all services (with settings of any state and any level):

  ```
  smw# cfgset create --mode interactive --state all --level advanced p0
  ```

Update a Config Set

Choosing the best strategy for updating a config set depends on the circumstances ("when to use"):

Strategy	When to use	Rationale
Update a Config Set Interactively	when one or more config sets require a few changes (e.g., cloned config sets that need to be adjusted for a particular purpose), when a software update introduces just a few new fields to configure, or to confirm that all required and basic settings have been set (very useful!)	Setting just a few configuration values one at a time in response to a series of prompts or when selected from a menu works well when there are just a few settings that need to be configured or updated.
Update a Config Set from Configuration Worksheets	when performing system upgrades and updates, or any time there is a large amount of configuration data to change	Worksheets can be generated, filled out offline with site-specific data by the appropriate staff, and then imported when needed.
Update a Config Set without Callbacks	when no hardware is attached to the XC system, as in some testing scenarios	Pre- and post-configuration callback scripts may invoke utilities that query hardware in order to provide additional config set content.
Rename a Config Set	when a config set needs to be renamed as well as updated, or just renamed	This could become necessary for a variety of reasons.
Update a Single Service in a Config Set	when setting up a new service, or when just one service requires modification	This can be done either interactively or with worksheets, so refer to those circumstances and rationales for the right strategy.

These strategies all use the `cfgset` command. Use `cfgset update -h` for information about the update subcommand. See *Config Set Create/Update Process* on page 116 for an outline of the process followed by `cfgset` each time the `create` or `update` subcommand is used.

Update a Config Set Interactively

Prerequisites

This procedure assumes an existing config set needs to be updated.

About this task

Use this procedure when one or more config sets require a few changes (e.g., cloned config sets that need to be adjusted for a particular purpose), or to confirm that all required and basic settings have been set (very useful!). To update just one service in a config set, see *Update a Single Service in a Config Set* on page 115.

`cfgset` has two modes that initiate an interactive configurator session: `auto` (default) and `interactive`. This procedure shows examples of updating config set `p0` of type `cle` interactively in either mode. For additional examples, use `cfgset update -h`.

Note that the `cfgset` command is run as root.

Procedure

Invoke the configurator in auto mode (default) or interactive mode.

- **Interactive mode.**

  To display a menu of services in config set `p0` that have configuration settings with state `unset` (default) and level `basic` (default):

  ```
  smw# cfgset update --mode interactive p0
  ```

  To display a menu of services in config set `p0` that have configuration settings with level `required` and state `unset`:

  ```
  smw# cfgset update --mode interactive --level required p0
  ```

  To display a menu of all services in config set `p0`, use the broadest state and level filters:

  ```
  smw# cfgset update --mode interactive --state all --level advanced p0
  ```

- **Auto mode.**

  To confirm that all required and basic settings have been set (in which case, the configurator will not initiate an interactive session) or to be presented with all settings with state `unset` (default) and level `basic` (default) in all services in config set `p0`:

  ```
  smw# cfgset update p0
  ```

  For a discussion of common outcomes of this command, see *cfgset Troubleshooting Tips* on page 124.

  To be presented with all settings in config set `p0`, use the broadest state and level filters:

  ```
  smw# cfgset update --state all --level advanced p0
  ```

Update a Config Set from Configuration Worksheets

Prerequisites

This procedure assumes an existing config set needs to be updated.

About this task

Use this procedure when performing system upgrades and updates, or any time there is a large amount of configuration data to change. The configurator overwrites all data in a service with the contents of the worksheets specified on the command line. If a worksheet with stale data is used to update the config set, data loss may occur. To ensure that the worksheets used to update the config set are as up-to-date as possible, use this process:

1. Generate worksheets from the current config set.
2. Copy the worksheets to a new location on the management node.
3. Edit the worksheets.
4. Import the worksheets to the current config set.

The detailed steps of this procedure show an example of how to update config set `p0` of type `cle` (default) from configuration worksheets. To update just one service in a config set, see *Update a Single Service in a Config Set* on page 115.

Note that the `cfgset` command is run as root.

Procedure

1. Generate new worksheets from configuration service packages installed on the system and config set `p0`.

   ```
   smw# cfgset update --mode prepare p0
   ```

2. Locate the newly generated worksheets and copy them to a new location on the management node.

   ```
   smw# cfgset show --fields path p0
   p0:
     path: /var/opt/cray/imps/config/sets/p0

   smw# cp /var/opt/cray/imps/config/sets/p0/worksheets/* /some/edit/location
   ```

3. Edit one or more worksheets to make the needed changes.

   To edit the worksheets, open those with settings that need to be changed and make changes, as needed. Each worksheet is a YAML file that contains instructions on how to edit it.

4. Import the completed worksheets to `p0` using `cfgset update`.

   ```
   smw# cfgset update --worksheet-path '/some/edit/location/*_worksheet.yaml' p0
   ```

   The argument to the `--worksheet-path` option is a file glob to allow multiple worksheets to be imported in a single create/update operation. Full paths to single worksheets can also be used. The configurator will replace config set data with imported worksheet data only for services that have matching worksheets provided on the command line.

   REMEMBER: When using `cfgset` with the `--worksheet-path` option to import worksheets,

   - Always add single quote marks around the worksheet path.
   - Do not add mode, state, level, or service options; the configurator ignores them for worksheet import.
   - The type of the config set must match the type of the worksheets being imported.

Update a Config Set without Callbacks

Prerequisites

This procedure assumes an existing config set needs to be updated.

About this task

Pre- and post-configuration callback scripts may invoke utilities that query hardware in order to provide additional config set content. Use this procedure when no hardware is attached to the XC system, as in some testing scenarios. This procedure shows an example of updating config set `p0` of type `cle` interactively while skipping all callback scripts. The `--no-scripts` option can also be used when updating a config set from worksheets.

Note that the `cfgset` command is run as root.

Procedure

Update a config set without callbacks.

```
smw# cfgset update --no-scripts p0
```

CAUTION: Skipping callback script processing invalidates a config set. A config set cannot be considered validated unless it is updated successfully without the `--no-scripts` option. Update all config sets to run the callback scripts before using the config set with the system.

Rename a Config Set

Prerequisites

This procedure assumes an existing config set.

About this task

Use this procedure when a config set needs to be renamed or updated as well as renamed. The renaming operation follows the same basic configurator flow as a regular update but renames the config set prior to other processing. If auto-cloning is enabled, config set backups of the original config set will not be renamed. This procedure shows an example of renaming config set `p0`.

Note that the `cfgset` command is run as root.

Procedure

Rename a config set using the `update` subcommand with the `--rename` option.

```
smw# cfgset update p0 --rename p0.new
```

Note that the config set being operated on (`p0` in this example), does not have to be the last argument on the command line.

Update a Single Service in a Config Set

Prerequisites

This procedure assumes an existing config set.

About this task

Use this procedure when setting up a new service, or when just one service requires modification. This procedure provides examples of updating a single service at a time instead of the entire config set, and it can be done either interactively or using a configuration worksheet.

Procedure

Update a single service in config set `p0`.

- **Update interactively: use the `--service` option.**

  IMPORTANT: For a service with configuration template file `cray_example_config.yaml` and configuration worksheet `cray_example_worksheet.yaml`, use only the `cray_example` portion on the command-line when specifying a single service.

  To display a menu of settings in the `cray_example` service in config set `p0` that are level `required` and any state (default for interactive mode when only one service is specified):

```
smw# cfgset update --service cray_example --mode interactive \
--level required p0
```

To display a menu of all settings (with settings of any state and any level):

```
smw# cfgset update --service cray_example --mode interactive \
--level advanced p0
```

To be presented with all settings (with settings of any state and any level):

```
smw# cfgset update --service cray_example --state all --level advanced p0
```

- **Update with a worksheet: use the `--worksheet-path` option.**

  To update the service using a worksheet, use the `--worksheet-path` option instead of `--service`. Unlike the `--service` option, with the `--worksheet-path` option it is necessary to provide the full path to the worksheet for that service, which includes the `_worksheet.yaml` portion.. The configurator will replace only the config set data that corresponds to the data in the worksheet being imported.

  ```
  smw# cfgset update --worksheet-path \
  /path/to/worksheets/cray_example_worksheet.yaml p0
  ```

Config Set Create/Update Process

Config sets are created and updated using the `cfgset` command with the `create` and `update` subcommands, respectively. Invoking `cfgset` with one of those subcommands initiates the following process, which defines how configuration content is discovered from service packages installed on the management node and used, along with site-supplied content, to create or update a config set.

1. `cfgset` searches for service packages in `/opt/cray/imps_config`.
2. `cfgset` copies to the config set (for `create`) or overwrites in the config set (for `update`) ansible and dist content from each service package. Note that it is only content from service packages that is overwritten; content placed in those directories manually is unchanged.

   NOTE: Manual changes to service package content in this directory will be overwritten!
3. `cfgset` runs pre-configuration callback scripts from each service package. Scripts act on the config set to create content necessary for system configuration, which they place into the `files` subdirectory of the config set.
4. `cfgset` invokes the configurator to do steps 4 through 6.

   Configurator finds configuration templates from each service package that match the config set type, and then copies them into the config set (for `create`) or merges them with the templates already in the config set (for `update`).
5. Configurator takes *one* of these actions to further modify config set template data, depending on the command-line options used:

interacts with user	Initiates an interactive session with the user and modifies config set template data based on the values supplied by the user. Occurs when `--mode interactive` option used or no mode option used, which defaults to `auto` mode.
does not interact with user	Does not initiate an interactive session and does no further modification to config set template data beyond the copy/merge of service package data already done in step 4.

Occurs when `--mode prepare` option used. Note that although this action is associated with preparing worksheets, all three actions result in worksheets being written in step 6.

imports worksheets

Imports configuration worksheets and modifies config set template data based on the values in each service worksheet.

Occurs when `--worksheet-path` *`FILEPATH`* option used.

6. Configurator writes configuration template data, configuration worksheets, and a changelog to the config set. Note that the configurator never modifies the configuration templates in service packages, which are found in `/opt/cray/imps_config/`*`SERVICE PACKAGE`* for each service package.
7. `cfgset` runs post-configuration callback scripts from each service package.
8. `cfgset` autosaves the config set to a time-stamped clone.

The following three figures illustrate how this eight-step process is used to create a CLE config set. They differ in how configuration data in a config set is further modified in step 5, corresponding to the three different actions: interacting with the user (modification through user interaction), not interacting with the user (no further modification), and importing worksheets (modification through bulk import of configuration worksheets). Black lines indicate `cfgset` actions, and red lines indicate actions taken by the configurator when invoked by `cfgset`.

This first figure shows how the configurator creates config set templates (in the `config` subdirectory) from service package templates in step 4, enables the user to enter new or modify existing configuration data in step 5, and then saves the new/modified data to the config set templates and worksheets in step 6.

Figure 16. Process to Create a Config Set Interactively

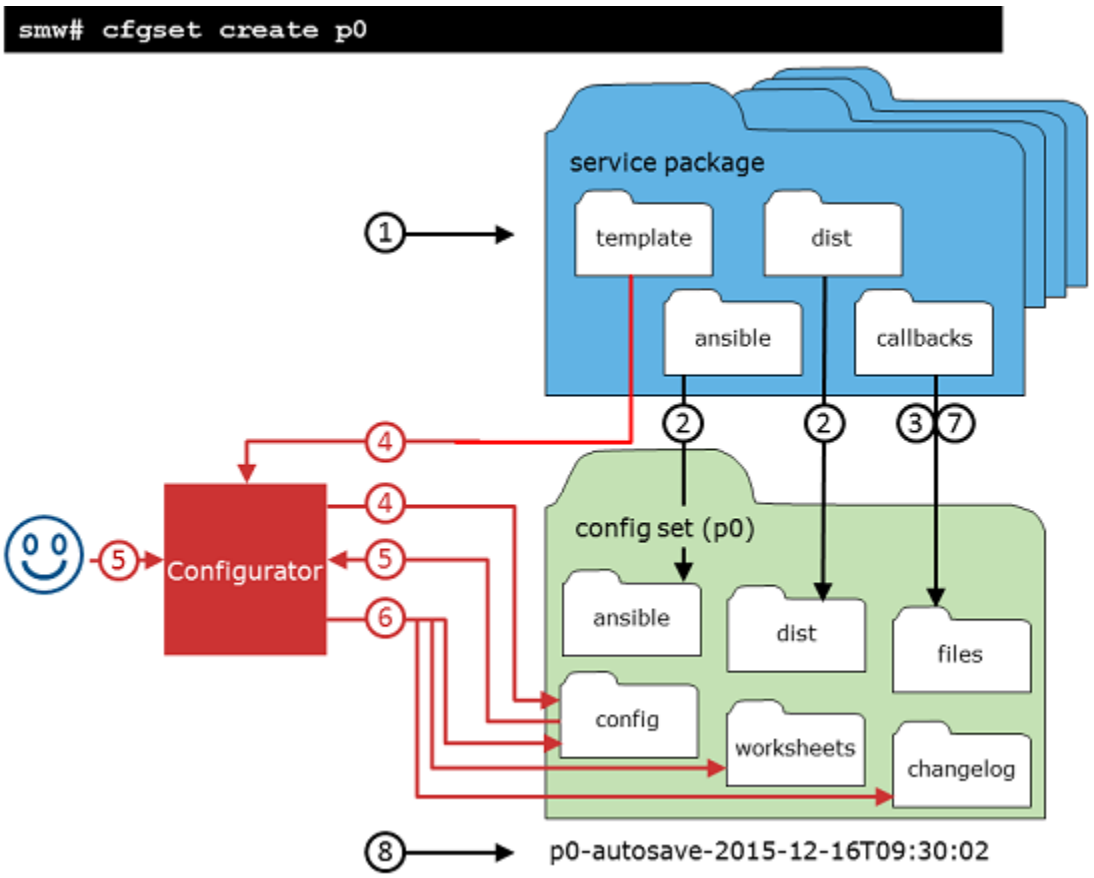
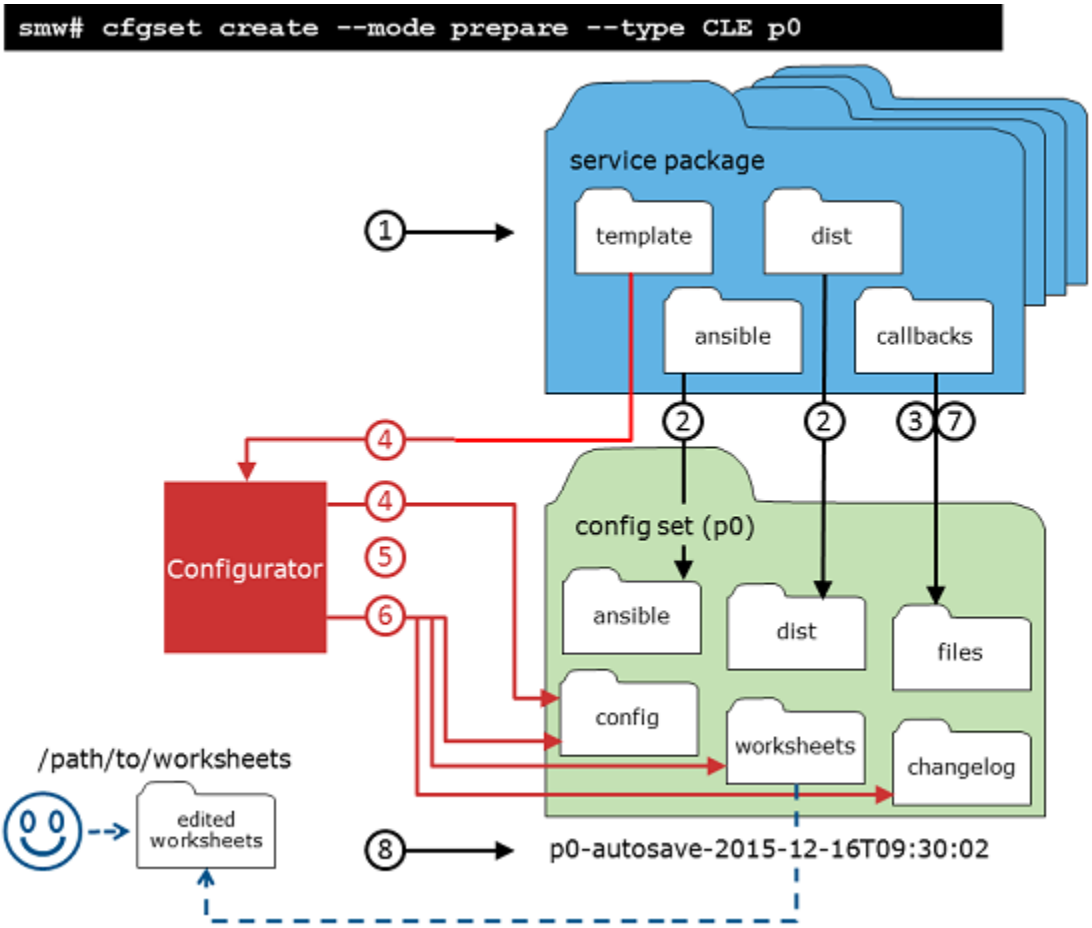
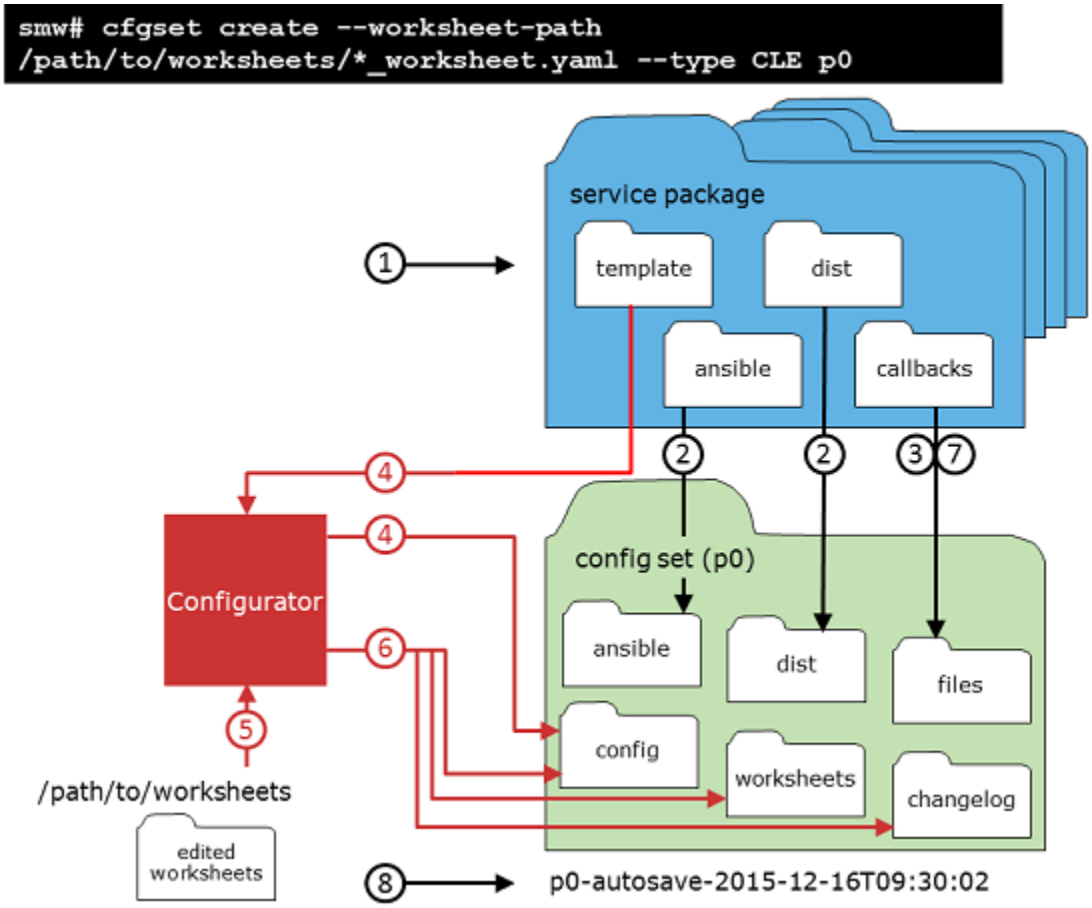


Figure 17. Process to Create a Config Set and Prepare Worksheets



The prepare-mode figure shows how the configurator creates config set templates from service package templates in step 4, does nothing to that configuration data in step 5, and then saves the data from step 4 to config set templates and worksheets in step 6. The blue dashed line indicates an action taken by the user after `cfgset` has completed the create/update process to prepare worksheets. The user (usually an installer or system administrator) copies the worksheets prepared by the configurator to a location outside the config set and edits them (or has other site staff edit them) with site-specific configuration values. It is these edited worksheets that are used when creating (or updating) a config set from worksheets (shown in worksheets figure).

Figure 18. Process to Create a Config Set from Worksheets



The worksheets figure shows how the configurator creates config set templates from service package templates in step 4, imports new or modified configuration data from worksheets in step 5, and then saves the new/modified data to the config set templates and worksheets in step 6.

Tips for Configurator Interactive Sessions

When a user invokes `cfgset` in `auto` or `interactive` mode to create or update a config set, `cfgset` invokes the configurator to initiate an interactive session with the user. The configurator provides command help to aid users in navigating the tool and adding/updating configuration data. These tips supplement that help.

Know the difference between the two "interactive" modes

Interactive mode and auto mode can both result in a configurator interactive session, but their uses and behaviors are quite different.

auto mode	Helpful for verifying that all desired settings have been set. Auto mode initiates an interactive session when there are one or more settings in the config set that meet state and level filtering criteria. Those settings are presented one at a time, and when all have been presented, the configurator exits the session.
interactive mode	Helpful for seeing the "big picture" and having more control over which services/settings are presented for configuration.

Interactive mode always initiates an interactive session. It provides two tiers of menus from which users can select one or more services/settings to drill down and configure just what is needed. The configurator presents the selected settings one at a time, as in auto mode, but when all selected settings have been presented, it returns the user to the menu from which the selection was made.

- *Service Configuration List Menu* (or Service List Menu) lists the services in the config set
- *Service Configuration Menu* (or service menu) lists the settings in a particular service

Filter wisely

Level and state filters determine what the configurator displays to users: what is included in the menu of services/settings for selection in interactive mode, and what setting fields are presented automatically for configuration in auto mode. The filters can be specified on the command line when invoking `cfgset`, and they can be changed in interactive mode. If not specified, they default to level `basic` and state `unset` (exception: for interactive mode, if a single service is specified, the default state is `all`).

In interactive mode, the configurator populates the Service List Menu with only those services that meet state and level filtering criteria; both filters can be switched to different values on this menu screen. In the case of a service menu, the configurator populates it with only those setting fields that meet level filtering criteria (shows all states); level can be switched on this menu screen, but state cannot. Just for fun, cycle through all levels/states, noting how level affects which services appear in the list, while state affects the status displayed for each service.

> **TIP:** If the desired service/setting is not visible in an interactive-mode menu, simply switch level.

In auto mode, the configurator presents only those setting fields that meet state and level filtering criteria. There is no opportunity to switch filter values in auto mode, except by first switching to interactive mode.

> **TIP:** A good way to confirm that all basic settings have been set is to run `cfgset update p0` (where `p0` is the config set name), which defaults to auto mode, level `basic`, and state `unset`. If the configurator does not present any settings, it means that no `basic` or `required` settings are unset.

How to switch states and levels (interactive mode only):

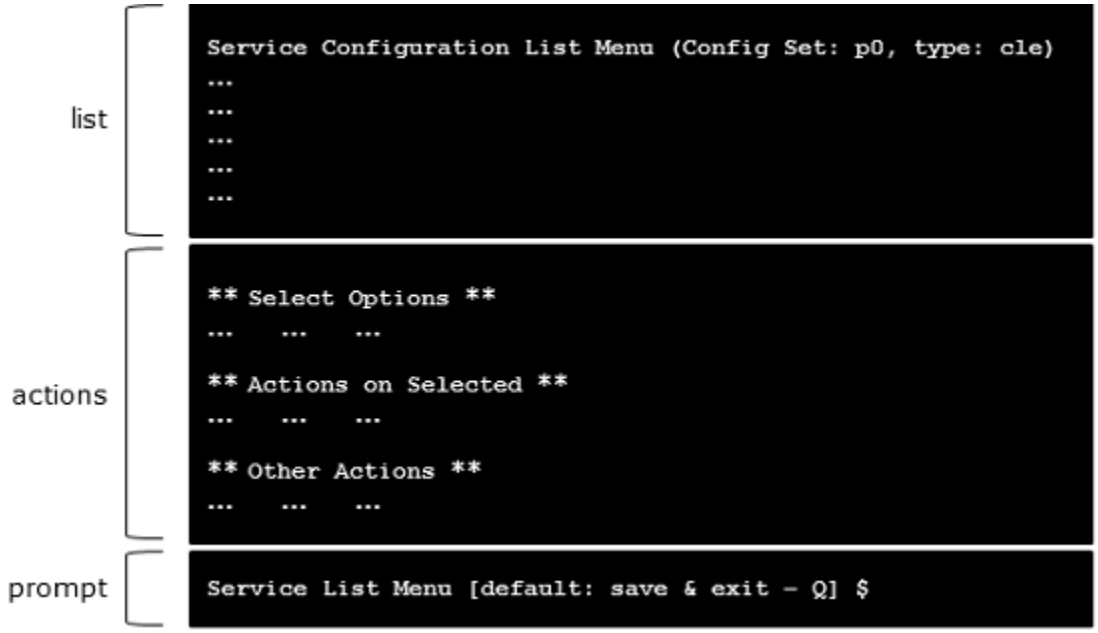
switch states	Enter **`s`** at the configurator prompt to switch from the current state to the next one: `unset` → `set` → `all`. To see all services/settings with the specified level, enter **`s`** until `state=all` displays in the menu header.
switch levels	Enter **`l`** (lowercase L) at the configurator prompt to switch from the current level to the next one: `basic` → `advanced` → `required`. To view all services/settings with the specified state, enter **`l`** until `level=advanced` displays in the menu header.

To see all possible services/settings, switch to `state=all` and `level=advanced`.

Get familiar with menus in interactive mode

The Service List Menu and all service menus have the same three-section layout: a list of services/settings, actions the user can take, and a prompt.

Figure 19. Sections of Interactive-Mode Menus



list The menu name, config set name, and config set type are shown at the top of the list section. This section is helpful for seeing which services still have unconfigured settings (status column—see what changes when state is switched) and for selecting which service(s) to configure or reconfigure.

In a service menu, the list items are configuration settings for that particular service, filtered by level only (state is set to `all` and cannot be switched). This list is helpful for seeing the current state and value of the settings and for selecting which setting(s) to set or change.

actions These three submenus show all commands currently available. Always use an action from the **Select Options** submenu before using any from the **Actions on Selected** submenu. Items in the **Other Actions** submenu can be used at any time (with the obvious exceptions of the exit commands **Q** and **x**, because when one of those is used, the configurator exits the interactive session).

Select Options	Actions that select one or more services/settings from the list. The selected services/settings are the only ones that can be acted upon. Once selected, an asterisk appears in the **Selected** column next to the item and its font color changes.
Actions on Selected	Actions that can be used on the selected service(s) or setting(s); a selection must be made first. Shows in parentheses how many items have been selected. A few of these actions, like toggle whether a service is enabled and toggle whether it inherits setting values from the global version of its template (applies to only a few services) move to the **Other Actions** submenu on service menu screens.
Other Actions	Actions that can be used on all services/settings or on the current configurator session. The most commonly used are the filter switches and help (`?`).

prompt The prompt shows which menu is active and what the default action is. Before a selection is made, the default action is to save and exit (as shown in previous figure). When a selection is made, the default action is to configure the selected service(s) or setting(s), and the prompt changes to

```
MENU_NAME [default: configure – C] $
```

Note that accepting this default action (or entering **c**) displays the configuration setting screen for the first selected setting.

Get familiar with configuration setting screens

A configuration setting screen shows users information about the setting field to be configured (default/current values, data type, level, current state, etc.) and enables the user to navigate among setting fields, enter/change field values, and switch to interactive mode. The configuration setting screen is displayed when a user makes a selection and enters **c** in interactive mode, or when a setting matches state and level filters in auto mode. Configuration setting screens have a prompt that is packed with useful information. Consider this example of a prompt:

```
cray_lmt.settings.lmt_database.data.database_fstype
[<cr>=set 'ext3', <new value>, ?=help, @=less] $
```

The first line is the full name of the setting field being presented (this is the same as the corresponding entry in the configuration worksheet for this service). The part that precedes `.settings.` is the service name (`cray_lmt`, the Lustre Monitoring Tool service, in the example), and the part that follows is the setting field being presented. In the example, the setting is `lmt_database` and the field to be set (one of several for that setting) is `database_fstype`.

The second line lists available commands. In the example, the default command (selected by pressing **Enter** or <cr>) sets the value to `ext3`, which is the default value provided in the configuration template for that service. If this setting field had already been configured with the value `ext3`, the default command would be `<cr>=keep 'ext3',` (`set` becomes `keep`). This list of available commands is not exhaustive: to see all possible options, enter **?** after the prompt, which will insert a context-sensitive menu of commands between the information section and the prompt.

Switch to interactive mode, as needed

When in a configuration setting screen, whether the user has arrived there by invoking `cfgset` in auto mode or by making a selection and entering **c** in interactive mode, it is possible to switch to interactive mode and display either the service menu (lists settings for a single service) or the Service List Menu (lists services in the config set).

switch from setting screen to a service menu

To switch to interactive mode and display the service menu, enter **^** at the configurator prompt. Example:

```
cray_node_health.enabled
[<cr>=keep 'true', <new value>, ?=help, @=less] $ ^
```

switch from setting screen to Service List Menu

To switch to interactive mode and display the Service List Menu, enter **^^** at the configurator prompt. This action can be taken only if `cfgset` was invoked for all services (as this is the default, this is true unless the `--service` or `-s` option was used). Example:

```
cray_node_health.enabled
[<cr>=keep 'true', <new value>, ?=help, @=less] $ ^^
```

Switch between menus in interactive mode, as needed

switch from Service List Menu to service menu	When a service has been selected from the Service List Menu in interactive mode, enter **v** (view settings) to switch to the selected service's menu instead of taking the default action of Configure (**c**). The **v** action is available if only a single service is selected. If multiple services are selected, **c** is the only action available. Example: `Service List Menu [default: configure - C] $ v`
switch from service menu to Service List Menu	To switch from a service menu to the Service List Menu, enter **^^** at the configurator prompt. This action can be taken only if `cfgset` was invoked for all services (as this is the default, this is true unless the `--service` or `-s` option was used). Example: `Node Health Service Menu [default: save & exit - Q] $ ^^`

When in doubt, jump out

It is better to leave a setting field unconfigured than set it to an incorrect value or 'none.' If unsure what the value should be or whether that setting field is needed, jump out using one of these methods:

- *Switch to interactive mode, as needed*.
- Skip to the next setting field: enter **>** at the configurator prompt.

Get help early and often

Enter **?** at the configurator prompt at any time to see a list of available commands. In interactive mode, this simply displays a verbose list of the same commands listed in the menu's three action submenus. However, in a configuration setting screen, entering **?** displays a context-sensitive menu of available commands not displayed elsewhere. Here is an example of the commands available in the context of configuring a multival setting in a service (multival settings are configured by adding/changing entries). Use the **?** command in configuration setting screens early and often to learn the available commands.

```
|--- Command Help
|   *     ++     - double view limit (currently 2)
|   *     --     - decrease view limit by half (currently 2)
|   *     *      - view all entries (no limit)
|   *     +      - add entries
|   *     <#>*   - change the <#> entry. Example: '2b*' selects sub-item b in entry 2
to change
|   *     <#>-   - delete the <#> entry. Example: '4-' deletes entry 4
|   *     d      - delete all entries in the list
|   *     <cr>   - accept the current value(s)
|   *     #      - set the value to its default
|   *     <      - go back to the previous setting
|   *     >      - skip and go to the next setting
|   *     ^      - Go to the 'cray_dvs' service menu (interactive mode)
|   *     ^^     - Go to the service list menu (interactive mode)
|   *     Q      - write out changes and exit the configurator
|   *     x      - revert all changes and exit the configurator
|   *     r      - refresh the screen
|   *     @      - toggle more/less info
|   *     ?      - show this help
```

cfgset Troubleshooting Tips

Unable to Update a Service in a Config Set

The following command to update *SERVICE* in config set `p0` can result in a variety of outcomes, depending on the level and state of the settings in that service.

```
smw# cfgset update --service SERVICE p0
```

Note that for a service with configuration template file `cray_example_config.yaml` and configuration worksheet `cray_example_worksheet.yaml`, use only the `cray_example` portion on the command-line when specifying a single service.

- **Outcome 1: No configuration settings presented.**

  ```
  INFO - Running pre-configuration scripts
  ...
  INFO - Merging configuration templates and validating schema.
  INFO - Configuration worksheets will be saved to /var/opt/cray/imps/config/sets/
  p0/worksheets
  INFO - Changelog will be written to
         - /var/opt/cray/imps/config/sets/p0/changelog/
  changelog_2015-12-02T16:39:25.yaml
  INFO - Running post-configuration scripts
  ...
  INFO - ConfigSet 'p0' has been updated.
  ```

  The command does not specify mode, level, or state, so defaults are used: `auto` mode, level `basic`, and state `unset`. Therefore, the configurator looks only for required and basic settings that are unset. If it finds none, no interaction with the user is necessary, so it proceeds directly to saving worksheets and logs, and then `cfgset` runs post-configuration activities and exits automatically. **If the intention was to confirm that all required and basic settings have been set, then this is the desired outcome.** However, if the intention was to view all settings and perhaps change a few, use this command instead:

  ```
  smw# cfgset update --service SERVICE --level advanced --mode interactive p0
  ```

- **Outcome 2: Some configuration settings presented, but not the ones that need to be changed.**

  The settings that need to be set/changed are not presented because either they are already set or they are level `advanced`. Try this:

  1. Enter **^** at the configurator prompt to switch to `interactive` mode. Now settings of all states are displayed in the service menu and can be selected and set/changed. If the desired settings are still not found in the service menu, continue to the next step.
  2. Enter **l** (lowercase L) at the configurator prompt to switch to the next level (cycles through all three levels) until `level=advanced` displays in the service menu header. Now settings of all levels and states are displayed in the service menu and can be selected and set/changed.

- **Outcome 3: Some new and unfamiliar configuration settings presented.**

  If the service package that contains the service being updated has been reinstalled, the associated service configuration template may have new or revised settings and values. The configurator will find that template in `/opt/cray/imps_config/`*SERVICE_PACKAGE*`/default/configurator/template` and merge its contents with configuration data already in the config set. When the configurator presents those new settings

to the user, they may appear unfamiliar. If settings other than the ones presented need to be set/changed, see Outcome 2.

About Simple Sync

The Cray Simple Sync service (cray_simple_sync) provides a simple, easy-to-use, generic mechanism for administrators to make configuration changes to their system without resorting to writing a custom Ansible play. When enabled, the service automatically copies files found in source directories in the config set on the SMW to one or more target nodes. Simple Sync is a simple tool and not intended as the sole solution for making configuration changes to the system. Writing custom Ansible plays might provide better maintainability, flexibility and scalability in the long term.

The Simple Sync service is enabled by default and has no additional configuration options. It can be enabled or disabled during the initial installation using worksheets or with the `cfgset` command at any time.

```
smw# cfgset update --service cray_simple_sync --mode interactive <config_set_name>
```

For more information, see man cfgset(8).

How Simple Sync Works

When enabled, Simple Sync is executed on all CLE nodes at boot time and whenever the site administrator executes `/etc/init.d/cray-ansible start` on a CLE node. When Simple Sync is executed, files placed in the following directory structure are copied onto nodes that match these criteria:

`smw:/var/opt/cray/imps/config/sets/<config_set>/files/simple_sync/`

./common/files/	Matches all nodes.
./hardwareid/<hardwareid>/files/	Matches a specific node with that hardware ID, which is the cname of a CLE node or the output of the `hostid` command (e.g., 1eac0b0c) on other nodes. An admin must create both the `<hardwareid>` directory and the `files` directory.
./hostname/<hostname>/files/	Matches a node with the specified hostname. An admin must create both the `<hostname>` directory and the `files` directory.
./nodegroups/<node_group_name>/files/	Matches all nodes in the specified node group. The directories for this `nodegroups` directory are automatically stubbed out when the config set is updated after node groups are defined in configured in the cray_node_groups service.
./platform/[compute, service]/files/	Matches all compute nodes or all service nodes, depending on whether they are placed in `platform/compute/files` or `platform/service/files`. Each time the config set is updated, the HSS data store is queried to update which nodes are service and which are compute.
./README	Provides brief guidance on using Simple Sync and a list of existing node groups in the order in which files will be copied. This ordering enables an administrator to predict

behavior in cases where a file may be duplicated within the Simple Sync directory structure.

Simple Sync copies content into place prior to the standard Linux startup (`systemd`) and before `cray-ansible` runs any other services. As a result, Cray services that make small changes to files will operate on the administrator-provided file. Afterwards, the file will contain both non-conflicting administrator-provided content as well as the changes made by the Cray service. Because these changes happen prior to Linux startup, the changes will be in place when the services start up.

Note that there are some config files that are entirely managed by Cray services. Where possible, such files have a comment at the top indicating that the file is completely under the management of the Cray service. Files that have been changed by Cray services can be identified by checking the change logs on the running node in `/var/opt/cray/log/ansible`. Simple Sync does not provide a mechanism to override changes made by Cray services. To override changes made by Cray services, refer to the documentation for the specific service.

The ownership and permissions of copied directories and files are preserved when they are copied to `/` on the matching target nodes. An administrator can run cray_ansible multiple times, as needed, and only the files that have changed will be copied to the target nodes.

Because of the way it works, Simple Sync can be used to configure services that have configuration parameters not currently supported by configuration templates and worksheets. An administrator can create a configuration file with the necessary settings and values, place it in the Simple Sync directory structure, and it will be distributed and applied to the specified node(s).

Characteristics of Simple Sync

Simple Sync is:	**Simple Sync is NOT:**
for simple and straightforward use cases	a comprehensive system management solution
for copying a moderate number of moderately sized files	intended to transfer large objects or a large volume of files
	an interface to configure Cray "turnkey" services such as ALPS, Node Health or Lightweight Log Manager (LLM)

Simple Sync was introduced with the CLE 6.0.UP00 / SMW 8.0.UP00 release. For this release, Simple Sync (Simple Sync v2) has been enhanced to:

- run as early in the Ansible execution sequence as possible (it now runs BEFORE other cray-ansible plays, so it can be used to make changes to files that Cray updates, like `sshd_config`)
- run during the Netroot setup sequence so it can be used to change LNet and DVS settings, if needed
- support Node Groups for targeting which system nodes to copy files to (the Node Groups feature is not yet complete, but it can be used with Simple Sync—see *About Node Groups* on page 130)

Simple Sync v2 still does not support:

- removing files
- appending to files
- changing file ownership and permissions (the permissions of the file in the config set are mirrored on-node)
- backing up files

- overriding Cray-set values (it cannot be used to change files that Cray completely overwrites, such as `alps.conf`, or change values in files that Cray modifies such as PermitRootLogin in /etc/ssh/sshd_config)

Note that the original Simple Sync root directory was different: `smw:/var/opt/cray/imps/config/sets/<config_set>/files/roles/simple_sync`. This means that users of the original Simple Sync need to migrate their files from the old location to the new. For help with that, see *Migrate Content from Simple Sync to Simple Sync v2*.

Using Simple Sync with eLogin

Sites that want to use Simple Sync on eLogin nodes should note the following issues:

- eLogin uses the original Simple Sync in CLE 6.0UP01, not Simple Sync v2. This means content destined for the eLogin nodes must be placed in the old Simple Sync directory structure.
- Simple Sync v1 runs late in the sequence, overwriting content put into place by Cray services.
- Simple Sync v1 does not support node groups.

Cautions about the Use of Simple Sync

- Simple Sync copies files from the config set, which in the case of nodes without a persistent root file-system is cached in a compressed form, locally, in memory. As a result, each file stored in the config set uses some memory on the node. Therefore, using Simple Sync to copy binary files or large numbers of files is inadvisable.
- Be aware of differences in node environments when using Simple Sync. For example, systems configured with direct-attached Lustre (DAL) have nodes running CentOS instead of SLES. Administrators would have to be very careful to avoid putting an inappropriate configuration file into place when using the Simple Sync platform/service target in such a situation.
- Storage and distribution of verbatim config files through Simple Sync creates the potential for unintentional impact to the system when config files evolve due to software changes. Making minimal necessary changes through a site-local Ansible playbook provides more flexibility and minimizes the potential for unintended consequences.

Use Cases

Copy a non-conflicting file to all nodes

1. Place `etc/myfile` under `./common/files/` in the Simple Sync directory structure.
2. Simple Sync copies it to `/etc/myfile` on all nodes.

Copy a non-conflicting file to a service node

1. Place `etc/servicefile` under `./platform/service/files/` in the Simple Sync directory structure.
2. Simple Sync copies it to `/etc/servicefile` on all service nodes.

Copy a non-conflicting file to a compute node

1. Place `etc/computefile` under `./platform/compute/files/` in the Simple Sync directory structure.
2. Simple Sync copies it to `/etc/computefile` on all compute nodes.

Copy a non-conflicting file to a specific node

1. Place `etc/mynode` under `./hostname/c0-0c0s0n0/files/` in the Simple Sync directory structure.
2. Simple Sync copies it to `/etc/mynode` on c0-0c0s0n0.

Copy a non-conflicting file to a user-defined collection of nodes

1. Create a node group called "my_nodes" containing a list of nodes.
2. Update the config set.

```
smw# cfgset update p0
```

3. Place `etc/mynodes` under `./nodegroups/my_nodes/files/` in the Simple Sync directory structure.
4. Simple Sync copies it to `/etc/mynodes` on all nodes listed in node group my_nodes.

Copy to a node a file that has Cray-maintained content

1. Place a version of `sshd_config` with the value “PermitEmptyPasswords yes” under `./nodegroups/login/files/etc/ssh/` in the Simple Sync directory structure.
2. The booted system will contain both:
   - “PermitEmptyPasswords yes” (from the file copied by Simple Sync)
   - “PasswordAuthentication yes” (from modification of file by Cray)

Copy to a node a file that is exclusively maintained by Cray

Files exclusively maintained by Cray such as `alps.conf` cannot be updated using Simple Sync. Please refer to the owning service (such as ALPS) for information on how to update the contents.

Copy to a node a file that resides on a file system that will be mounted during Linux boot

No special operational changes are necessary. However, Simple Sync will put the file in place early in the boot sequence, and then it will be over-mounted by the file system. Because Simple Sync runs again later, it will copy the file into the mounted file system. Due to the ordering of operations, the file will not be present between the time the file system was mounted until the late execution of Ansible.

On Netroot login nodes, modify an LNet modprobe parameter

1. Generate a file `zz_lnet.conf` containing `options lnet router_ping_timeout=100`.
2. Place `zz_lnet.conf` under `./nodegroups/login/files/etc/modprobe.d/` in the Simple Sync directory structure.
3. The `lnet router_ping_timeout` value will be 100.

Note that normally Simple Sync does not allow the user to override Cray values, but this procedure takes advantage of the standard Linux mechanism to override Kernel module options.

Copy a file with an incompatible content to a node file that has Cray-maintained content

While Simple Sync allows an administrator to make changes to the same configuration files as modified by Cray, be very careful to avoid introducing syntax errors or incompatible values that may cause the system to fail to operate correctly.

Configure Files for Cray Simple Sync Service

About this task

Cray Simple Sync provides a generic mechanism to automatically distribute files to targeted locations on the system. When enabled, the Simple Sync service is executed on all CLE nodes at boot time and whenever the administrator executes `/etc/init.d/cray-ansible start` on a CLE node. When Simple Sync is executed, files placed in the following directory structure are copied to the root file system (`/`) on the target nodes. Create whatever directory structure is needed to place the target file(s) in the proper location.

The root directory for Simple Sync v2 is as follows:
`smw:/var/opt/cray/imps/config/sets/<config_set>/files/simple_sync/`

./common/files/	# copies to all nodes
./platform/[compute, service]/files/	# copies to all compute or service nodes
./hardwareid/<hardwareid>/files/	# copies to nodes with matching hardware id
./hostname/<hostname>/files/	# copies to nodes with matching hostname
./nodegroups/<node_group_name>/files/	# copies to members of <node_group_name>
./README	

Any directory structure and files below `./files/` in the Simple Sync directory structure on the SMW is replicated on the target node starting at `/`. For example, `smw:/var/opt/cray/imps/config/sets/<config_set>/files/simple_sync/common/files/etc/myapplicat` will be placed on all nodes as `/etc/myapplication.conf`.

The ownership and permissions of files in the config set is preserved in the copies made to nodes. For more information and use cases, see *About Simple Sync* on page 125.

About the Node Image Mapping Service (NIMS)

The Node Image Mapping Service (NIMS) maps a node to *boot attributes*, which are used when the node is booted.

The primary NIMS component is the daemon, `nimsd`. Interact with `nimsd` either by sending a Hardware Supervisory System (HSS) event or by using the NIMS command line interface (CLI). The HSS Boot Manager daemon communicates with `nimsd` via HSS events. All other interactions with `nimsd` take place through the CLI.

The `nimsd` daemon provides these boot attributes to Boot Manager upon request. Boot Manager uses the boot attributes when it boots or reboots nodes. Boot Manager also provides the boot attributes to the `xtcli` command.

Two conceptual components, nodes and maps, are affected by `nimsd`. A node represents a physical, bootable node on the mainframe. A map is a collection of nodes, typically all the nodes in a partition, or for a non-partitioned system, all the nodes in the entire mainframe.

There can be multiple NIMS maps. However, only one map can be active at a time. The reason to have multiple maps is to differentiate the boot attributes. For example, one map may be a test map to allow booting nodes with a test boot image or a test Config Set.

About Node Groups

The Cray Node Groups service (cray_node_groups) enables administrators to define and manage logical groupings of system nodes. Nodes can be grouped arbitrarily, though typically they are grouped by software functionality or hardware characteristics, such as login, compute, service, DVS servers, and RSIP servers.

Sites are encouraged to define their own node groups and specify their members. Administrators can define and manage node groups using any of these methods:

- Edit and upload the node groups configuration worksheet (`cray_node_groups_worksheet.yaml`).
- Use the `cfgset` command to view and modify node groups interactively with the configurator.
- Edit the node groups configuration template (`cray_node_groups_config.yaml`) directly.

After using any of these methods, remember to validate the config set.

Characteristics of Node Groups

- Node group membership is not exclusive, that is, a node may be a member of more than one node group.
- Node group membership is specified as a list of cnames. However, if the SMW is part of a node group, it is specified with the output of the `hostid` command.
- All compute nodes and/or all service nodes can be added as node group members by including the keywords “platform:compute” and/or “platform:service” in a node group.
- The Configuration Management Framework (CMF) exposes node group membership of the current node through the local system "facts" provided by the Ansible runtime environment. This means that each node knows what node groups it belongs to, and that knowledge can be used in Cray and site-local Ansible playbooks.

Admin Use Cases

The use cases that follow have been gathered from Cray developers and system administrators and some customer sites to provide examples of tasks that are done differently within the new Cray system management model. For XC-40 systems with release CLE 6.0 and later, Cray uses Ansible to orchestrate the boot sequence and configuration. Configuration content is centralized in a config set located on the SMW. Within a config set, Cray provides a drop zone for customers to place Ansible plays and other content referenced by those plays. All content within the config set is accessible by every CLE node on the system, which is how configuration information is distributed throughout the system.

The example Ansible plays included in many of these use cases contain the following three elements:

hosts Specifies where the play will run. In Cray systems, this is typically set to `localhost`, because unlike many configuration management tools that push information out to nodes from a centralized location, Ansible (as used by Cray in this release) pulls information to the local node and runs all plays there.

vars Defines variables scoped to the play and all other plays that come after it. Ansible provides access to facts about the system—network interfaces, disks, operating system version, and so forth—for use within each Ansible play ("built-in" facts). In addition, Cray provides access to facts that are Cray-specific, such as nid name, cname, node type (smw, sdb, boot, etc.), for use within each Ansible play. The facts are all accessible without having to define these variables; however it is good practice to define variables using the provided facts because they can be assigned shorter names and can be set to useful boolean values.

To view all available built-in facts, run this command on the node where the Ansible plays will run:

```
smw# ansible <hostname> -m setup
```

To view Cray-supplied facts, run this Python script:

```
smw# /etc/ansible/facts.d/cray_system.fact
```

tasks Each Ansible play `task` is like a line in a Bash script. Each task must have a name, a directive or module, and a conditional (a `when` clause), which indicates the conditions under which the play should execute that task on the node. See the Module Index in the *Ansible Documentation* website for a description of all Ansible modules and their arguments. Look for modules for Ansible 1.9.2 and earlier versions.

There are many ways to accomplish the same thing in the new system management model. For example, in some of the use cases that follow, a site could choose to keep using a favorite script within the Ansible framework, convert it to an Ansible play, or use it outside of the framework.

Use Case: boot.last Script

About this task

Many Cray customer sites used to run a `boot.last` script, or something like it, to start up and manage additional services, configurations, and settings, such as tuning Lustre, starting a secondary `sshd` for a customer network, starting a workload manager, or setting up service nodes to talk to other service nodes using `ssh`. It would run last on each service node as it booted. Its value lay partly in enabling sites to specialize nodes and/or classes in a scalable way.

This procedure shows two ways to accomplish the same thing using Ansible.

Procedure

1. Choose how to accomplish the purpose of the original `boot.last` script.

   - Option 1: Write an Ansible play that uses Ansible modules (directives, a bit like function calls) to do individual steps equivalent to the lines in the `boot.last` script.
   - Option 2: Write an Ansible play that simply uses the `shell` directive (an Ansible module) to run the original `boot.last` script.
   - Option 3: Run the `boot.last` script outside the Ansible framework, after the system nodes have finished booting.

   Options 1 and 2 would be executed when cray-ansible, which is a wrapper around Ansible, gathers all Ansible plays into a master playbook and then runs that playbook. Option 3 would occur after the system has booted. Because Option 3 does not use the Ansible framework, it is not described further in this procedure.

2. Write the Ansible play.

   In the example code below, the variable `run_after` is a Cray-provided way to specify the order in which a play is executed; it is set to a list of Ansible plays that this play should follow. In this example, it is set to `simple_sync`, a play that runs at the end of the boot cycle, which is when the `boot.last` play should run.

   The first task in this example corresponds to Option 1. In the example, the task is to start a service named `awesomed`. There could be many other tasks that a site would want to include in a `boot.last` play. The second task in this example corresponds to Option 2.

```
    - hosts: localhost
      vars:       # Cray-provided node "facts" + config set data
        nid:      ansible_local.cray_system.nid
        is_nid7:  ansible_local.cray_system.nid == "7"
        is_login: ansible_local.cray_system.hostid in
cray_login.settings.login_nodes.data.members
        is_sdb:   "sdb" in ansible_local.cray_system.roles
        in_init:  ansible_local.cray_system.in_init
        is_svc:   ansible_local.cray_system.platform == "service"


        run_after:       # e.g., call out a runtime dependency, a Cray-ism
        - simple_sync

      tasks:
      # Option 1: Use Ansible modules to do individual steps (e.g., start a
service)
      - name: start awesomed service on nid0007, sdb, login nodes
        service: name=awesomed state=started args="-f /path/to/
awesome_config.conf"
        when:
          (is_nid7 or is_login or is_sdb) and not in_init

      # Option 2: Just do everything in my script
      - name: run my script on all service nodes
        shell: /etc/opt/cray/config/current/dist/site_script.sh >> somelog.txt
        when:
          is_svc and not in_init
```

3. Drop the Ansible play and any supporting content into the config set.

/var/opt/cray/imps/config/sets/p0/ansible/	Location in config set p0 for site Ansible plays, like this new `boot.last.yaml`.
/var/opt/cray/imps/config/sets/p0/dist/	Location in config set p0 for content that supports or is used by site Ansible plays. If using Option 2, drop the `boot.last` script here.

4. Run the new Ansible play manually to test it.

```
smw# /etc/init.d/cray-ansible boot.last.yaml
```

This Ansible play will be distributed to all nodes. When the system boots, this play will run on all nodes, and the conditional (`when`) clauses will determine whether a particular task will execute on any given node. This play will run after the `simple_sync` play, and both of the tasks will execute only during the second phase of running Ansible plays, which occurs after Linux finishes booting. The first phase, `init`, occurs prior to Linux booting.

Use Case: Change a File on a Compute Node

About this task

System administrators sometimes need to change files such as modprobe.conf, fstab, and nodehealth.conf on compute nodes. For example, to tune DataWarp or Lustre, the modprobe.conf file might need to be changed. Cray provides configuration templates and Ansible plays for most Cray services (e.g., `cray_net`, `cray_rsip`, `cray_node_health`, and `cray_dvs`), which generate or change such files automatically as part of the boot process or after reconfiguring a service. If no Cray-provided play exists to make the needed changes or an existing play does not cover a needed use case, administrators can change these files directly.

This procedure shows three approaches to changing a file on a compute node. Cray recommends the third approach, which is also applicable to changing files on any node, not just compute nodes.

Procedure

1. Choose an approach:
   - Option 1: After building an image, chroot into the node and put the file there (or merge it with an existing file).
   - Option 2: Use the Cray-provided Simple Sync service.
   - Option 3: (recommended) Write an Ansible play that changes the file directly or runs a script to change the file.

Option	Pros	Cons
Option 1 (chroot)	• works well for static files • easy to do: just copy/edit a file • done on the SMW	a maintenance headache: • if done manually, would need to be done for lots of compute images after each image rebuild • if done using Extended Image Recipes, would need to clone and customize the base Cray image recipe with the desired extensions, and repeat that every time a new Cray image recipe became available

Option	Pros	Cons
Option 2 (Simple Sync)	• easy to do: just put a file in a directory and turn on the Simple Sync service • done on the SMW • can specialize targets to a limited set of targets: by class, cname, or hostname (hostnames used for non-Cray platforms that do not have cnames) • works best for providing access during run time to small admin tools (e.g., a widget or script like the shell alps script workaround) and third-party software	• Simple Sync writes the file to the desired place without regard for what may already be there, so without knowledge of what else touches the file (e.g., other Ansible plays), admins risk killing the node • does not scale well • not reusable on a different system
Option 3 (Ansible)	• done on SMW in config set • can specialize the target nodes further than possible with Simple Sync: any grouping of nodes • can choose when the Ansible play is run during the boot sequence • can edit or replace a file programmatically, careful to not clobber something that needs to be there • once a play is set up and tested, easy to maintain • easily scales to large systems • if play written at a high enough level of abstraction, can reuse for different systems (just change the target node list)	• requires some knowledge of the boot process (ordering, timing) • more work up front to set up a play • plays/scripts must be tested

The remaining steps provide instructions for Option 3 only.

2. Write the Ansible play.

The first task in this example corresponds to using the Ansible module `lineinfile` to change a file directly (Option 3a). The second task in this example corresponds to using the Ansible module `shell` to run a script to change a file (Option 3b).

```
    - hosts: localhost
      vars:       # Cray-provided node "facts" + config set data
        in_init:  ansible_local.cray_system.in_init

      tasks:
      # Option 3a: Use Ansible modules to do individual steps (e.g., add a
line to a file)
      - name: add mount to fstab
        lineinfile:
          dest=/etc/fstab
          regexp="^172.30.79.66:/home/users"
```

```
          line="172.30.79.66:/home/users /home/users nfs nfsvers=3,noacl 0 0"
          backup=yes
        when: in_init

      # Option 3b: Just do everything in my script
      - name: run my script on all service nodes
        shell: /etc/opt/cray/config/current/dist/site_script.sh >> somelog.txt
        when: in_init
```

3. Drop the Ansible play and any supporting content into the config set.

/var/opt/cray/imps/config/sets/p0/ansible/	Location in config set p0 for site Ansible plays, like this new `change_file.yaml`.
/var/opt/cray/imps/config/sets/p0/dist/	Location in config set p0 for content that supports or is used by site Ansible plays. If using Option 3b, drop the script here.

4. Run the new Ansible play manually to test it.

```
smw# /etc/init.d/cray-ansible change_file.yaml
```

This Ansible play will be distributed to all nodes. When the system boots, this play will run on all nodes, and the conditional (`when`) clauses will determine whether a particular task will execute on any given node.

Use Case: Install Third-Party Software

About this task

Any software that is created independent from Cray *and* that is not delivered with a Cray system is third-party software that administrators install as add-ons to the Cray system. The information in this section does not pertain to software installed on an external file system that is connected to a Cray system. There are several ways to install third-party software:

- Add a third-party software package to an image recipe.
- Use the `chroot` command to install the software to an existing image.
- Use the `zypper` to install software on a node.

Installing software via an image recipe is the best method to use because the update to the image is persisted in the recipe and each time a node boots from the image, the third-party software is available. Using `chroot` or `zypper` to install software is usually less desirable because the installations are not persisted. However, using `chroot` or `zypper` can be useful when persistence is not important, such as during testing of third-party software. Installations using `zypper` are lost the next time the node is booted. Installations using `chroot` are lost when a node image is rebuilt from a recipe.

To include third-party software in an image recipe:

Procedure

1. Find the image recipe to update with third-party software.

```
smw:~# recipe list
```

The console displays a list of the available recipes.

2. Clone the recipe that needs updating. Create, for example, the *my_recipe* recipe.

```
smw:~#recipe create -clone original_recipe my_recipe
```

3. Add the RPM package of the third-party software, *my_package*, to the recipe using the update subcommand and the `-p` option.

```
smw:~# recipe update my_recipe -p my_package
```

4. Build the new image, `my_image`, for example.

```
smw:~# image create -r my_recipe my_image
```

5. Export the new image.

```
smw:~#image export my_image
```

6. Use the `cnode` to change the node to use the new image.

```
smw:~#cnode update --set-image /path/to/my_image.cpio for --node cname
```

7. Reboot the node to use the new image.

Use Case: Start a Service on Specific Nodes at Boot Time

Using an Ansible play is the best way to start a service on specified. In this use case, the objective is to start `cron` at boot time on login nodes only.

```
---

- name: only run cron on login nodes
  hosts: localhost

  tasks:
  - name: control cron
    service:
      name: cron
      state: stopped
      when:
      ansible_local.cray_system.hostid not in
cray_login.settings.login_nodes.data.members
      and not ansible_local.cray_system.in_init
```

Specify the service to be started (`cron`). Specify what to do to the service (stop, start, or restart). Provide a conditional state to specify the nodes where the action is to be taken. Service is Ansible module. More information about the service module is at: *http://docs.ansible.com/ansible/service_module.html*

Provide a conditional statement to specify the nodes where the action is to be taken. The `cron` service starts on all nodes, so the play causes `cron` to boot in a stopped state on all nodes except the login nodes. So if the node that the play is running on is not a login node, the `cron` is stopped. The `cray_login.settings.login_nodes.data.members` string refers to the listed login nodes in the `/var/opt/cray/imps/config/sets/p0/config/cray_login_config.yaml` file. The

`ansible_local.cray_system.hostid` string is a Cray-provided Ansible fact that is used to specify the local node. To view Cray-supplied facts, run this Python script: `/etc/ansible/facts.d/cray_system.fact`.

A second condition indicates when to run the play. Ansible is run twice during boot: either in `init` before Linux starts or after Linux starts. These runs are called early Ansible and late Ansible. When running plays that control processes, it is usually best to avoid running plays in `init`. To accomplish that, use `not ansible_local.cray_system.in_init`, which is a Cray-supplied fact.

Use Case: Changing root and crayadm Passwords

About this task

Cray system passwords are initially set during system installation by either editing the `cray_local_users_config.yaml` directly or by using the Configurator in interactive mode. The procedure in this section is to make password changes by editing the yaml file directly. All system user passwords can be changed using the method in this section, not just the `root` and `crayadm` passwords.

Procedure

1. Generate an encrypted password using the `mkpasswd` command or an equivalent command. Users will likely do this step themselves and provide the encrypted output to the administrator.

   ```
   smw:~# mkpasswd --method=sha-512
   ```

2. Edit the `/var/opt/cray/imps/config/sets/p0/config/cray_local_users_config.yaml` file. Locate the `users` section of the file. The example code that follows shows the `crayadm` user.

   ```
   users:
       data:
       -   key: crayadm
           uid: '12795'
           group: crayadm
           crypt: $6$[...]y2/
           other_groups: []
           description: default cray administrative user
           shell: /bin/bash
           home: /home/crayadm
           system: true
           deleted: false
           domains:
           - compute
           - login
           - admin
           passwordless_ssh: false
   ```

3. Paste the encrypted new password in the crypt field. The length of the encrypted password in the preceding example is shortened.

4. Run Ansible or reboot the system to propagate passwords.

InfiniBand and OpenFabrics Interconnect Drivers

InfiniBand (IB) and OpenFabrics remote direct memory access (RDMA) is supported on service nodes for Cray systems running the Cray Linux Environment (CLE) operating system.

No separate installation is required. The kernel-space libraries and drivers are built against Cray's kernel. OpenFabrics Enterprise Distribution (OFED™) and InfiniBand RPMs are included in the CLE release and installed by default. However, OFED will not run on the Cray system until the I/O nodes are configured to use IB.

To configure IB and OFED, see the procedures provided in this chapter; to configure IB and OFED during installation or upgrade of the CLE software, see XC™ Series Software Initial Installation and Configuration Guide.

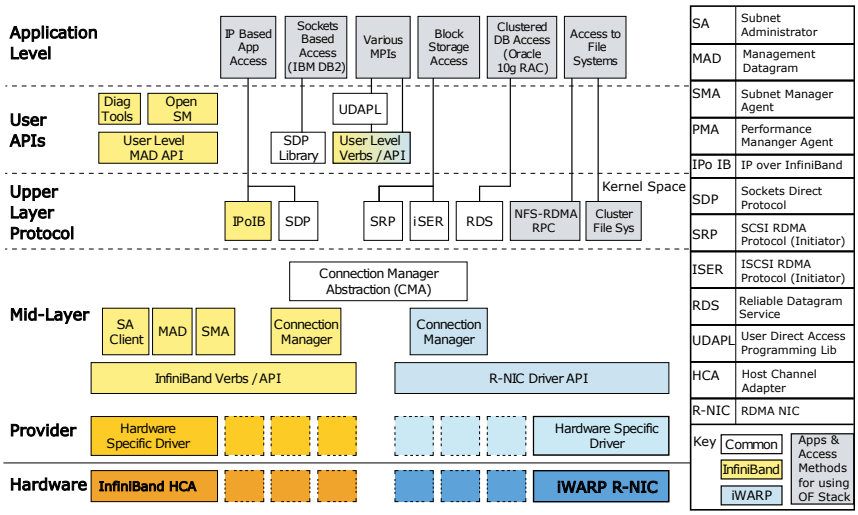
Cray supports InfiniBand as an I/O interconnect. IB enables efficient zero-copy, low-latency RDMA transfers between network peers. As a result, IB gives Cray systems the most efficient transfer mechanism from the high speed network (HSN) to external I/O devices.

CLE includes a subset of the OpenFabrics Enterprise Distribution (OFED) to support the use of InfiniBand on Cray I/O nodes. OFED is the software stack on the host that coordinates user-space and kernel-space access to the IB hardware. IB support is restricted to I/O service nodes that are equipped with PCI Express (PCIe) cards for network connectivity.

IB can be used on Lustre router nodes as a network interconnect between the Cray system and external Lustre servers.

The OFED software stack consists of many different components. These components can be categorized as kernel modules (drivers) and user/system libraries and utilities, commands and daemons for InfiniBand administration, configuration, and diagnostics; Cray maintains the kernel modules so that they are compatible with CLE.

Figure 20. The OFED Stack (source: OpenFabrics Alliance)



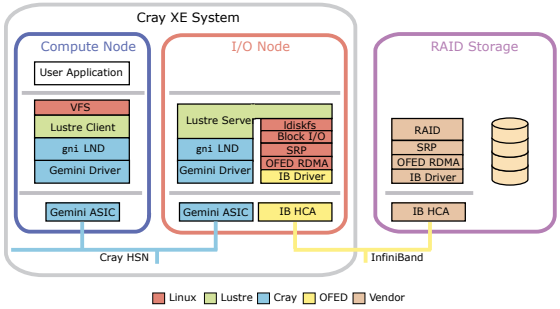
InfiniBand Uses

InfiniBand is a payload-agnostic transport. It can move small messages or large blocks efficiently between network endpoints. The following examples demonstrate how Cray uses InfiniBand and the OFED stack to support block I/O, file I/O, and standard network inter-process communication.

Storage Area Networking

InfiniBand can transport block I/O requests to external storage targets. ANSI T10's SCSI RDMA Protocol (SRP) is currently the only SCSI-transporting protocol supported on Cray systems with InfiniBand. *Cray System Connected to Storage Using SRP* on page 139 shows SRP on InfiniBand connecting the Cray to an external RAID array. The OFED stack is shown in the storage array for clarity; it is provided by the site-specific third party storage vendor.

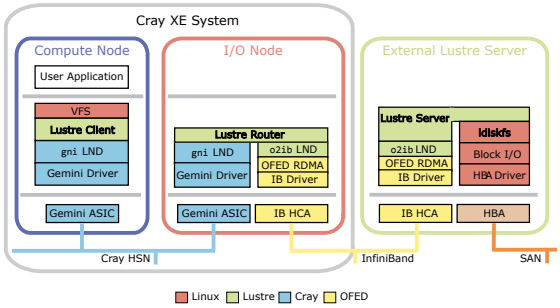
Figure 21. Cray System Connected to Storage Using SRP



Lustre Routing

Cray uses InfiniBand on the service nodes to connect Cray compute nodes to Lustre File System by Cray (CLFS) servers, as shown in *Cray Service Node Acting as an InfiniBand Lustre Router* on page 139. In this configuration, the Cray service node is no longer a Lustre server. Instead, it runs a Lustre router provided by the LNet layer. The router moves LNet messages between the Cray HSN and the external IB network, which transports file-level I/O requests between the clients on the Cray HSN and the servers over the IB fabric. Please speak with a Cray service representative regarding an CLFS solution.

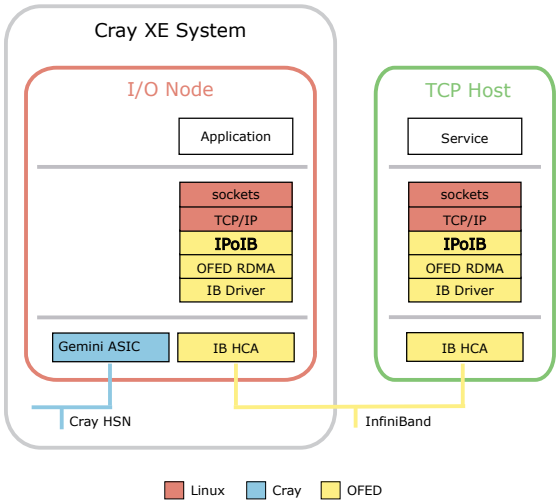
Figure 22. Cray Service Node Acting as an InfiniBand Lustre Router



IP Connectivity

InfiniBand can also carry socket-based inter-process traffic typical of commodity clusters and TCP/IP networking. InfiniBand supports the IP over IB (IPoIB) protocol. Since IB plugs-in below the socket interface, neither the application nor the service needs to be recompiled to communicate over an InfiniBand network. Both protocols are diagrammed on a service node in *Cray Service Node in IP over IB Configuration* on page 140.

Figure 23. Cray Service Node in IP over IB Configuration



Upper Layer InfiniBand I/O Protocols

In addition to the OFED RDMA stack, Cray supports three upper layer protocols (ULPs) on its service nodes as shown in the table. Because all ULPs use the OFED stack, the InfiniBand configuration must be followed for all IB service nodes. It is only necessary to configure the specific ULPs intended for use on the service node.

For example, a Lustre server with an IB direct-attached storage array uses the SCSI RDMA Protocol (SRP), not the LNet Router. On the other hand, if the Lustre servers are external to the Cray system, the service node uses the LNet router instead of SRP. IP over InfiniBand (IPoIB) is used to connect non-RDMA socket applications across the IB network.

Table 2. Upper Layer InfiniBand I/O Protocols for Cray Systems

Upper Layer Protocol	Purpose
IP over IB (IPoIB)	Provides IP connectivity between hosts over IB.
Lustre (OFED LND)	Base driver for Lustre over IB. On service nodes, this protocol enables efficient routing of LNet messages from Lustre clients on the Cray HSN to external IB-connected Lustre servers. The name of the LND is `o2iblnd`.
SCSI RDMA Protocol (SRP)	T10 standard for mapping SCSI over IB and other RDMA fabrics. Supported by DDN and LSI for their IB-based storage controllers.

Subnet Manager (OpenSM) Configuration

InfiniBand fabrics require at least one Subnet Manager (SM) operating on each IB subnet in order to activate its respective IB port connected to the fabric. This is one critical difference between IB fabrics and Ethernet, where simply connecting a cable to an Ethernet port is sufficient to get an active link. Managed IB switches typically include an SM and, therefore, do not require any additional configuration of any of the hosts. Unmanaged IB switches, which are considerably less expensive, do not include a SM and, thus, at least one host connected to the switch must act as a subnet manager. InfiniBand standards also support switchless (point-to-point) connections as long as an SM is installed. An example of this case is when a service node is connected to direct-attached storage through InfiniBand.

The OpenFabrics distribution includes OpenSM, an open-source IB subnet management and subnet administration utility. Either one of the following configuration steps is necessary if no other subnet manager is available on the IB fabric. The subnet manager RPMs are installed in the shared root by running `CLEinstall`. OpenSM can be started from the service node on a single port at boot time or manually from the command line to load multiple instances per host.

Start OpenSM at Boot Time

About this task

Follow this procedure to start a single instance of OpenSM on a service node at boot time.

This procedure assumes that the IB HCA is in node 8.

Procedure

1. Access the service node that will host the instance of OpenSM.

   ```
   boot:~ # xtopview -x /etc/opt/cray/sdb/node_classes -n 8
   ```

2. Specialize `/etc/sysconfig/opensm` for the IB node.

   ```
   node/8:/ # xtspec -n 8 /etc/sysconfig/opensm
   ```

3. Edit `/etc/sysconfig/opensm` to have OpenSM start at boot time

   ```
   # To start OpenSM automatically set ONBOOT=yes
   ONBOOT=yes
   ```

4. Add IB services to the service nodes by using standard Linux mechanisms, such as executing the `chkconfig` command while in the `xtopview` utility or executing `/etc/init.d/opensmd start|stop|restart|status` (which starts or stops the OpenSM service immediately). The `chkconfig` command can be used to ensure that the OpenSM service is started at system boot.

   ```
   node/8:/ # /sbin/chkconfig --force opensmd on
   ```

Monitor the System

Manage Log Files Using CLE and HSS Commands

Boot, diagnostic, and other Hardware Supervisory System (HSS) events are logged on the SMW in the `/var/opt/cray/log` directory, which is created during the installation process. The time-stamped `bootinfo`, `console`, `consumer`, and `netwatch` log files are located in the `/var/opt/cray/log/`*`sessionid`* directory by default.

For example, the HSS `xtbootsys` command starts the `xtconsole` command, which redirects the output to a time-stamped log file, such as `/var/opt/cray/log/p0-20120716t104708/console-20120716`.

The `SMWinstall`, `SMWconfig`, and `SMWinstallCLE` commands create several detailed log files in the `/var/adm/cray/logs` directory. The log files are named using the PID of the `SMWinstall` or the `SMWinstallCLE` command; the exact names are displayed when the command is invoked.

CLE logs are saved on the SMW in `/var/opt/cray/log/`*`sessionid`*.

Controller logs are saved on the SMW in `/var/opt/cray/log/controller/`*`cabinet`*`/`*`controller`*`/messages-`*`yyyymmdd`*, where *`cabinet`* is of the form `c0-0`, `c1-0`, etc.; and *`controller`* is either of the form `c0-0`, `c1-0` for cabinet controllers (CC) or `c0-0c0s0` for blade controllers (BC) .

For more information, see the `intro_llm_logfiles(5)` man page.

Filter the Event Log

The `xtlogfilter` command enables the system administrator to filter the event log for information such as the time a particular event occurred or messages from a particular cabinet.

For more information, see the `xtlogfilter(8)` man page.

Finding information in the event log

For this example, search for all console messages from node `c9-2c0s3n2`:

```
crayadm@smw:~> xtlogfilter -f /var/opt/cray/log/event-yyyymmdd
c9-2c0s3n2
```

Add Entries to Log Files

The system administrator can add entries (e.g., the start or finish of system activities) to the `syslog` with the `logger` command. The entry is then available to anyone who reads the log.

For more information, see the `logger(1)` man page.

Add entries to syslog file

For this example, mark the start of a new system test:

```
login# logger -is "Start of test 4A $(date) "
Start of test 4A Thu Jul 14 16:20:43 CDT 2011
```

The system log shows:

```
Jul 14 16:20:43 nid00003 xx[21332]:
Start of test 4A Thu Jul 13 16:20:43 CDT 2012
```

Examine Log Files

Time-stamped log files of boot, diagnostic and other HSS events are located on the SMW in the `/var/opt/cray/log` directory. The time-stamped `bootinfo`, `console`, `consumer`, and `netwatch` log files are located in the `/var/opt/cray/log/`*`sessionid`* directory by default.

For example, the HSS `xtbootsys` command starts the `xtconsole` command, which redirects the output to a time-stamped log file, such as `/var/opt/cray/log/p0-20120716t104708/console-20120716`.

The `SMWinstall`, `SMWconfig`, and `SMWinstallCLE` commands create several detailed log files in the `/var/adm/cray/logs` directory. The log files are named using the PID of the `SMWinstall` or the `SMWinstallCLE` command; the exact names are displayed when the command is invoked.

Remove Old Log Files

The `xttrim` utility provides a simple and configurable method to automate the compression and deletion of old log files. The `xttrim` utility is intended to be run on the SMW from `cron` and is automatically configured to do this as part of the SMW software installation process. Review the `xttrim.conf` configuration file and ensure that `xttrim` will manage the desired directories and that the compression and deletion times are appropriate.

The `xttrim` utility does not perform any action unless the `--confirm` flag is used (to avoid unintended actions), nor will `xttrim` perform any action on open files. All actions are based on file-modified time.

For additional information, see the `xttrim(8)` and `xttrim.conf(5)` man pages.

Check the Status of System Components

Check the status of the system or a component with the `xtcli status` command on the System Management Workstation (SMW). By default, the `xtcli status` command returns the status of nodes.

The `xtcli status` command has the following form:

```
xtcli status [-n] [-m] [{-t type -a}] node_list
```

Where *`type`* may be: `cc`, `bc`, `cage`, `node`, `aries`, `aries_lcb`, `pdc`, or `qpdc`. The list must have component IDs only and contain no wild cards.

Use the `-m` option to display all nodes that were repurposed by using the `xtcli mark_node` command. (The `xtcli mark_node` command can be used to repurpose a service node to a compute role or to repurpose a compute node to a service role.)

For more information, see the `xtcli(8)` man page.

Show the status of a component

For this example, display all nodes that were repurposed using the `xtcli mark_node` command:

```
crayadm@smw:~> xtcli status -m c0-0c0
Network topology: class 2
Network type: Aries
             Nodeid: Service  Core Arch|  Comp state   [Flags]
   ------------------------------------------------------------------
         c0-0c0s2n0:       -  SB16  X86|         off   [noflags|]
         c0-0c0s3n0: service  SB16  X86|         off   [noflags|]
   ------------------------------------------------------------------
```

This shows that `c0-0c0s2n0` is a service node repurposed as a compute node, and that `c0-0c0s3n0` is a compute node repurposed as a service node.

Check the Status of Compute Processors

Use the `xtprocadmin` command on a service node to check that compute nodes are available after the system is booted.

Use the `xtprocadmin` command on a node to check that compute nodes are available after the system is booted.

Identify nodes in down or admindown state

```
nid00007:~> xtprocadmin | grep down
```

Use the user `xtnodestat` command to display the current allocation and status of each compute processing element and the application that it is running. A simplified text display shows each processing element on the Cray system interconnection network.

Display current allocation and status of each compute processing element and the application that it is running

```
nid00007:~> xtnodestat
Current Allocation Status at Wed Jul 06 13:53:26 2011

     C0-0
  n3 AAaaaaaa
  n2 AAaaaaaa
  n1 Aeeaaaa-
c2n0 Aeeaaaaa
  n3 Acaaaaa-
  n2 cb-aaaa-
  n1 AA-aaaa-
c1n0 Aadaaaa-
  n3 SASaSa--
  n2 SbSaSa--
  n1 SaSaSa--
c0n0 SASaSa--
```

```
    s01234567

Legend:
   nonexistent node                 S  service node
;  free interactive compute node    -  free batch compute node
A  allocated interactive or ccm node ?  suspect compute node
W  waiting or non-running job       X  down compute node
Y  down or admindown service node   Z  admindown compute node

Available compute nodes:          0 interactive,         15 batch

Job ID      User      Size   Age        State        command line
--- ------- --------  -----  ---------  --------  ---------------
a   3772974    user1     48      0h06m       run              app1
b   3773088    user2      2      0h01m       run              app2
c   3749113    user3      2     28h26m       run              app3
d   3773114    user4      1      0h00m       run              app4
e   3773112    user5      4      0h00m       run              app5
```

For more information, see the `xtprocadmin(8)` and `xtnodestat(1)` man pages.

Monitor the System with the System Environmental Data Collector (SEDC)

The System Environment Data Collections (SEDC) manager, `sedc_manager`, monitors the system's health and records the environmental data and status of hardware components such as power supplies, processors, temperature, and fans. SEDC can be set to run at all times or only when a client is listening. The SEDC configuration file provided by Cray has automatic data collection set as the default action.

The SEDC configuration file (`/opt/cray/hss/default/etc/sedc_srv.ini` by default) configures the SEDC server. In this file, the administrator can create sets of different configurations as groups so that the blade and cabinet controller daemons can scan components at different frequencies. The `sedc_manager` sends out the scanning configuration for specific groups to the cabinet and blade controllers and records the incoming data by group.

For information about configuring the SEDC manager, see *XC™ Series System Environment Data Collections (SEDC) Guide*.

Monitor the Health of PCIe Channels

Processors are connected to the high-speed interconnect network (HSN) ASIC through PCIe channels.

The `xtpcimon` command is executed from the System Management Workstation (SMW) and is started and run during the boot process.

Any PCIe-related errors are reported to `stdout`, unless directed to a log file.

`xtpcimon` also displays CLE-originated GHAL-based Advanced Error Reporting (AER) errors for PCIe.

If the optional `/opt/cray/hss/default/etc/xtpcimon.ini` initialization file is present, the `xtpcimon` command uses the settings provided in the file.

For more information, see the `xtpcimon(8)` man page.

Report PCIe-related errors to stdout

```
crayadm@smw:~> xtpcimon
starting
----> connection to event router made
121017 04:57:01  #############  #################   ##################
121017 04:57:01  Node           Category            Description
121017 04:57:01  #############  #################   ##################
Received all responses to request to start monitoring
121017 04:58:01  c0-0c0s7a0n1   CorrectableMemErr   0:0:0 AER Correctable: Non-fatal \
                                                      error (mask bit: 1)
121008 05:42:00  c0-0c1s6a0n2   CorrectableMemErr   Link CRC error (cnt: 3)
121008 05:43:30  c0-0c1s6a0n2   Info                Correctable/CRC error
```

Examine Activity on the HSS Boot Manager

Use the HSS `xtcli session show` command to examine sessions in the boot manager. A session corresponds to running a specific command such as `xtcli power up` or `xtcli boot`. This command reports on sessions, not daemons.

For more information, see the `xtcli(8)` man page.

View a session running on the boot manager

```
crayadm@smw:~> xtcli session show BM
```

Poll a Response from an HSS Daemon, Manager, or the Event Router

Use the HSS `xtalive` command to verify that an HSS daemon, manager, or the event router is responsive.

For more information, see the `xtalive(8)` man page.

Check the boot manager

```
crayadm@smw:~> xtalive -l smw -a bm s0
```

Validate the Health of the HSS

The `xtcheckhss` command initiates a series of tests that validate the health of the HSS by gathering and displaying information supplied by scripts located on blade controllers (BCs) and cabinet controllers (CCs). `xtcheckhss` includes the following tests:

- **Version Checker**: Reads the current version running on the L0C, QLOC, L0Ds, BC micro, CC micro, CC FPGA, CHIA FPGAs, Tolapai BIOSes, and Node BIOS. The version that is read from each device is compared to the currently installed versions on the SMW.
- **Sensor Checker**: Reads environment sensors including temperatures, voltages, currents, and other data.
- **SEEP Checker:** Reads serial electrically erasable PROMs (SEEPs) in the system. This test can report any un-initialized, zeroed, or unreadable SEEPs.

- **AOC Checker**: Reads all active optical cable (AOC) data. This test displays any outliers relative to the average data calculated by previous runs.
- **ITP Checker**: Validates the embedded ITP path
- **NTP Checker**: Reads system time on all controllers and compares them with the SMW time; displays any mismatches.
- **Control Checker**: Examines and modifies system controls.
- **Configuration Information Checker**: Reads the system hardware configuration and reports the system setup, including the blade type, daughter card type, CPU type and count, and the CPU and PDC mask.
- **PCI checker**: Checks for missing or degraded PCIe connectivity on add-in cards on an IBB. This test requires that the nodes be powered up and bounced. Any cards that do not train to the PCIe Gen or Width specified in the Link Capability register are flagged. Any cards that are reported as physically present but not seen by the node are flagged.

For complete information, see the `xtcheckhss(8)` man page.

Monitor Event Router Daemon (erd) Events

The HSS `xtconsumer` command enables the system administrator to monitor events mediated by the event router daemon `erd`, which runs passively.

Monitor for specific events

For this example, watch two events: `ec_heartbeat_stop`, which will be sent if either the node stops sending heartbeats or if the system interconnection network ASIC stops sending heartbeats, and `ec_l0_health`, which will be sent if any of the subcomponents of a blade controller report a bad health indication:

```
crayadm@smw:~> xtconsumer -b ec_heartbeat_stop ec_l0_health
```

Use the `xthb` command to confirm the stopped heartbeat. Use the `xthb` command only when actively looking into a known problem because it is intrusive and degrades system performance.

Check events except heartbeat

```
crayadm@smw:~> xtconsumer -x ec_l1_heartbeat
```

For more information, see the `xtconsumer(8)` and `xthb(8)` man pages.

Monitor Node Console Messages

The `xtbootsys` command automatically initiates an `xtconsole` session, which displays the console text of a specified node(s) or accelerator(s). The `xtconsole` command operates in a shell window and monitors the event router daemon (`erd`) for console messages. The node or accelerator ID appears at the beginning of each line. The messages are written into `/var/opt/cray/log/`*`sessionid`*`/console-`*`yyyymmdd`* where the administrator may monitor them.

The `xtconsole` utility may only have one concurrent instance.

For more information, see the `xtconsole(8)` man page.

View Component Alert, Warning, and Location History

Use the `xtcli comp_hist` command to display component alert, warning, and location history. Either an error history, which displays alerts or warnings found on designated components, or a location history may be displayed.

Display the location history for component `c0-0c0s0n1`

```
crayadm@smw:~> xtcli comp_hist -o loc c0-0c0s0n1
```

For more information, see the `xtcli(8)` man page.

Display Component Information

Use the HSS `xtshow` command to identify compute and service components. Commands are typed as `xtshow --option_name`. Combine the `--service` or `--compute` option with other `xtshow` options to limit the selection to the specified type of node.

For a list of all `xtshow --option_name` options, see the `xtshow(8)` man page.

Identify all service nodes

```
crayadm@smw:~> xtshow --service
L1s ...
Cages ...
L0s ...
         c0-0c0s0: service        X86|       ready      [noflags|]
         c0-0c0s1: service        X86|       ready      [noflags|]
         c1-0c0s0: service        X86|       ready      [noflags|]
         c1-0c0s1: service        X86|       ready      [noflags|]
         c2-0c0s1: service        X86|       ready      [noflags|]
         c2-0c1s1: service        X86|       ready      [noflags|]
Nodes ...
       c0-0c0s0n0: service        X86|       empty      [noflags|]
       c0-0c0s0n1: service  SB08  X86|       ready      [noflags|]
       c0-0c0s0n2: service  SB08  X86|       ready      [noflags|]
       c0-0c0s0n3: service        X86|       empty      [noflags|]
       c0-0c0s1n0: service        X86|       empty      [noflags|]
       c0-0c0s1n1: service  SB08  X86|       ready      [noflags|]
            .
            .
            .
Aries ...
       c0-0c0s0a0: service        X86|          on      [noflags|]
       c0-0c0s1a0: service        X86|          on      [noflags|]
       c1-0c0s0a0: service        X86|          on      [noflags|]
       c1-0c0s1a0: service        X86|          on      [noflags|]
       c2-0c0s1a0: service        X86|          on      [noflags|]
       c2-0c1s1a0: service        X86|          on      [noflags|]
AriesLcbs ...
    c0-0c0s0a0l00: service        X86|          on      [noflags|]
    c0-0c0s0a0l01: service        X86|          on      [noflags|]
```

```
    c0-0c0s0a0l02: service        X86|          on      [noflags|]
    c0-0c0s0a0l03: service        X86|          on      [noflags|]
    c0-0c0s0a0l04: service        X86|          on      [noflags|]
    c0-0c0s0a0l05: service        X86|          on      [noflags|]
    c0-0c0s0a0l06: service        X86|          on      [noflags|]
      .
      .
      .
```

Identify compute nodes in the `disabled` state

```
crayadm@smw:~> xtshow --compute --disabled
L1s ...
Cages ...
L0s ...
Nodes ...
       c0-0c2s0n3:       -        X86|    disabled      [noflags|]
      c0-0c2s11n0:       -        X86|    disabled      [noflags|]
      c0-0c2s11n3:       -        X86|    disabled      [noflags|]
      c1-0c0s11n2:       -        X86|    disabled      [noflags|]
Aries ...
AriesLcbs ...
```

Identify components with a status of not empty

```
crayadm@smw:~> xtshow --not_empty c0-0c0s0
L1s ...
             c0-0:       -           |          on      [warn|alert|]
Cages ...
L0s ...
         c0-0c0s0: service        X86|       ready      [noflags|]
Nodes ...
       c0-0c0s0n1: service  SB08  X86|       ready      [noflags|]
       c0-0c0s0n2: service  SB08  X86|       ready      [noflags|]
Aries ...
       c0-0c0s0a0: service        X86|          on      [noflags|]
AriesLcbs ...
    c0-0c0s0a0l00: service        X86|          on      [noflags|]
    c0-0c0s0a0l01: service        X86|          on      [noflags|]
    c0-0c0s0a0l02: service        X86|          on      [noflags|]
    c0-0c0s0a0l03: service        X86|          on      [noflags|]
    c0-0c0s0a0l04: service        X86|          on      [noflags|]
    c0-0c0s0a0l05: service        X86|          on      [noflags|]
    c0-0c0s0a0l06: service        X86|          on      [noflags|]
      .
      .
      .
```

Display Alerts and Warnings

Use the `xtshow` command to display alerts and warnings. Type commands as **`xtshow`** `--option_name`, where *`option_name`* is `alert`, `warn`, or `noflags`.

Alerts are not propagated through the system hierarchy, only information for the component being examined is displayed. For example, invoking the `xtshow --alert` command for a cabinet does not display an alert for a node. Similarly, checking the status of a node does not detect an alert on a cabinet.

Show all alerts on the system

```
crayadm@smw:~> xtshow --alert
```

Alerts and warnings typically occur while the HSS `xtcli` command operates; these alerts and warnings are listed in the command output with an error message. After they are generated, alerts and warnings become part of the state for the component and remain set until manually cleared.

For example, the temporary loss of a heartbeat by the blade controller may set a warning state on a chip.

For additional information, see the `xtshow(8)` man page.

Display System Network Congestion Protection Information

Two utilities help to identify the time and duration of system network congestion events, either by parsing through logs (`xtcpreport`) or in real time (`xtcptop`):

xtcpreport This command uses information contained in the given `xtnlrd` file to extract and display information related to system network congestion protection. See the `xtcpreport(8)` man page for additional information.

xtcptop This command monitors an `xtnlrd` file that is currently being updated and displays real-time system network congestion protection information, including start time, duration, and apid. See the `xtcptop(8)` man page for additional information.

To use these utilities, load the `congestion-tools` module if it is not already loaded.

```
crayadm@smw:~> module load congestion-tools
```

Clear Component Flags

Use the `xtclear` command to clear system information for selected components. Type commands as `xtclear --option_name`, where *`option_name`* is `alert`, `reserve`, or `warn`.

Clear all warnings in specified cabinet

For this example, clear all warnings in cabinet `c13-2`:

```
smw:~> xtclear --warn c13-2
```

Alerts, reserves, and warnings must be cleared before a component can operate. Clearing an alert on a component frees its state so that subsequent commands can execute *System Component States*.

For more information, see the `xtclear(8)` man page.

Display Error Codes

When an HSS event error occurs, the related message is displayed on the SMW. The `xterrorcode` command on the SMW displays a single error code or the entire list of error codes.

Display HSS error codes

```
crayadm@smw:~>  xterrorcode errorcode
```

A system error code entered in a log file is a bit mask; invoking the `xterrorcode` *`bitmask_code_number`* command on the SMW displays the associated error code.

Display an HSS error code using its bit mask number

```
crayadm@smw:~> xterrorcode 131279
Maximum error code (RS_NUM_ERR_CODE) is 447
code = 207, string = 'Node Voltage Fault'
```

Cray Lightweight Log Management (LLM) System

The Cray Lightweight Log Management (LLM) system is the log infrastructure for Cray systems and must be enabled for systems to successfully log events. At a high level, a library is used to deliver messages to `rsyslog` utilizing the RFC 5424 protocol; `rsyslog` transports those messages to the SMW and places the messages into log files.

The LLM system relies on the *`sessionid`* that is generated by `xtbootsys`. Therefore, systems must always be booted using `xtbootsys`. If the site has multi-part boot procedures or uses manual procedures, have the process started by an `xtbootsys` session. That session can be effectively empty -- it is only needed to initiate a boot *`sessionid`*. Subsequent `xtbootsys` calls can then use `--session last` or manual processes.

By default, LLM has a log trimming mechanism enabled called `xttrim`.

> **IMPORTANT:** Do not use the `xtgetsyslog` command because it is not compatible with LLM.For additional information, see *Manage Log Files Using CLE and HSS Commands* on page 142.

For further information, see the `intro_LLM(8)` and `intro_LLM_logfiles(5)` man pages.

cdump and crash Utilities for Node Memory Dump and Analysis

The `cdump` and `crash` utilities may be used to analyze the memory on any Cray service node or CNL compute node. The `cdump` command is used to dump node memory to a file. After `cdump` completes, the `crash` utility can be used on the dump file generated by `cdump`.

Cray recommends executing the `cdump` utility only if a node has panicked or is hung, or if a dump is requested by Cray.

To select the desired access method for reading node memory, use the `cdump -r` *`access`* option. Valid access methods are:

xt-bhs The `xt-bhs` method uses a basic hardware system server that runs on the SMW to access and read node memory. `xt-bhs` is the default access method for these systems.

xt-hsn The `xt-hsn` method utilizes a proxy that reads node memory through the High-speed Network (HSN). The `xt-hsn` method is faster than the `xt-bhs` method, but there are situations where it will not work (for example, if the ASIC is not functional). However, the `xt-hsn` method is preferable because the dump completes in a short amount of time and the node can be returned to service sooner.

xt-file The `xt-file` method is used for memory dump file created by the `-z` option. The compressed memory dump file must be uncompressed prior to executing this command. Use the file name for *`node-id`*.

To dump Cray node memory, *`access`* takes the following form:

```
method[@host]
```

For additional information, see the `cdump(8)` and `crash(8)` man pages.

Resource Utilization Reporting

Resource Utilization Reporting (RUR) is an administrator tool for gathering statistics on how system resources are being used by applications or jobs. RUR is a low-noise, scalable infrastructure that collects compute node statistics before an application or job runs and again after it completes. The extensible RUR infrastructure allows plugins to be easily written to collect data uniquely interesting to each site. Cray supplied plugins collect a variety of data, including process accounting, energy usage, memory usage, and GPU accounting.

When RUR is enabled on a Cray system running CLE, resource utilization statistics are gathered from compute nodes running all applications or jobs. RUR is configured to run per application, per job, or both. RUR runs primarily before an application/job has started and after it ends, ensuring minimal impact on performance.

Prior to application/job runtime, the ALPS or WLM prologue script calls an RUR prologue script that, based on enabled plugins, initiates pre-application/pre-job data staging on all compute nodes used by the application/job. This staging may involve resetting counters to zero or collecting initial values of counters. Following application/job completion, the ALPS or WLM epilogue script calls an RUR epilogue script that gathers these counters, compares them to the initial values, where applicable, stages the data on the compute nodes, and then transfers data from the compute nodes to the login/MOM node. RUR post-processes the data to create a summary report that is written out to a log file or other backing store.

Plugin Architecture

RUR supports a plugin architecture, allowing many types of usage data to be collected while using the same software infrastructure. Two basic types of RUR plugins are supported: *data plugins*, which collect particular statistics about system resources, and *output plugins*, which send the output of the RUR software stack to a backing store.

Cray supplies plugins as part of the RUR distribution, including six data collection plugins, three output plugins, and one example plugin. Sites choose which plugins to enable or disable by modifying the RUR configuration file. See *Enable/Disable Plugins* for more information. Sites can also create custom plugins, specific to their needs, as described in *Create Custom RUR Data Plugins* on page 162 and *Create Custom RUR Output Plugins* on page 163.

RUR Configuration File

The RUR configuration file `/etc/opt/cray/rur/rur.conf` is located on the shared root. The file consists of distinct sections, identified by a header, `[section]`, that contain settings for specific RUR components. Changing the behavior of RUR components is possible through modification of the config file. Configurable values fall into five categories:

- Script/binary location
- Temporary data storage location
- Component timeout specs
- Enabling/disabling plugins
- Optional plugin behavior

Changes to the configuration file are automatically propagated to the nodes via the shared-root mount, requiring no administrator intervention or system reboot. An example configuration file `rur.config.example` is distributed with the RPM.

The energy Data Plugin (Cray XC Series only)

The `energy` plugin collects compute node energy usage data. The amount of data reported and the format in which it is written is determined by the value of the argument `arg` set in the `[energy]` section of the RUR configuration file.

If `arg` is not set (default) or set to `json-list`, the plugin reports the following, written in JavaScript Object Notation (JSON) list format:

`energy_used` The total energy (joules) used across all nodes. On nodes with accelerators, this value includes `accel_energy_used`, the total energy used by the accelerators.

RUR default `energy` output

This example shows default `energy` data as written to `/var/opt/cray/log/partition-current/messages-date` on the SMW:

```
2013-08-30T11:19:06.545114-05:00 c0-0c0s2n2 RUR 18657
p2-20130829t090349 [RUR@34] uid: 12345, apid: 10963, jobid: 0,
cmdname: /opt/intel/vtune_xe_2013/bin64/amplxe-cl plugin: energy
['energy_used', 318]
```

If `arg` is set to `json-dict`, the plugin also reports the following extended energy data, written in JSON dictionary format:

`error` If a Python exception occurs during the post or staging scripts, the following data is reported:

`traceback`	Stack frame list
`type`	Python exception type
`value`	Python exception parameter
`nid`	NID on which exception occurred
`cname`	cname on which exception occurred

nodes	Number of nodes in job
nodes_cpu_throttled	Number of nodes experiencing CPU power/thermal throttling
nodes_memory_throttled	Number of nodes experiencing memory power/thermal throttling
nodes_power_capped	Number of nodes with nonzero power cap
nodes_throttled	Number of nodes experiencing any of the following types of throttling: • CPU power/thermal throttling • Memory power/thermal throttling
nodes_with_changed_power_cap	Number of nodes with power caps that changed during execution. On nodes with accelerators, this value includes the number of accelerators with power caps that changed.
max_power_cap	Maximum nonzero power cap
max_power_cap_count	Number of nodes with the maximum nonzero power cap
min_power_cap	Minimum nonzero power cap
min_power_cap_count	Number of nodes with the minimum nonzero power cap

On nodes with accelerators, the extended data also include the following data:

accel_energy_used	Total accelerator energy (joules) used
nodes_accel_power_capped	Number of accelerators with nonzero power cap
max_accel_power_cap	Maximum nonzero accelerator power cap
max_accel_power_cap_count	Number of accelerators with the maximum nonzero power cap
min_accel_power_cap	Minimum nonzero accelerator power cap
min_accel_power_cap_count	Number of accelerators with the minimum nonzero power cap

RUR extended `energy` output

This example shows extended `energy` data as written to `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW:

```
2014-01-17T10:05:54.026557-06:00 c0-0c0s1n1 RUR 11674
p0-20140116t214834 [rur@34] uid: 12345, apid: 286342, jobid: 0,
cmdname: /bin/cat, plugin: energy {"energy_used": 5641,
"accel_energy_used": 1340, "nodes": 32, "nodes_power_capped": 3,
"min_power_cap": 155, "min_power_cap_count": 2, "max_power_cap": 355,
"max_power_cap_count": 1, "nodes_accel_power_capped": 3,
"min_accel_power_cap: 200, min_accel_power_cap_count": 3,
"max_accel_power_cap": 200, "max_accel_power_cap_count": 3,
"nodes_throttled": 0, "nodes_with_changed_power_cap": 0}
```

The gpustat Data Plugin

The `gpustat` plugin collects the following utilization statistics for NVIDIA GPUs, if present. The data is written in JSON list format.

maxmem	Maximum memory used across all nodes
summem	Total memory used across all nodes
gpusecs	Time spent processing on GPUs

RUR `gpustat` output

This example shows `gpustat` data as written in `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW.

```
2013-07-09T15:50:42.761257-05:00 c0-0c0s2n2 RUR 11329
p2-20130709t145714 [RUR@34] uid: 12345, apid: 8410, jobid: 0,
cmdname: /tmp/dostuff plugin: gpustats ['maxmem', 108000, 'summem',
108000, 'gpusecs', 44]
```

The kncstats Data Plugin

The `kncstats` plugin collects the following process accounting data from Intel Xeon Phi (KNC) coprocessors, if present. The data is written in JSON list format.

core	Set to `1` if core dump occurred
exitcode	Lists all unique exit codes
max_rss	Maximum memory used
rchar	Characters read by process
stime	System time
utime	User time
wchar	Characters written by process

RUR `kncstats` output

This example shows `kncstats` data as written to `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW:

```
2014-02-25T18:49:12.101383-06:00 c0-0c0s1n1 RUR 11274
p0-20140225t135439 [RUR@34] uid: 12345, apid: 539224, jobid: 0,
cmdname: /bin/date, plugin: kncstats ['utime', 8000, 'stime', 20000,
'max_rss', 620, 'rchar', 2730, 'wchar', 119, 'exitcode:signal',
['0:0'], 'core', 0]
```

The memory Data Plugin

The `memory` plugin collects information from `/proc` and `/sys` that is useful when assessing the memory performance of an application or job. The data is written in JSON dictionary format. The type of data reported is determined by the value of the argument `arg` set in the `[memory]` section of the RUR configuration file.

IMPORTANT: The `memory` plugin does not provide consolidated information for all nodes within an application; instead it reports memory statistics for each node within the application. This can result in a large amount of RUR output data for systems of even modest size. When the `memory` plugin is enabled, it produces a significant amount of output.

If `arg` is not set (default), the plugin reports the following data:

error	If a Python exception occurs during the post or staging scripts, the following data is reported:
	traceback Stack frame list
	type Python exception type
	value Python exception parameter
	nid NID on which exception occurred
	cname cname on which exception occurred
%_of_boot_mem	The % of boot memory for each order chunk in `/proc/buddyinfo` summed across all memory zones
Active(anon)	Total amount of memory in active use by the application
Active(file)	Total amount of memory in active use by cache and buffers
boot_freemem	Contents of `/proc/boot_freemem`
current_freemem	Contents of `/proc/current_freemem`
free	Number of hugepages that are not yet allocated
hugepages-*size*kB	The hugepage size for the select entries from `/sys/kernel/mm/hugepages/hugepages-*kB/*`
Inactive(anon)	Total amount of memory that is candidate to be swapped out
Inactive(file)	Total amount of memory that is candidate to be dropped from cache
nr	Number of hugepages that exist at this point
resv	Number of hugepages committed for allocation, but no allocation has occurred
Slab	Total amount of memory used by the kernel
surplus	Number of hugepages above `nr`

RUR default `memory` output

This example shows the default `memory` data as written to `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW.

```
2014-03-21T11:37:24.480982-05:00 c0-0c0s0n2 RUR 23710
p0-20140321t091957 [RUR@34] uid: 12345, apid: 33079, jobid: 0,
cmdname: /bin/hostname, plugin: memory {"current_freemem": 21858372,
"meminfo": {"Active(anon)": 35952, "Slab": 105824, "Inactive(anon)":
1104}, "hugepages-2048kB": {"nr": 5120, "surplus": 5120},
"%_of_boot_mem": ["67.23", "67.23", "67.23", "67.22", "67.21",
"67.18", "67.11", "67.04", "66.94", "66.83", "66.77", "66.66",
"66.53", "66.38", "65.87", "65.07", "63.05", "61.43"], "nid": "8",
"cname": "c0-0c0s2n0", "boot_freemem": 32432628}
```

If `arg` is set to `extended_buddy`, the output relating to `/proc/buddyinfo` includes NUMA node granularity information in addition to the existing node granularity information. This information is useful when troubleshooting certain fragmentation related issues.

RUR extended `memory` output

This example shows extended `memory` data as written to `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW:

```
2014-03-21T11:37:24.480982-05:00 c0-0c0s0n2 RUR 23710
p0-20140321t091957 [RUR@34] uid: 12345, apid: 33079, jobid: 0,
cmdname: /bin/hostname, plugin: memory {"current_freemem": 21858372,
"meminfo": {"Active(anon)": 35952, "Slab": 105824, "Inactive(anon)":
1104}, "hugepages-2048kB": {"nr": 5120, "surplus": 5120},
"Node_0_zone_DMA": ["0.05", "0.05", "0.05", "0.05", "0.05", "0.05",
"0.05", "0.05", "0.05", "0.04", "0.04", "0.03", "0.00", "0.00",
"0.00", "0.00", "0.00", "0.00"],"%_of_boot_mem": ["67.23", "67.23",
"67.23", "67.22", "67.21", "67.18", "67.11", "67.04", "66.94",
"66.83", "66.77", "66.66", "66.53", "66.38", "65.87", "65.07",
"63.05", "61.43"], "nid": "8", "cname": "c0-0c0s2n0", "boot_freemem":
32432628, "Node_0_zone_DMA32": ["6.07", "6.07", "6.07", "6.07",
"6.07", "6.07", "6.07", "6.06", "6.05", "6.04", "6.01", "5.94",
"5.86", "5.76", "5.46", "4.85", "3.23", "3.23"], "Node_0_zone_Normal":
["61.11", "61.11", "61.11", "61.11", "61.09", "61.07", "60.99",
"60.93", "60.84", "60.75", "60.72", "60.70", "60.67", "60.62",
"60.42", "60.22", "59.81", "58.20"]}
```

The taskstats Data Plugin

ATTENTION: The default setting of the configuration argument `arg` will change from `json-list` to `json-dict` in release CLE6.0. This will result in changes to the content and format of the default output. `json-list` is deprecated and will be removed in a future release but will remain functional until that time.

The `taskstats` plugin collects process accounting data. The amount of data reported and the format in which it is written is determined by the value of the argument `arg` set in the `[taskstats]` section of the RUR configuration file.

If `arg` is not set (default), the plugin reports the following basic process accounting data similar to that provided by UNIX process accounting or `getrusage`. These values are sums across all nodes, except for the memory used, which is the maximum value across all nodes. The data is written in JSON list format.

`core`	Set to `1` if core dump occurred
`exitcode`	Lists all unique exit codes
`max_rss`	Maximum memory used
`rchar`	Characters read by process
`stime`	System time
`utime`	User time
`wchar`	Characters written by process

RUR default `taskstats` output

This example shows default `taskstats` output as written to `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW.

For a job that exits normally:

```
2013-11-02T11:09:49.457770-05:00 c0-0c1s1n2 RUR 2417
p0-20131101t153028 [RUR@34] uid: 12345, apid: 86989, jobid: 0,
cmdname: /lus/tmp/rur01.2338/./CPU01-2338 plugin: taskstats ['utime',
10000000, 'stime', 0, 'max_rss', 940, 'rchar', 107480, 'wchar', 90,
'exitcode:signal', ['0:0'], 'core', 0]
```

For a job that core dumps:

```
2013-11-02T11:12:45.020716-05:00 c0-0c1s1n2 RUR 3731
p0-20131101t153028 [RUR@34] uid: 12345, apid: 86996, jobid: 0,
cmdname: /lus/tmp/rur01.3657/./exit04-3657 plugin: taskstats ['utime',
4000, 'stime', 144000, 'max_rss', 7336, 'rchar', 252289, 'wchar', 741,
'exitcode:signal', ['0:9', '139:0', '0:11', '0:0'], 'core', 1]
```

If `arg` is set to `xpacct`, the plugin also provides the following extended process accounting data similar to that which was collected by the deprecated Cray System Accounting (CSA).

abortinfo	If abnormal termination occurs, a list of `abort_info` fields is reported
apid	Application ID as defined by application launcher
bkiowait	Total delay time (ns) waiting for synchronous block I/O to complete
btime	UNIX time when process started
comm	String containing process name. May be different than the header, which is the process run by the launcher.
coremem	Integral of RSS used by process in MB-usec
ecode	Process exit code
etime	Total elapsed time in microseconds
gid	Group ID
jid	Job ID - the PAGG job container used on the compute node
majfault	Number of major page faults
minfault	Number of minor page faults
nice	POSIX `nice value` of process
nid	String containing node ID
pgswapcnt	Number of pages swapped; should be `0` on Cray compute nodes
pid	Process ID
pjid	Parent job ID - the PAGG job container on the MOM node
ppid	Parent process ID
prid	Job project ID
rcalls	Number of read system calls

[1] The current memory usage is added to these counters (i.e., `coremem`, `vm`) every time. A tick is charged to a task's system time. Therefore, at the end we will have memory usage multiplied by system time and an average usage per system time unit can be calculated.

rchar	Characters read by process
rss	RSS highwater mark
sched	Scheduling discipline used on node
uid	User ID
vm	Integral of virtual memory used by process in MB-usecs[2]
wcalls	Number of write system calls
wchar	Characters written by process

RUR extended `taskstats` output

This example shows RUR extended `taskstats` output:

```
2013-10-18T10:29:38.285378-05:00 c0-0c0s1n1 RUR 24393
p1-20131018t081133 [RUR@34] uid: 12345, apid: 370583, jobid: 0,
cmdname: /bin/cat, plugin: taskstats ['btime', 1386061749, 'etime',
8000, 'utime', 0, 'stime', 4000, 'coremem', 442, 'max_rss', 564,
'max_vm', 564, 'pgswapcnt', 63, 'minfault', 15, 'majfault', 48,
'rchar', 2608, 'wchar', 686, 'rcalls', 19, 'wcalls', 7, 'bkiowait',
1000, 'exitcode:signal', [0], 'core', 0]
```

If `arg` is set to `xpacct, per-process`, the plugin reports extended accounting data for every compute node process rather than a summary of all processes for an application. `per-process` must be set in combination with `xpacct`.

CAUTION: If `per-process` is set and many processes are run on each node, the volume of data generated and stored on disk can become an issue.

RUR per-process `taskstats` output

This exaple shows RUR per-process `taskstats` output. This output was generated with the `json-dict` option set.

```
2013-12-03T13:25:34.446167-06:00 c0-0c2s0n2 RUR 7623
p3-20131202t090205 [RUR@34] uid: 12345, apid: 1560, jobid: 0,
cmdname: ./it.sh, plugin: taskstats {"uid": 12795, "wcalls": 37,
"pid": 2997, "vm": 16348, "jid": 395136991233, "bkiowait": 1201616,
"majfault": 1, "etime": 0, "btime": 1386098731, "gid": 0, "ppid":
2992, "utime": 0, "nice": 0, "sched": 0, "nid": "92", "prid": 0,
"comm": "mount", "stime": 4000, "wchar": 3465, "rss": 1028,
"minfault": 352, "coremem": 1109, "ecode": 0, "rcalls": 22, "pjid":
7045, "pgswapcnt": 0, "rchar": 12208}

2013-12-03T13:25:34.949138-06:00 c0-0c2s0n2 RUR 7623
p3-20131202t090205 [RUR@34] uid: 12345, apid: 1560, jobid: 0,
cmdname: ./it.sh, plugin: taskstats {"uid": 12795, "wcalls": 0, "pid":
2998, "vm": 20268, "jid": 395136991233, "bkiowait": 0, "majfault": 0,
```

[2] The current memory usage is added to these counters (i.e., `coremem`, `vm`) every time. A tick is charged to a task's system time. Therefore, at the end we will have memory usage multiplied by system time and an average usage per system time unit can be calculated.

```
"etime": 0, "btime": 1386098731, "gid": 0, "ppid": 2992, "utime": 0,
"nice": 0, "sched": 0, "nid": "92", "prid": 0, "apid": 1560, "comm":
"ls", "stime": 4000, "wchar": 0, "rss": 1040, "minfault": 360,
"coremem": 3140, "ecode": 0, "rcalls": 19, "pjid": 7045, "pgswapcnt":
0, "rchar": 10629}
```

If `arg` is set to `json-dict`, the data is written in JSON dictionary format.

If `arg` is set to `json-list`, the data is written in JSON list format (default).

The timestamp Data Plugin

The `timestamp` plugin collects the start and end times of an application or job.

RUR `timestamp` output

This example shows `timestamp` data, as written in `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW, for an application that slept 20 seconds:

```
2013-08-30T14:32:07.593469-05:00 c0-0c0s5n2 RUR 12882
p3-20130830t074847 [RUR@34] uid: 12345, apid: 6640, jobid: 0,
cmdname: /bin/sleep plugin: timestamp APP_START 2013-08-30T14:31:46CDT
APP_STOP 2013-08-30T14:32:06CDT
```

The file Output Plugin

The `file` plugin allows RUR data to be stored to a flat text file on any file system to which the login node can write. This plugin is also intended as a very simple guide for anyone interested in writing an output plugin.

This example shows sample output from `file` to a location defined in the RUR configuration file:

```
uid: 1000, apid: 8410, jobid: 0, cmdname: /tmp/dostuff plugin:
taskstats ['utime', 32000, 'stime', 132000, 'max_rss', 1736, 'rchar',
44524, 'wchar', 289] uid: 1000, apid: 8410, jobid: 0, cmdname: /tmp/
dostuff plugin: energy ['energy_used', 24551] uid: 1000, apid: 8410,
jobid: 0, cmdname: /tmp/dostuff plugin: gpustats ['maxmem', 108000,
'summem', 108000]
```

The llm Output Plugin

The `llm` plugin aggregates log messages from various Cray nodes and places them on the SMW. `llm` has its own configuration options, but typically it will place RUR messages into the messages log file `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW. The messages shown in the previous sections are in LLM log format.

The user Output Plugin

The `user` plugin writes RUR output for a user's application to the user's home directory (default) or a user-defined location, only if the user has indicated that this behavior is desired (as described below).

The naming of the default output file(s), `rur.`*`suffix`*, is dependent on the value of the argument `arg`, which defines a report type and is set in the `user` section of the RUR configuration file. If `arg` is set to:

apid	An output file is created for each application executed and *suffix* is the `apid`.
jobid	An output file is created for each job submitted and *suffix* is the `jobid`
single	All output is placed in a single file and no suffix is appended to the output file name.

User Options

Users have the option to opt-in or out for the `user` plugin, redirect plugin output to a specific file or directory, or override the default report type.

- By default, RUR data is written to a user's directory. A user must either create the file `~/.rur/user_output_optin` to indicate that data should be written, or create a file that initiates one of the following two options.

  1. Users may redirect the output of RUR by specifying a redirect location in `~/.rur/user_output_redirect`. The contents of this file must be a single line that specifies the absolute or relative (from the user's home directory) path of the directory or file to which the RUR output data is to be written. If the redirect file either does not exist, points to a path that does not exist, or points to a path to which the user does not have write permission, then the output is written to the user's home directory.
  2. A user with an existing `~/.rur/user_output_redirect` file can temporarily stop RUR data from being written by setting the redirect path to `/dev/null`.

- Additionally, the user may override the default report type by specifying a valid report type in `~/.rur/user_output_report_type`. Valid report types are `apid`, `jobid`, or `single`, resulting in the user's RUR data being written to one file per application, one file per job, or a single file, respectively. If the file `~/.rur/user_output_report_type` is empty or contains an invalid type, then the default report type, as defined in the configuration file, is created.

The database Example Output Plugin

The `database` plugin is provided as a guide for sites wanting to output RUR data to a site-supplied database. Sites will need to configure their own systems, provide an external database, create their own tables, and modify `database_output.py` to collect the desired data.

MySQL is the database supported by the example plugin. The following arguments are defined for connecting to a database:

- `DB_NAME='rur'`
- `DB_USER='rur_user'`
- `DB_PASS='rur_pass'`
- `DB_HOST='rur_host'`

  The `database` plugin collects the values: `energy_used`, `apid`, `jobid`, and `uid`, and saves this data to a table, `energy`. It does this by performing the following:

- Digests RUR data into a dictionary and saves it to class `DbData`
- Creates rules for saving data collected in `DbData` to particular tables
- Uses the rules to scan the `DbData` dictionary and `INSERT` that data into a database

  Cray recommends that the database is not hosted on SDB or login nodes. It should also be noted that, depending on job load, interacting with an external database may cause system latency.

Create Custom RUR Data Plugins

A data plugin is comprised of a *staging component* and a *post processing component*. The data plugin staging component is called by `rur-stage.py` on the compute node prior to the application/job running and again after the application/job has completed. The staging component may reset counters before application/job execution and collect them after application/job completion, or it may collect initial and final values prior to and after application/job execution, respectively, and then calculate the delta values. Python functions have been defined to simplify writing plugins, although it is not necessary for the plugin to be written in Python. The interface for the data plugin staging component is through command line arguments.

Data Plugin Staging Component

All data plugin staging components must support the following arguments:

--apid=*apid*	Defines the application ID of the running application.
--timeout=*time*	Defines a timeout period in seconds during which the plugin must finish running. Set to `0` for unlimited; default is unlimited.
--pre	Indicates the plugin is being called prior to the application/job.
--post	Indicates the plugin is being called after the application/job.
--outputfile=*output_file*	Defines where the output data is written. Each plugin should define a default output file in `/var/spool/RUR/` if this argument is not provided.
--arg=*arg*	A plugin-specific argument, set in the RUR config file. RUR treats this as an opaque string.

The output of an RUR data plugin staging component is a temporary file located in `/var/spool/RUR` on the compute node. The file name must include both the name of the plugin, as defined in the RUR config file, and *`.apid`*. The RUR gather phase will automatically gather the staged files from all compute nodes after the application/job has completed and place it in `gather_dir` as defined in the configuration file.

Data plugin staging component

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# Sample data plugin staging component
#
#!/usr/bin/env python
import sys, os, getopt
from rur_plugins import rur_plugin_args
def main():
   apid, inputfile, outputfile, timeout, pre, post, \
                parg = rur_plugin_args(sys.argv[1:])
   if outputfile is "":
       outputfile = "/var/spool/RUR/pluginname."+str(apid)
   if (pre==1):
       zero_counters()
   else:
       write_postapp_stateto(outputfile)

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

Data Plugin Post Processing Component

A data plugin also requires a post processing component that processes the data staged by the staging component and collected during the RUR gather phase. The post processing component is called by `rur-post.py`. The input file contains records, one node per line, of all of the statistics created by the staging component. The output of the post processing component is a file containing the summary of data from all compute nodes.

All data plugin post processing components must support the following arguments:

**`--apid=`*`apid`*	Defines the application ID of the running application.
**`--timeout=`*`time`*	Defines a timeout period in seconds during which the plugin must finish running. Set to `0` for unlimited; default is unlimited.
**`--inputfile=`*`input_file`*	Specifies the file from which the plugin gets its input data.
**`--outputfile=`*`output_file`*	Specifies the file to which the plugin writes its output data.

Data plugin post processing component

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# Sample data plugin post processing component
#
#!/usr/bin/env python
import sys, os
from rur_plugins import rur_args

def main():
   apid, inputfile, outputfile, timeout = rur_args(sys.argv[1:])
   if outputfile is "":
       outputfile = inputfile + ".out"

   pc = PostCompute()
   pc.process_file(inputfile)
   formated = pc.present_entries([('plugin_foo_data','sum')])
   fout=open(outputfile, 'w+')
   fout.write("energy %s" % formated)

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

Create Custom RUR Output Plugins

Output plugins allow RUR data to be outputted to an arbitrary backing store. This can be a storage device or another piece of software that then consumes the RUR data. The output plugin is passed a number of command line arguments that describe the application/job run and provide a list of input working files (the output of data plugin post processing components). The plugin takes the data in the working files and exports it to the destination specified in the RUR configuration file for the specific output plugin.

Data passed to custom output plugins can be optionally configured to be JSON-formatted by adding the `use_json` argument to the `[global] section of the configuration file and setting it to True`, `yes`, `1`, or `enable`.

TIP: If there is an error from an output plugin, the error message appears in the ALPS log `/var/opt/cray/alps/log/apsys` on the service node rather than the LLM logs on the SMW.

Output Plugin

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# Sample output plugin
#
#!/usr/bin/env python
import sys, os
from rur_plugins import rur_output_args

def main():
   apid, jobid, uid, cmdname, inputfilelist, timeout, \
   parg = rur_output_args(sys.argv[1:])

   outfile = open(parg, "a")
   for inputfile in inputfilelist:
       infile =  open(inputfile, "r")
       lines = infile.readlines()
       for line in lines:
           outfile.write(line)
       infile.close()
   outfile.close()
```

Implement a Site-Written RUR Plugin

About this task

For a site written plugin to run, it must be added to the RUR configuration file and enabled. Follow the procedure to define and configure a new plugin.

Procedure

1. Ensure that the site written plugin is located on a file system that is readable by compute nodes, owned by `root`, and not writeable by non-`root` users.

2. Add a new plugin definition section to the RUR configuration file:

   a. If adding a data plugin, the definition section must include: the plugin name, a `stage` definition (the complete path to the plugin's data staging script), and a `post` definition (the complete path to the plugin's post processing script).

      For example, to define the site written data plugin `siteplug`, the entry within the RUR configuration would be similar to the following:

```
# The siteplug Data Plugin collects data that is
#     of particular interest to this site.
# Stage - The staging component run by rur_stage on the
#     compute node
# Post - The post-processing component run by rur_post on
#     the login/mom node
[siteplug]
```

```
stage: /opt/cray/rur/default/bin/siteplug_stage.py
post: /opt/cray/rur/default/bin/siteplug_post.py
```

b. If adding an output plugin, the definition section must include: the plugin name, an `output` definition (the complete path to the output plugin script or binary), and an optional argument.

   For example, to define the site written output plugin `siteout`, the entry within the RUR configuration would be similar to the following:

```
# The siteout output plugin.
# Write RUR output to a text file on the site's huge
# archive file system.
[siteout]
output:/opt/cray/rur/site/bin/site_output.py
arg:hsmuser@hsmbackup.site.com:/hsmuser/rurbackup
```

3. Add the new plugin to either the data plugin or output plugin configuration section, labeled `[plugins]` or `[outputplugins]`, respectively. Indicate `true` to enable or `false` to disable plugin execution.

   This example shows the new `siteplug` data plugin enabled and the `siteout` output plugin disabled:

```
# Data Plugins section Configuration
# Define the supported Data plugins and enable/disable
# them. Plugins defined as "Plugin: False" will not run,
# but will be parsed for correct config file syntax.
[plugins]
gpustat: true
taskstats: true
siteplug: true

# Output Plugins section Configuration
# Define which output plugins are supported, and enable/
# disable them. Plugins defined as "Plugin: False" will
# not run, but will be parsed for correct config file
# syntax.
[outputplugins]
llm: true
file: false
siteout: false
```

Additional Plugin Examples

This is a set of RUR plugins that report information about the number of available huge pages on each node. The huge page counts are reported in `/proc/buddyinfo`. There are two versions of the staging component: one that reports what is available and the second that reports changes during the application run.

Huge pages data plugin staging component (version A)

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# This is an RUR plugin that reports information about the number of
available
# huge pages on each node.  This is reported in /proc/buddyinfo.
#
# Each node reports its nid and the number of available pages of
each size.
```

```
#
#!/usr/bin/env python
import sys, os, getopt
from rur_plugins import rur_plugin_args
def main():
    apid, inputfile, outputfile, timeout, pre, post, parg
=rur_plugin_args(sys.argv[1:])
    if outputfile == 0:
         outputfile = "/var/spool/RUR/buddyinfo."+str(apid)
    if (pre==1):
         zero_counters()
    else:
         nidf = open("/proc/cray_xt/nid", "r")
         n = nidf.readlines()
         nid = int(n[0])
         inf = open("/proc/buddyinfo", "r")
         b = inf.readlines()
         sizes = dict( [( '2M' , 0 ), ('4M', 0), ('8M', 0), ('16M',
0), ('32M', 0), ('64M', 0)])

         for line in b:
              l = line.split()
              sizes['2M'] += int(l[13])
              sizes['4M'] += int(l[14])
              sizes['8M'] += int(l[15])
              sizes['16M'] += int(l[16])
              sizes['32M'] += int(l[17])
              sizes['64M'] += int(l[18])

        o = open(outputfile, "w")
        o.write("{6} {0} {1} {2} {3} {4}
{5}".format(sizes['2M'],sizes['4M'], \
        sizes['8M'], sizes['16M'], sizes['32M'], sizes['64M'], nid))
        o.close()

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

Huge pages data plugin staging component (version B)

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# This is an RUR plugin that reports information about the number of
available
# huge pages on each node.  This is reported in /proc/buddyinfo.
#
# This plugin records the number of available pages before the job
is launched.
# At job completion time it reports the change
#
#!/usr/bin/env python
import sys, os, getopt
from rur_plugins import rur_plugin_args
def main():
    apid, inputfile, outputfile, timeout, pre, post, parg
=rur_plugin_args(sys.argv[1:])
    if outputfile == 0:
```

```
            outputfile = "/var/spool/RUR/buddyinfo."+str(apid)
    if (pre==1):
            inf = open("/proc/buddyinfo", "r")
            b = inf.readlines()
            sizes = dict( [( '2M' , 0 ), ('4M', 0), ('8M', 0),
('16M', 0), ('32M', 0), ('64M', 0)])
            for line in b:
                    l = line.split()
                    sizes['2M'] += int(l[13])
                    sizes['4M'] += int(l[14])
                    sizes['8M'] += int(l[15])
                    sizes['16M'] += int(l[16])
                    sizes['32M'] += int(l[17])
                    sizes['64M'] += int(l[18])

        o = open("/tmp/buddyinfo_save", "w")
        o.write("{0} {1} {2} {3} {4}
{5}".format(sizes['2M'],sizes['4M'], \
                sizes['8M'], sizes['16M'], sizes['32M'],
sizes['64M']))
        o.close()
    else:
            nidf = open("/proc/cray_xt/nid", "r")
            n = nidf.readlines()
            nid = int(n[0])
            inf = open("/proc/buddyinfo", "r")
            b = inf.readlines()
            sizes = dict( [( '2M' , 0 ), ('4M', 0), ('8M', 0),
('16M', 0), ('32M', 0), ('64M', 0)])

            for line in b:
                    l = line.split()
                    sizes['2M'] += int(l[13])
                    sizes['4M'] += int(l[14])
                    sizes['8M'] += int(l[15])
                    sizes['16M'] += int(l[16])
                    sizes['32M'] += int(l[17])
                    sizes['64M'] += int(l[18])

            obf = open("/tmp/buddyinfo_save", "r")
            ob = obf.readlines()
            n=0

            obd0 = ob[0]
            obd = obd0.split()

            diff = [
                    (int(obd[0]) - sizes['2M']),
                    (int(obd[1]) - sizes['4M']),
                    (int(obd[2]) - sizes['8M']),
                    (int(obd[3]) - sizes['16M']),
                    (int(obd[4]) - sizes['32M']),
                    (int(obd[5]) - sizes['64M'])
                    ]

        o = open(outputfile, "w")
    # uncomment the following line to get the actual sizes
        #o.write("sizes {6} {0} {1} {2} {3} {4}
{5}\n".format(sizes['2M'],sizes['4M'], \
                sizes['8M'], sizes['16M'], sizes['32M'],
sizes['64M'], nid))
```

```
        o.write("diff {6} {0} {1} {2} {3} {4} {5}".format(diff[0],
diff[1], diff[2], \
                diff[3], diff[4], diff[5], nid))
        o.close()
    os.unlink("/tmp/buddyinfo_save")

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

Huge pages data plugin post processing component

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# This is a RUR postprocessing pluging for the buddyinfo data
# collection. It copies the input files to output, adding a
# "buddyinfo" label.
#
#!/usr/bin/env python
import sys, os
from rur_plugins import rur_args

def main():
   apid, inputfile, outputfile, timeout = rur_args(sys.argv[1:])
   if outputfile == 0:
       outputfile = inputfile + ".out"

   fin=open(inputfile, "r")
   l = fin.readlines()

   fout=open(outputfile, 'w+')
   for line in l:
            fout.write("buddyinfo {0}".format(line))

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

Application Completion Reporting (ACR) to RUR Migration Tips

Cray supplied RUR data plugins collect the same data found in Mazama's Application Completion Reporting (ACR) feature (deprecated), but RUR does not include a reporting utility like `mzreport`. When using RUR's `llm` output plugin, the type of data reported by `mzreport` can be extracted from the output files as demonstrated in the following sections.

ACR Job Reporting

The information provided by `mzreport -j` and `mzreport --job` can easily be obtained in the RUR environment from the log files `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* by invoking the following command:

```
smw:~ # grep -e "RUR" messages-* |grep -e "jobid: jobid"
```

ACR Timespan Reporting

In ACR, `mzreport -t` and `mzreport -T` control the span of time over which job completions are reported. The following example is a simple Python script, `timesearch.py`, that provides this functionality.

```
#cat timesearch.py
#!/usr/bin/env python
for rurline in [line for line in open(sys.argv[1], 'r') if 'RUR' in line]:
   if (rurline.split(' ')[1] > sys.argv[2]) and (rurline.split(' ')[1] <
sys.argv[3]):
       print rurline
```

The script is called with the log file of interest and the desired start/stop time stamps, where *start_time* and *end_time* are formatted as "*yyyy-mm-dd*T*hh:mm:ss*", as follows:

```
smw:~ # python ./timesearch.py messages-date "start_time" "end_time"
```

ACR Exit Code Reporting

The `get_exit.py` Python script listed here provides a list of the user IDs with the most non-zero exit codes.

```
# cat get_exit.py
#!/usr/bin/env python
import os,sys,re

statre = re.compile("'(\w*):(\w*)',\s*\[('(\w*):(\w*)'(, )?)+\]")
statsre = re.compile("(\w*):(\w*)")
uidre  = re.compile("uid:\s*(\w*)")
cnt = {}

for rurline in [line for line in open(sys.argv[1], 'r') if 'RUR' in line]:
   if 'taskstats' in rurline:
       sus = statre.search(rurline)
       status = sus.group()
       stats = statsre.findall(status)
       for stat in stats[1:]:
           if stat[0] != '0':
               uid = int(uidre.findall(rurline)[0])
               if cnt.get(str(uid)):
                   cnt[str(uid)] += 1
               else:
                   cnt[str(uid)] = 1

x = sorted(cnt, key = cnt.get, reverse=True)
print "uids with the most non-zero exit codes %s" % x[:sys.argv[2]]
```

The script is called with the log file of interest and the number of user IDs on which to report, as follows:

```
smw:~ # python ./get_exit.py messages-date num
```

Application Resource Utilization (ARU) to RUR Migration Tips

Sites that use ARU (deprecated) will have an easy transition to RUR as all of the data provided in ARU is available in RUR, but in a slightly different format.

> This example shows that the following ARU output is available by enabling the `taskstats` plugin's default behavior:

ARU output:

```
2012-11-26T08:52:37.802113-06:00 c0-0c0s0n2 apsys 19864
p0-20121126t060549 -
apid=6240364, Finishing, user=8855, batch_id=114.sdb, exit_code=0,
exitcode_array=0,
exitsignal_array=0, utime=0 stime=0 maxrss=3168 inblocks=0 outblocks=0
cpus=8
start=Mon Nov 26 08:52:37 2012 stop=Wed Dec 31 18:00:00 1969
cmd=growfiles
```

RUR `taskstats` default output:

```
2013-11-02T11:09:49.457770-05:00 c0-0c1s1n2 RUR 2417
p0-20131101t153028 [RUR@34]
uid: 10973, apid: 86989, jobid: 0, cmdname: /lus/esfs/overby/
rur01.2338/./CPU01-2338
plugin: taskstats ['utime', 10000000, 'stime', 0, 'max_rss', 940,
'rchar', 107480,
'wchar', 90, 'exitcode:signal', ['0:0'], 'core', 0]
```

This example shows that the following ARU output is available by enabling the RUR `timestamp` plugin.

ARU output:

```
2012-11-26T08:53:15.618239-06:00 c0-0c0s0n2 apsys 20604
p0-20121126t060549 -
apid=6240378, Finishing,user=8855, batch_id=121.sdb, exit_code=0,
exitcode_array=0,
exitsignal_array=0, utime=0 stime=0 maxrss=3152 inblocks=0 outblocks=0
cpus=1
start=Mon Nov 26 08:52:51 2012 stop=Wed Dec 31 18:00:00 1969
cmd=close2_01
```

RUR `timestamp` plugin output:

```
2013-08-30T14:32:07.593469-05:00 c0-0c0s5n2 RUR 12882
p3-20130830t074847 [RUR@34] uid: 0,
apid: 6640, jobid: 0, cmdname: /bin/sleep plugin: timestamp APP_START
2013-08-30T14:31:46CDT APP_STOP 2013-08-30T14:32:06CDT
```

CSA to RUR Migration Tips

The Cray supplied RUR data plugin `taskstats`, when enabled and configured for extended accounting data, collects all of the data in the CSA process accounting record with the exception of `ac_sbu`, the system billing units.

RUR extended `taskstats` output

This example shows RUR extended `taskstats` output:

```
2013-10-18T10:29:38.285378-05:00 c0-0c0s1n1 RUR 24393
p1-20131018t081133 [RUR@34] uid: 12345, apid: 370583, jobid: 0,
cmdname: /bin/cat, plugin: taskstats ['btime', 1386061749, 'etime',
8000, 'utime', 0, 'stime', 4000, 'coremem', 442, 'max_rss', 564,
```

```
'max_vm', 564, 'pgswapcnt', 63, 'minfault', 15, 'majfault', 48,
'rchar', 2608, 'wchar', 686, 'rcalls', 19, 'wcalls', 7, 'bkiowait',
1000, 'exitcode:signal', [0], 'core', 0]
```

RUR does not include the report generation capabilities provided by CSA, however, the type of data reported by CSA can be extracted from the messages files on the SMW. The following is a short Python script for searching through these files. It allows filtering for group ID (`-g`), job ID (`-j`), user ID (`-u`), and system time exceeding a certain value (`-s`); similar to the `csacom` filters `-g`, `-j`, `-u`, `-O`, respectively.

```
#!/usr/bin/env python
# Usage: filter-messages [-g gid] [-j jid] [-u uid] [-s stime] -f messages-date
import os,sys,re,getopt,collections

def getcmdlineargs(args):
   arglist = collections.defaultdict(lambda: 0, {})
   options, remainder = getopt.getopt(args,
       'g:j:u:s:f:',
       ['gid=', 'jid=', 'uid=', 'Stimeexceeds=', 'filename='])

   for opt,arg in options:
       if opt in ('-g', '--gid'):
           arglist['gid'] = arg
       if opt in ('-j', '--jid'):
           arglist['jid'] = arg
       if opt in ('-u', '--uid'):
           arglist['uid'] = arg
       if opt in ('-s', '--Stimeexceeds'):
           arglist['stimeexceeds'] = arg
       if opt in ('-f', '--filename'):
           arglist['filename'] = arg
   return arglist

def reeqgt(tag, restr, rurline, eq):
   retre = re.compile("'" + str(restr) + "'," + "\s*(\w*)")
   field = retre.findall(rurline)
   if field == []:
       return False
   if eq and tag == field[0]:
       return True
   elif (not eq) and tag <= field[0]:
       return True
   return False

arglist = getcmdlineargs(sys.argv[1:])
if not arglist['filename']:
   exit(1)
for rurline in [line for line in open(arglist['filename'], 'r') if 'RUR' in
line]:
  if 'taskstats' in rurline:
       if arglist['jid'] and not (reeqgt(arglist['jid'], 'jid', rurline, 1)):
               continue
       if arglist['uid'] and not (reeqgt(arglist['uid'], 'uid', rurline, 1)):
               continue
       if arglist['gid'] and not (reeqgt(arglist['gid'], 'gid', rurline, 1)):
               continue
       if arglist['stimeexceeds'] and not (reeqgt(arglist['stimeexceeds'],
'stime',
           rurline, 0)):
               continue
```

```
        print "%s" % rurline,
```

Modify an Installed System

Disable Boot-node Failover

About this task

For the examples in this procedure, the primary boot node is `c0-0c0s0n1` and the backup boot node is `c2-0c1s7n1`.

Procedure

1. Halt the primary and backup boot nodes.

```
crayadm@smw:~> xtcli halt c0-0c0s0n1,c2-0c1s7n1
```

2. Update the default boot configuration.

```
crayadm@smw:~> xtcli boot_cfg update -b c0-0c0s0n1,c0-0c0s0n1
```

3. Update the HSS daemon.

```
crayadm@smw:~> xtdaemonconfig c0-0c0s0 stonith=false
```

The Node ARP Management Daemon (rca_arpd)

The node ARP management daemon (`rca_arpd`) manages the system ARP cache. This daemon deletes the IP to hardware address (ARP) mappings for failed nodes and reads them when they become available. It only manages ARP mappings on the high speed interconnect network and not external network interfaces such as Ethernet. If failover is configured, `rca_arpd` also manages ARP mappings for the backup boot or SDB node. When a node failed event from the primary boot or SDB node is received, `rca_arpd` updates the ARP mapping for the boot or SDB node virtual IP address to point to the backup node.

This functionality is included in the `cray-rca-compute` and `cray-rca-service` RPMs and is installed by default.

Create Logical Machines for Cray XC Series Systems

Configure a logical machine (sometimes known as a *system partition*) with the `xtcli part_cfg` command. Partition IDs are predefined as `p0` to `p31`. The default partition `p0` is reserved for the complete system and is no longer a valid ID once a system has been partitioned.

systems can have one or more cabinets. Systems with one or two compute cabinets scale at the blade level. For larger liquid-cooled systems, every cabinet is fully populated (with 3 chassis), with the possible exception of the last cabinet.

For systems, groups are made up of two-cabinet pairs starting from the beginning. The last group may not be completely full, and it can consist of 1 to 6 fully-populated chassis.

Multiple Group Systems

When a system contains multiple groups, the system administrator can partition the system at a per-group level of granularity. Groups do not need to be sequentially positioned in a multi-group partition.

If a system has more than 2 cabinets, every partition can consist of any number of groups; the last group (or remainder of system chassis that is not part of a full 6-chassis group) in the system should be considered a group whether it is fully-populated or not in this partitioning context.

Single Group, Multiple-chassis Systems

When a system contains between two and six fully-populated chassis, then the administrator can partition the system at a per-chassis level of granularity. Each partition must be at least one full chassis, and a chassis cannot be shared between partitions. Chassis do not need to be sequentially positioned in a multi-chassis partition.

Single Chassis Systems

When a system is composed of a single fully-populated chassis, each slot must be in the same partition with its corresponding even/odd pair, because even/odd pair nodes (for example, slot 0 and slot 1, or slot 8 and slot 9) share optical connections and therefore must be in the same partition.

There are 16 slots (or blades) in a single chassis, making 8 even/odd slot pairs, and a maximum of 8 partitions. Single chassis systems can have any combination of even/odd slot pairs (e.g., 4-4, 6-2, 4-2-2, 2-2-1-1-1-1), and even/odd slot pairs do not need to be sequentially positioned in a multiple slot pair partition. In order for a partition to be bootable, it must have a boot node, an SDB node, an I/O node, and a login node.

Configure a Logical Machine

The logical machine can have one of three states:

- `EMPTY` - not configured
- `DISABLED` - configured but not activated
- `ENABLED` - configured and activated

  When a partition is defined, its state changes to `DISABLED`. Undefined partitions are `EMPTY` by default.

The `xtcli part_cfg` command

Use the `xtcli part_cfg` command with the *`part_cmd`* option (`add` in the following example) to identify the operation to be performed and the *`part_option`* (`-m`, `-b`, `-d` and `-i`) to specify the characteristics of the logical machine. The boot image may be a raw device, such as `/raw0`, or a file.

Create a logical machine with a boot node and SDB node specifying the boot image path

- When using a file for the boot image, the same file must be on both the SMW and the `bootroot` at the same path.

- For the logical machine to be bootable, both the boot node and SDB node IDs must be specified.

```
crayadm@smw:~> xtcli part_cfg add p2 -m c0-0,c0-1,c0-2,c0-3 \
-b c0-0c0s0n0 -d c0-0c0s2n1 -i /bootimagedir/bootimage
```

To watch HSS events on the specified partition, execute the `xtconsumer -p partition_name` command.

To display the console text of the specified partition, execute the `xtconsole -p partition_name` command.

For more information, see the `xtcli_part(8)`, `xtconsole(8)`, and `xtconsumer(8)` man pages.

Boot a Logical Machine

The `xtbootsys --partition pN` option enables the administrator to indicate which logical machine (partition) to boot. If a partition name is not specified, the default partition `p0` (component name for the entire system) is booted. Alternatively, if a partition name is not specified and the `CRMS_PARTITION` environment variable is defined, this variable is used as the default partition name. Valid values are in the form `pN`, where *N* ranges from `0` to `31`.

`xtbootsys` manages a link from `/var/opt/cray/log/`*partition*`-current` to the current *sessionid* directory for that partition, allowing changes to `/var/opt/cray/log/p1-current`, for example.

Configure the NFS client to Mount the Exported Lustre File System

About this task

Depending on the site client system, the configuration may be different. This procedure contains general information that will help configure the client system to properly mount the exported Lustre file system. Consult the client system documentation for specific configuration instructions.

Procedure

1. As `root`, verify that the `nfs` client service is started at boot.

2. Add a line to the `/etc/fstab` file to mount the exported file system. The list below describes various recommended file system mount options. For more information on NFS mount options, see the `mount(8)` and `nfs(5) man` pages.

   ```
   server@network:/filesystem /client/mount/point lustre file_system_options 0 0
   ```

   Recommended file system mount options.

rsize=1048576,wsize=1048576	Set the read and write buffer sizes from the server at 1MiB. These options match the NFS read/write transaction to the Lustre filesystem block size, which reduces cache/buffer thrashing on the service node providing the NFS server functionality.
soft,intr	Use a soft interruptible mount request.

async	Use asynchronous NFS I/O. Once the NFS server has acknowledged receipt of an operation, let the NFS client move along even though the physical write to disk on the NFS server has not been confirmed. For sites that need end-to-end write-commit validation, set this option to `sync` instead.
proto=tcp	Force use of TCP transport—this makes the larger `rsize`/`wsize` operations more efficient. This option reduces the potential for UDP retransmit occurrences, which improves end-to-end performance.
relatime,timeo=600,local_lock=none	Lock and time stamp handling, transaction timeout at 10 minutes.
nfsvers=3	Use NFSv3 specifically. NFSv4 is not supported at this time.

3. Mount the file system manually or reboot the client to verify that it mounts correctly at boot.

Repurpose Compute Nodes

When a compute node is configured for a non-compute role, that node is a *repurposed compute node*. Compute nodes can be repurposed to become service nodes for use as tier2 servers (recommended) or in other capacities. Compute nodes should not be repurposed as service nodes for services that require external connectivity, such as dynamically shared libraries (DSL).

> **NOTE:** (SMW HA only) For SMW HA systems, perform this step only on the first SMW. This procedure is not required on the second SMW.

Use the `xtcli mark_node` command to repurpose a node in a compute blade. In this example, two compute nodes are being repurposed as service nodes and marked accordingly in the shared database.

```
crayadm@smw> xtcli mark_node service c0-0c0s2n0,c0-0c0s2n1
```

Note that service nodes can be repurposed as compute nodes as well. In that case, the command would be **`xtcli mark_node compute`**.

Node Attributes

Users can control the selection of the compute nodes on which to run their applications and can select nodes on the basis of desired characteristics (*node attributes*). This allows a placement scheduler to schedule jobs based on the node attributes.

A user invokes the `cnselect` command to specify node-selection criteria. The `cnselect` script uses these selection criteria to query the table of node attributes in the SDB and returns a node list to the user based on the results of the query. When launching the application, the user includes the node list using the aprun -L *`node_list`* option as described on the `aprun(1)` man page. The ALPS placement scheduler allocates nodes based on this list.

To meet specific user needs, the administrator can modify the `cnselect` script. For additional information about the `cnselect` script, see the `cnselect(1)` man page.

View and Temporarily Set Node Attributes

Use the xtprocadmin command to view current node attributes. The xtprocadmin -A option lists all attributes of selected nodes. The xtprocadmin -a attr1,attr2 option lists selected attributes of selected nodes.

An administrator can use the xtprocadmin -a *attr=value* command to temporarily set certain site-specific attributes. Using the xtprocadmin -a *attr=value* command to set certain site-specific attributes is not persistent across reboots. Attribute settings that are intended to be persistent across reboots (such as labels) must be specified in the attr.defaults file.

> **NOTE:** For compute nodes, xtprocadmin changes to attributes require restarting the apbridge daemon on the boot node in order for ALPS to detect changes that the xtprocadmin command has made to the SDB. Restarting the other ALPS components (for example, on the SDB node or on the login node if they are separate nodes) is not necessary. To restart apbridge, log into the boot node as root and execute the following command:
>
> ```
> boot:~ # /etc/init.d/alps restart
> ```

For example, the following command creates a new label1 attribute value for the compute node whose NID is 350. The xtprocadmin command must be executed by root from a service node and the SDB must be running:

```
boot:~ # xtprocadmin -n 350 -a label1=eedept
```

```
Connected
NID    (HEX)    NODENAME    TYPE     LABEL1
350    0x15e    c1-0c1s0n0  compute  eedept
```

Then restart the apbridge daemon on the boot node in order for ALPS to detect changes that the xtprocadmin command has made to the SDB.

```
boot:~ # /etc/init.d/alps restart
```

The XTAdmin Database segment Table

The XTAdmin database contains a segment table that supports the memory affinity optimization tools for applications and CPU affinity options for all Cray compute nodes. The CPU affinity options apply to all Cray multicore compute nodes.

The segment table is similar to the attributes table but differs in that a node may have multiple segments associated with it; the attributes table provides summary information for each node.

In order to address the application launch and placement requirements for compute nodes with two or more NUMA nodes, the Application Level Placement Scheduler (ALPS) requires additional information that characterizes the intranode topology of the system. This data is stored in the segment table of the XTAdmin database and acquired by apbridge when ALPS is started, in much the same way that node attribute data is acquired.

The segment table contains the following fields:

node_id The node identifier that maps to the nodeid field of the attributes table and processor_id field of the processor table.

socket_id Contains a unique ordinal for each processor socket.

die_id	Contains a unique ordinal for each processor die; with this release, `die_id` is `0` in the segment table and is otherwise unused (reserved for future use).
numcores	The number of integer cores per node; in systems with accelerators this only applies to the host processor (CPU).
coremask	The processor core mask. The coremask has a bit set for each core of a CPU. 24-core nodes will have a value of `16777215` (hex `0xFFFFFF`). `coremask` is deprecated and will be removed in a future release.
mempgs	Represents the amount of memory available, in Megabytes, to a single segment.

The `/etc/sysconfig/xt` file contains `SDBSEG` field, which specifies the location of the `segment` table file; by default, `SDBSEG=/etc/opt/cray/sdb/segment`.

To update the `segment` table, use the following service database commands:

- `xtdb2segment`, which converts the data into an ASCII text file that can be edited
- `xtsegment2db`, which writes the data back into the database file

For more information, see the `xtdb2segment(8)` and `xtsegment2db(8)` man pages.

After manually updating the `segment` table, log on to any login node or the SDB node as root and execute the `apmgr resync` command to request that ALPS reevaluate the configuration node segment information and update its information.

If ALPS or any portion of the feature fails in relation to segment scheduling, ALPS reverts to the standard scheduling procedure.

Reuse One or More Previously-failed HSN Links

About this task

To integrate failed links back into the HSN configuration, the `xtwarmswap` command may be invoked with one of the following:

- `-s` *`LCB, ...`*, specifying the list of LCBs to bring back up
- `-s all`, to bring in all available LCBs
- `-s none`, to cause a reroute without changing the LCBs that are in use

Procedure

1. Execute an `xtwarmswap -s` *`LCB_names`* `-p` *`partition_name`* to tell the system to reroute the HSN using the specified set of LCBs in addition to those that are currently in use.

   Doing so will clear the alert flags on the specified LCBs automatically. If the warm swap fails, the alert flag will be restored to the specified LCBs.

2. Execute an `xtwarmswap -s all -p` *`partition_name`* command to tell the system to reroute the HSN using all available links.

The `xtwarmswap` command results in `xtnlrd` performing the same link recovery steps as for a failed link, but with two differences: no alert flags are set, and an `init_new_links` and a `reset_new_links` step are

performed to initialize both ends of any links to be used, before new routes are asserted into the Aries™ routing tables.

The elapsed time for the warm swap synchronization operation is typically about `30` seconds.

Add or Remove a High-speed Network Cable from Service

To specify a high-speed network (HSN) cable to add or remove from service, use the `xtwarmswap --add-cable` *`cable`* command or the `xtwarmswap --remove-cable` *`cable`* command, respectively.

These options provide the ability to replace a single cable without removing blades or shutting down the system. The routing of the Cray HSN will be updated to route around the removed cable.

The `--add-cable` and `--remove-cable` options are not supported if more than a single active partition exists in the system. Do not specify the `-p|--partition` option when using these options. In addition, do not use the `--linktune` option when using the `--remove-cable` option.

Remove a Compute Blade from Service While the System is Running

About this task

A compute blade can be physically removed for maintenance or replacement while the system is running; however, the applications using the nodes on the blade to be removed must be allowed to drain, or be killed beforehand.

Before starting this warm swap procedure, verify that the proposed system configuration is routable. Doing this in advance of idling the nodes on the blades to be removed provides assurance that a valid set of nodes is being taken out of service before affecting the system. Log on to the SMW as `crayadm` and execute the following command, where `p`*`N`* is the partition from which the blades are being removed:

```
smw:~> rtr -S --id test --remove=blade_ID pN
```

CAUTION: This procedure warm swaps a compute blade from service while the system is running. Do not warm swap service blades, unless the blade is an I/O base blade (IBB) that has InfiniBand cards and is an LNET blade. Before attempting to warm swap any service blade, it is advisable to consult with a Cray service representative.

Procedure

1. Log on to the login node as `root`.
2. Ensure that the batch system or Slurm marks the blade as unavailable for scheduling.
3. Execute the following command to mark the nodes on the compute blade as `admindown`. This tells ALPS not to launch new applications onto them. (This command may also be executed from the boot node as user `root`.)

   ```
   login:~ # xtprocadmin -k s admindown -n blade_ID
   ```

The arguments to the `-n` option should be the NID values for the nodes on the blade being removed, as shown by executing `xtprocadmin | grep` *`bladename`*.

For example, to find the NID values for the nodes on the blade `c0-0c0s2` being removed:

```
login:~ # xtprocadmin | grep c0-0c0s2
     8      0x8  c0-0c0s2n0  compute      up       batch
     9      0x9  c0-0c0s2n1  compute      up       batch
    10      0xa  c0-0c0s2n2  compute      up       batch
    11      0xb  c0-0c0s2n3  compute      up       batch
```

4. From the login node, execute the `apstat -n` command or the appropriate Slurm command to determine if any applications are running on the node marked `admindown`. This example shows that `apid 675722` is running on all nodes of blade `c0-0c0s2`.

```
login:~ # apstat -n | egrep -w 'NID|8|9|10|11
  8   XT UP  B 32 32  1    4K 16777216 8388608  262144   1 675722
  9   XT UP  B 32 32  1    4K 16777216 8388608  262144   1 675722
 10   XT UP  B 32 32  1    4K 16777216 8388608  262144   1 675722
 11   XT UP  B 32 32  1    4K 16777216 8388608  262144   1 675722
```

5. Wait until the applications using the nodes on the blade finish or use the `apkill` *`apid`* command to kill the application.

6. Log on to the SMW as `crayadm`.

7. Execute the `xtcli halt` *`blade_ID`* command to halt the blade.

```
smw:~> xtcli halt blade_ID
```

8. Execute the `xtwarmswap --remove` *`blade_ID`* command to remove the compute blade from service. The routing of the Cray HSN will be updated to route around the removed blade.

   The `--remove` stage of the `xtwarmswap` process uses the Aries™ resiliency infrastructure and takes about 30 seconds to complete.

```
smw:~> xtwarmswap --remove blade_ID
```

9. Execute the `xtcli power down` *`blade_ID`* command, which helps to identify which blade to pull (all lights are off on the blade).

```
smw:~> xtcli power down blade_ID
```

10. Physically remove the blade, if desired. To complete this step, see the hardware maintenance and replacement procedures documentation for the Cray system, or contact a Cray Service representative.

    CAUTION: If a blade cannot be reinstalled in the empty slot within 2 minutes, install a filler blade assembly in the empty slot; failure to do so can cause other blades in the system to overheat.

Return a Compute Blade into Service

About this task

After a blade has been repaired or when a replacement blade is available, use the following procedure to return the blade into service.

Procedure

1. Physically insert the blade into the slot. To complete this step, see the hardware maintenance and replacement procedures documentation for the Cray system, or contact a Cray Service representative.

2. On the SMW, execute the `xtcli power up` *`blade_ID`* command.

```
smw:~> xtcli power up blade_ID
```

3. Ensure that the blade is ready by entering the following command, and wait until the command returns the correct response:

```
smw:~> xtalive blade_ID
The expected response was received.
```

4. Verify the status of the blade controller to ensure that its "Comp state" is "up" and that there are no flags set.

```
smw:~> xtcli status -t bc blade_ID
```

5. Bounce the blade.

```
smw:~> xtbounce blade_ID
```

6. If the blade or PDC type is different, `su` to `root`, execute the `xtdiscover` command, and then exit `root`. Otherwise, skip this step.

```
smw:~> su - root
smw:~> xtdiscover
smw:~> exit
smw:~>
```

7. Execute the `xtzap --blade` command to update the BC BIOS, node BIOS, microcontroller, and FPGAs as required.

```
smw:~> xtzap --blade blade_cname
```

8. Execute the `xtbounce --linkdown` *`blade_ID`* command to prepare the blade for the warm swap (takes down all HSN links on the blade).

```
smw:~> xtbounce --linkdown blade_ID
```

9. Add the blade(s) to the HSN by executing the `xtwarmswap --add` *`blade_ID`*`,...` command. This command activates routing on the newly installed blade and automatically executes a mini-`xtdiscover` command once the warm swap steps have completed successfully. No additional manual invocation of `xtdiscover`, which gets the new hardware attributes from the added blades, is necessary.

```
smw:~> xtwarmswap --add blade_ID
```

Because the `xtwarmswap --add` command initializes the added blades, the time to return the blades back to service is about 10 minutes, including the time to initialize the blades, run the BIOS on the nodes, and initialize the links to the blades.

10. Boot the nodes on the blade(s) by executing the `xtcli boot CNL0` *`blade_ID`*`,...` command on the SMW.

```
smw:~> xtcli boot CNL0 blade_ID
```

11. As `root` on the login node, execute the following command to mark the nodes on the compute blade as `up`. This tells ALPS that new applications may be launched onto those nodes. (This command may also be executed from the boot node as user `root`.)

```
login:~ # xtprocadmin -k s up -n blade_ID
```

12. Verify that the blade is up.

```
login:~ # xtprocadmin | grep blade_ID
```

13. Ensure that the batch system or Slurm marks the blade as available for scheduling.

State Manager LLM Logging

The log data from the State Manager is written to `/var/opt/cray/log/sm-`*`yyyymmdd`*. The default setting for the State Manager is to enable LLM logging. If LLM or craylog failures occur, State Manager logging is not disrupted. Logging then reverts to behavior that is very similar to legacy State Manager logging, which is also used when State Manager LLM logging is turned off.

To disable LLM logging for the State Manager, add the `-L n`option to the `/opt/cray/hss/default/bin/rsms` script entry:

```
sm=(/opt/cray/hss/default/bin/state_manager sm "-L n")
```

Boot Manager LLM Logging

The log data from the Boot Manager is written to `/var/opt/cray/log/bm-`*`yyyymmdd`*. If the `-L` command line option is used with the `bootmanager` command or if LLM is not enabled, Boot Manager reverts to legacy logging, which writes log data to `/var/opt/cray/log/bm.out`. This is a less satisfactory logging method because each Boot Manager restart creates a new log and moves the previous log to `bm.out.1`. A third restart can possibly cause recent log data to be lost.

Configure Node Health Checker Tests

NHC is automatically invoked by ALPS upon the termination of an application. ALPS passes a list of CNL compute nodes associated with the terminated application to NHC. NHC performs specified tests to determine if compute nodes allocated to the application are healthy enough to support running subsequent applications. If not, it removes any compute nodes incapable of running an application from the resource pool. The CLE installation

and upgrade processes automatically install and enable NHC software; there is no need to change any installation configuration parameters or issue any commands.

Use the `cray_node_health_worksheet.yaml` file or configurator to configure the NHC tests, which test CNL compute node functionality. All tests that are enabled will run when NHC is in either Normal Mode or in Suspect Mode. Tests run in parallel, independently of each other, except for the `Free Memory Check` test, which requires that the `Application Exited Check` test passes before the `Free Memory Check` test begins.

The `xtcheckhealth` binary runs the NHC tests; for information about the `xtcheckhealth` binary, see the `intro_NHC(8)` and `xtcheckhealth(8)` man pages.

The NHC tests are listed below. In the default NHC configuration file, each test that is enabled starts with an action of `admindown`, except for `Free Memory Check`, which starts with an action of `log`.

Also read important test usage information in *Guidance for the Accelerator Test* on page 185, *Guidance for the Application Exited Check and Apinit Ping Tests* on page 186, *Guidance for the Filesystem Test* on page 186, *Guidance for the Hugepages Test* on page 187, and *Guidance for the NHC Lustre File System Test* on page 187.

Accelerator	Tests the health of any accelerators present on the node. It is an application set test and should not be run in the reservation set. The global accelerator test (`gat`) script detects the type of accelerator(s) present on the node and then launches a test specific to the accelerator type. The test fails if it is unable to run successfully on the accelerator, or if the amount of allocated memory on the accelerator exceeds the amount specified using the `gat -m` argument. Default: enabled
Application Exited Check	Verifies that any remaining processes from the most recent application have terminated. It is an application set test and should not be run in the reservation set because an application is not associated with a reservation cancellation. The `Application Exited Check` test checks locally on the compute node for processes running under the ID of the application (APID). If processes are running, NHC waits a period of time (defined in the configuration file) to determine if the application processes exit properly. If the process does not exit within that time, this test fails. Default: enabled
Apinit Log and Core File Recovery	A plugin script to copy `apinit` core dump and log files to a login/service node. It is an application set test. Default: not enabled. `Apinit Log and Core File Recovery` should not be enabled until a destination directory is determined and specified in the NHC configuration file.
Apinit Ping	Verifies that the ALPS daemon is running on the compute node and is responsive. It is an application set test. The `Apinit Ping` test queries the status of the `apinit` daemon locally on each compute node; if the `apinit` daemon does not respond to the query, then this test fails. Default: enabled
DataWarp	A plugin script to check that any reservation-affiliated DataWarp mount points have been removed. Note that the plugin can only detect a problem after the last reservation on a node completes.

Default: disabled

Free Memory Check Examines how much memory is consumed on a compute node while applications are not running. Use it only as a reservation test because an application within a reservation may leave data for another application in a reservation. If run in the application set, `Free Memory Check` could consider data that was intentionally left for the next application to be leaked memory and mark the node `admindown`. Run the `Free Memory Check` only after the `Reservation` test passes successfully.

Default: enabled (action is `log` only)

Filesystem Ensures that the compute node is able to perform simple I/O to the specified file system. It is configured as an application set test in the default configuration, but it can be run in the reservation set. For a file system that is mounted read-write, the test performs a series of operations on the file system to verify the I/O. A file is created, written, flushed, synced, and deleted. If a mount point is not explicitly specified, the mount point(s) from the compute node `/etc/fstabs` file will be used and a `Filesystem` test will be created for each mount point found in the file. If a mount point is explicitly specified, then only that file system will be checked. An administrator can specify multiple `FileSystem` tests by placing multiple `Filesystem` lines in the configuration file. For example, one line could specify the implicit `Filesystem` test, and the next line could specify a specific file system that does not appear in `/etc/fstab`. This could continue for any and all file systems.

When enabling the `Filesystem` test, an administrator can exclude mount points that should not be tested using the `excluding` setting in the configuration to list mount points that should not be tested by the `Filesystem` test. This allows intentionally excluding specific mount points even though they appear in the `fstab` file. This action prevents NHC from setting nodes to `admindown` because of errors on relatively benign file systems. Explicitly specified mount points cannot be excluded in this fashion; if they should not be checked, then they should simply not be specified.

The `Filesystem` test creates its temporary files in a subdirectory (`.nodehealth.fstest`) of the file system root. An error message is written to the console when the `unlink` of a file created by this test fails.

Default: enabled

Hugepages Calculates the amount of memory available in a specified page size with respect to a percentage of `/proc/boot_freemem`. It is a reservation set test.

This test will continue to check until either the memory clears up or the time-out is reached. The default time-out is 300 seconds.

Default: disabled

Sigcont Plugin Sends a SIGCONT signal to the processes of the current APID. It is an application set test.

Default: disabled

Plugin Allows scripts and executables not built into NHC to be run, provided they are accessible on the compute node. .

Default: disabled so that local configuration settings may be used

ugni_nhc_plugins Tests the User level Gemini Network Interface (uGNI) on compute nodes. It is a reservation set test and an application set test. By extension, testing the uGNI interface also tests the proper operation of parts of the network interface card (NIC). The test sends a datagram packet out to the node's NIC and back again.

Reservation checks for the existence of the `/proc/reservations/`*`rid`* directory, where *`rid`* is the reservation ID. It is a reservation set test, and should not be run in the application set.

If this directory still exists, the test will attempt to end the reservation and then wait for the specified *`timeout`* value for the directory to disappear. If the test fails and Suspect Mode is enabled, NHC enters Suspect Mode. In Suspect Mode, `Reservation` continues running, repeatedly requesting that the kernel clean up the reservation, until the test passes or until Suspect Mode times out. If the directory does not disappear in that time, the test prints information to the console and exits with a failure.

Default: enabled with a *`timeout`* value of 300 seconds

CCM plugin validates the cleanup of a cluster compatibility mode (CCM) environment at the end of a reservation. It is a reservation set test, and it will not run if it is misconfigured as an application test.

This test runs on a compute node only when `/var/crayccm` is detected. The test removes the `/var/lib/{empty,debus}` directories, unmounts CCM mount points if they still exist, and unmounts `/dsl/dev/random` and `/dsl/dev/pts`. If the unmounts are successful, the test removes the `/var/crayccm`, `/var/lib/rpcbind`, and `/var/spool/{PBS,torque}` directories.

The `CCM plugin` is not included in a site's NHC configuration file. Administrators must add the test to their configuration in order to use it. See the `cray_node_health_worksheet.yaml` file for `CCM plugin` settings to copy into a site's NHC configuration file.

Individual tests may appear multiple times in the configuration, with different variable values. Every time a test is specified, NHC will run that test. This means if the same line is specified five times, NHC will try to run that same test five times. This functionality is mainly used in the case of the `Plugin` test, allowing the administrator to specify as many additional tests as have been written for the site, or the `Filesystem` test, allowing the adminstrator to specify as many additional file systems as wanted. However, any test can be specified to run any number of times. Different parameters and test actions can be set for each test. For example, this could be used to set up hard limits and soft limits for the `Free Memory Check` test. Two `Free Memory Check` tests could be specified in the configuration file; the first test configured to only warn about small amounts of non-free memory, and the second test configured to `admindown` a node that has large amounts of non-free memory. See the `cray_node_health_worksheet.yaml` file for configuration information.

Guidance for the Accelerator Test

This test uses the global accelerator test (`gat`) script (`/opt/cray/nodehealth/default/bin/gat.sh`) to first detect the accelerator type and then launch the test specific to that type of accelerator.

The `gat` script supports two arguments for NVIDIA GPUs:

-m*maximum_memory_size* Specify the *`maximum_memory_size`* as either a kilobyte value or a percentage of total memory. For example, `-m 100` specifies that no more than 100 kilobytes of memory can be allocated, while `-m 10%` specifies that no more than 10 percent of memory can be allocated.

	In the default NHC configuration file, the specified memory size is `10%`.
`-r`	Perform a soft restart on the GPU and then rerun the test. In the default NHC configuration file, the `-r` argument is specified.

The `gat` script has the following options for Intel Xeon Phi:

`-M`*`kilobytes`* or **`-M`*`n`*`%`	This option works exactly as the `-m` option for the NVIDIA GPUs.
`-c`	Specifies the minimum number of cores that must be active on the Xeon Phi for the test to pass. If `-c` is omitted, the minimum number of active cores required to pass the test is the total number of cores on the Xeon Phi.

Guidance for the Application Exited Check and Apinit Ping Tests

These two tests must be enabled and both tests must have their action set as `admindown` or `die`; otherwise, NHC runs the risk of allowing ALPS to enter a live-lock. Only specify the `die` action when the `advanced_features` control variable is turned off.

ALPS must guarantee two conditions about the nodes in a reservation before releasing that reservation:

- that ALPS is functioning on the nodes
- that the previous application has exited from the nodes

Either those two conditions are guaranteed or the nodes must be set to some state other than `up`. When either ALPS has guaranteed these two conditions about the nodes or the nodes have been set to some state other than `up`, then ALPS can release the reservation.

These NHC tests guarantee two conditions:

- `Apinit_ping` guarantees that ALPS is functioning on the nodes
- `Application_Exited_Check` guarantees that the previous application has exited from the nodes

If either test fails, then NHC sets the nodes to `suspect` state if Suspect Mode is enabled; otherwise, NHC sets the nodes to `admindown` or `unavail`. Therefore, either the nodes pass these tests or the nodes are no longer in the `up` state. In either case, ALPS is free to release the reservation and the live-lock is avoided. Note that this only happens if the two tests are enabled and their action is set as `admindown` or `die`. The `log` action does not suffice because it does not change the state of the nodes. If either test is disabled or has an action of `log`, then ALPS may live-lock. In this live-lock, ALPS will call NHC endlessly.

Guidance for the Filesystem Test

The NHC `Filesystem` test can take an explicit argument (the mount point of the file system) or no argument. If an argument is provided, the `Filesystem` test is referred to as an explicit `Filesystem` test. If no argument is given, the `Filesystem` test is referred to as an implicit `Filesystem` test.

The explicit `Filesystem` test will test the file system located at the specified mount point.

The implicit `Filesystem` test will test each file system listed in the `/etc/fstab` file on each compute node. The implicit `Filesystem` test is enabled by default in the NHC configuration file.

The `Filesystem` test will determine whether a file system is mounted read-only or read-write. If the file system is mounted read-write, then NHC will attempt to write to it. If it is mounted read-only, then NHC will attempt to read the directory entities "." and ".." in the file system to guarantee, at a minimum, that the file system is readable.

Some file systems are mounted on the compute nodes as read-write file systems, while their underlying permissions are read-only. As an example, for an auto-mounted file system, the base mount-point may have read-

only permissions; however, it could be mounted as read-write. It would be mounted as read-write, so that the auto-mounted sub-mount-points could be mounted as read-write. The read-only permissions prevent tampering with the base mount-point. In a case such as this, the `Filesystem` test would see that the base mount-point had been mounted as a read-write file system. The `Filesystem` test would try to write to this file system, but the write would fail due to the read-only permissions. Because the write fails, the `Filesystem` test would fail, and NHC would incorrectly decide that the compute node is unhealthy because it could not write to this file system. For this reason, file systems that are mounted on compute nodes as read-write file systems, but are in reality read-only file systems, should be excluded from the implicit `Filesystem` test.

The administrator can exclude tests by adding an "Excluding: *`file system mount point`*" entry in the NHC configuration file. See the NHC configuration file for further details and an example.

A file system is deemed a critical file system if it is needed to run applications. All systems will likely need at least one shared file system for reading and writing input and output data. Such a file system would be a critical file system. File systems that are not needed to run applications or read and write data would be deemed as noncritical file systems. The administrator must determine the criticality of each file system.

Cray recommends the following:

- Exclude noncritical file systems from the implicit `Filesystem` test. See the NHC configuration file for further details and an example.
- If there are critical file systems that do not appear in the `/etc/fstab` file on the compute nodes (such file systems would not be tested by the implicit `Filesystem` test), these critical file systems should be checked through explicit `Filesystem` tests. Add explicit `Filesystem` tests to the NHC configuration file by providing the mount point of the file system as the final argument to the `Filesystem` test. See the NHC configuration file for further details and an example.
- If a file system that is mounted as read-write but it has read-only permissions, exclude it from the implicit `Filesystem` test. NHC does not support such file systems.

Guidance for the Hugepages Test

The `Hugepages` test runs the `hugepages_check` command, which supports two arguments:

`-t` *`threshold`* Use this argument to specify the *`threshold`* as a percentage of `/proc/boot_freemem`. If this test is enabled and this argument is not supplied, the default of `-t 90` is used.

`-s` *`size`* Specify the hugepage size. The valid sizes are `2`, `4`, `8`, `16`, `32`, `64`, `128`, `256`, and `512`. If this test is enabled and this argument is not supplied, the default of `-s 2` is used.

Guidance for the NHC Lustre File System Test

The Lustre file system has its own hard time-out value that determines the maximum time that a Lustre recovery will last. This time-out value is called `RECOVERY_TIME_HARD`, and it is located in the file system's `fs_defs` file. The default value for the `RECOVERY_TIME_HARD` is 15 minutes.

> **IMPORTANT:** The time-out value for the NHC `Lustre` file system test should be twice the `RECOVERY_TIME_HARD` value.

The default in the NHC configuration file is 30 minutes, which is twice the default `RECOVERY_TIME_HARD`. If the value of `RECOVERY_TIME_HARD` is changed, the time-out value of the NHC `Lustre` file system test must also change correspondingly.

The NHC time-out value is specified on the following line in the NHC configuration file:

```
# Lustre: <warning time-out> <test time-out> <restart delay>
Lustre: 900 1800 60
```

Further, the overall time-out value of NHC's Suspect Mode is based on the maximum time-out value for all of the NHC tests. Invariably, the NHC `Lustre` file system test has the longest time-out value of all the NHC tests.

> **IMPORTANT:** If the NHC `Lustre` file system test time-out value is changed, then the time-out value for Suspect Mode must also be changed. The time-out value for Suspect Mode is set by the `suspectend` variable in the NHC configuration file. The guidance for setting the value of `suspectend` is that it should be the maximum time-out value, plus an additional buffer. In the default case, `suspectend` was set to 35 minutes -- 30 minutes for the Lustre test, plus an additional 5 minute buffer. For more information about the `suspectend` variable, see *NHC Suspect Mode*.

NHC Control Variables

The following variables in `/etc/opt/cray/nodehealth/nodehealth.conf` affect the fundamental behavior of NHC.

`advanced_features` If set to `on`, this variable allows multiple instances of NHC to run simultaneously. This variable must be `on` to use CNCU and reservation sets.

Default: `on`

`dumpdon` If set to `off`, NHC will not request any dumps or reboots from `dumpd`. This is a quick way to turn off dump and reboot requests from NHC. The `dump`, `reboot`, and `dumpreboot` actions do not function properly when this variable is `off`.

Default: `on`

`anyapid` Turning `anyapid` on specifies that NHC should look for any `apid` in `/dev/cpuset` while running the `Application Exited Check` and print stack traces for processes that are found.

Default: `off`

Global Configuration Variables that Affect all NHC Tests

The following global configuration variables may be set in the `/etc/opt/cray/nodehealth/nodehealth.conf` file to alter the behavior of all NHC tests. The global configuration variables are case-insensitive.

`Runtests: Frequency` Determines how frequently NHC tests are run on the compute nodes. *`Frequency`* may be either `errors` or `always`. When the value `errors` is specified, the NHC tests are run only when an application terminates with a non-zero error code or terminates abnormally. When the value `always` is specified, the NHC tests are run after every application termination. If the `Runtests` global variable is not specified, the implicit default is `errors`.

This variable applies only to tests in the application set; reservations do not terminate abnormally.

`Connecttime: TimeoutSeconds` Specifies the amount of time, in seconds, that NHC waits for a node to respond to requests for the TCP connection to be established. If Suspect Mode is disabled and a

particular node does not respond after `connecttime` has elapsed, then the node is marked `admindown`. If Suspect Mode is enabled and a particular node does not respond after `connecttime` has elapsed, then the node is marked `suspect`. NHC will then attempt to contact the node with a frequency established by the `recheckfreq` variable.

If the `Connecttime` global variable is not specified, then the implicit default TCP time-out value is used. NHC will not enforce time-out on the connections if none is specified. The `Connecttime: TimeoutSeconds` value provided in the default NHC configuration file is `60` seconds.

The following global variables control the interaction of NHC and `dumpd`, the SMW daemon that initiates automatic dump and reboot of nodes.

`maxdumps: MaximumNodes`
Specifies the number of nodes that fail with the `dump` or `dumpreboot` action that will be dumped. For example, if NHC was checking on 10 nodes that all failed tests with the `dump` or `dumpreboot` actions, only the number of nodes specified by `maxdumps` would be dumped, instead of all of them. The default value is `1`.

To disable dumps of failed nodes with `dump` or `dumpreboot` actions, set `maxdumps: 0`.

`downaction: action`
Specifies the action NHC takes when it encounters a down node. Valid actions are `log`, `dump`, `reboot`, and `dumpreboot`. The default action is `log`.

`downdumps: number_dumps`
Specifies the maximum number of dumps that NHC will dump for a given APID, assuming that the `downaction` variable is either `dump` or `dumpreboot`. These dumps are in addition to any dumps that occur because of NHC test failures. The default value is `1`.

The following global variables control the interaction between NHC, ALPS, and the SDB.

`alps_recheck_max: number of seconds`
NHC will attempt to verify its view of the states of the nodes with the ALPS view. If NHC is unable to contact ALPS, this variable controls the maximum delay between rechecks.

Default value: 10 seconds

`alps_sync_timeout: number of seconds`
If NHC is unable to contact ALPS to verify the states of the nodes, this variable controls the length of time before NHC gives up and aborts.

Default value: 1200 seconds

`alps_warn_time: number of seconds`
If NHC is unable to contact ALPS to verify the states of the nodes, this variable controls how often warnings are issued.

Default value: 120 seconds

`sdb_recheck_max: number of seconds`
NHC will contact the SDB to query for the states of the nodes. If NHC is unable to contact the SDB, this variable controls the maximum delay between rechecks.

Default value: 10 seconds

`sdb_warn_time: number of seconds`
If NHC is unable to contact the SDB, this variable controls how often warnings are issued.

Default value: 120 seconds

node_no_contact_warn_time: ***number of seconds***
: If NHC is unable to contact a specific node, this variable controls how often warnings are issued.

  Default value: 600 seconds

The following global variable controls NHC's use of node states.

unhealthy_state: ***swdown***
: When a node is deemed unhealthy, it is normally set to `admindown`. This variable permits a different state to be chosen instead.

  Default: not set

unhealthy_state: ***rebootq***
: When a node is going to be rebooted, it is normally set to `Unavail`. This variable permits a different state to be chosen instead.

  Default: not set

Standard Variables that Affect Individual NHC Tests

The following variables are used with each NHC test; set each variable for each test. All variables are case-insensitive. Each NHC test has values supplied for these variables in the default NHC configuration file.

Specific NHC tests require additional variables, which are defined in the `nodehealth` configuration file.

action
: Specifies the action to perform if the compute node fails the given NHC test. *`action`* may have one of the following values:

  log
  : Logs the failure to the system console log. The `log` action will not cause a compute node's state to be set to `admindown`.

    IMPORTANT: Tests that have an action of `Log` do not run in Suspect Mode. If using plugin scripts with an action of `Log`, the script will only be run once, in Normal Mode. This makes log collecting and various other maintenance tasks easier to code.

  admindown
  : Sets the compute node's state to `admindown` (no more applications will be scheduled on that node) and logs the failure to the system console log.

    If Suspect Mode is enabled, the node will first be set to `suspect` state, and if the test continues to fail, the node will be set to `admindown` at the end of Suspect Mode.

  die
  : Halts the compute node so that no processes can run on it, sets the compute node's state to `admindown`, and logs the failure to the system console log. (The `die` action is the equivalent of a kernel panic.) This action is good for catching bugs because the state of the processes is preserved and can be dumped at a later time.

    If the `advanced_features` variable is enabled, `die` is not allowed.

    Each subsequent action includes the actions that preceded it; for example, the `die` action encompasses the `admindown` and `log` actions.

    If NHC is running in Normal Mode and cannot contact a compute node, and if Suspect Mode is not enabled, NHC will set the compute node's state to `admindown`.

The following actions control the NHC and `dumpd` interaction.

`dump` Sets the compute node's state to `admindown` and requests a dump from the SMW, in accordance with the `maxdumps` configuration variable.

`reboot` Sets the compute node's state to `unavail` and requests a reboot from the SMW. The `unavail` state is used rather than the `admindown` state when nodes are to be rebooted because a node that is set to `admindown` and subsequently rebooted stays in the `admindown` state. The `unavail` state does not have this limitation.

`dumpreboot` Sets the compute node's state to `unavail` and requests a dump and reboot from the SMW.

The following actions control the NHC and `dumpd` interaction.

`warntime` Specifies the amount of elapsed test time, in seconds, before `xtcheckhealth` logs a warning message to the console file. This allows an administrator to take corrective action, if necessary, before the *`timeout`* is reached.

`timeout` Specifies the total time, in seconds, that a test should run before an error is returned by `xtcheckhealth` and the specified *`action`* is taken.

`restartdelay` Valid only when NHC is running in Suspect Mode. Specifies how long NHC will wait, in seconds, to restart the test after the test fails. The minimum restart delay is one second.

`sets` Indicates when to run a test. The default NHC configuration specifies to run specific tests after application completion and to run an alternate group of tests at reservation end. When ALPS calls NHC at the end of the application, tests marked with `Sets: Application` are run. By default, these tests are: `Filesystem`, `Accelerator`, `ugni_nhc_plugins`, `Application Exited Check`, `Apinit Ping Test`, and `Apinit Log and Core File Recovery`. At the end of the reservation, ALPS calls tests marked `Sets: Reservation`. By default, these are: `Free Memory Check`, `ugni_nhc_plugins`, `Reservation`, and `Hugepages Check`.

If no set is specified for a test, it will default to `Application`, and run when ALPS calls NHC at the end of the application. If NHC is launched manually, using the `xtcheckhealth` command, and the `-m` *`sets`* argument is not specified on the command line, then `xtcheckhealth` defaults to running the `Application` set.

If a test is marked `Sets: All`, it will always run, regardless of how NHC is invoked.

NHC Suspect Mode

Upon entry into Suspect Mode, NHC immediately allows healthy nodes to be returned to the resource pool. Suspect Mode allows the remaining nodes, which are all in `suspect` state, an opportunity to return to healthiness. If the nodes do not return to healthiness by the end of the Suspect Mode (determined by the `suspectend` global variable; see below), their states are set to `admindown`. For more information about how Suspect Mode functions, see the `intro_NHC(8)` man page.

> **IMPORTANT:** Suspect Mode is enabled in the default configuration. Cray recommends that sites run NHC with Suspect Mode enabled.

If enabled, the default NHC configuration file uses the following Suspect Mode variables:

`suspectenable:` Enables Suspect Mode; valid values are `y` and `n`.

Default: `y`

suspectbegin: Sets the Suspect Mode timer. Suspect Mode starts after the number of seconds indicated by `suspectbegin` have expired.

Default: `180`

suspectend: Suspect Mode ends after the number of seconds indicated by `suspectend` have expired. This timer only starts after NHC has entered Suspect Mode.

Default: `2100`

Considerations when evaluating shortening the length of Suspect Mode:

- The length of Suspect Mode can be shortened if there are no external file systems, such as Lustre, for NHC to check.
- Cray recommends that the length of Suspect Mode be at least a few seconds longer than the longest time-out value for any of the NHC tests. For example, if the `Filesystem` test had the longest time-out value at 900 seconds, then the length of Suspect Mode should be at least 905 seconds.
- The longer the Suspect Mode, the longer nodes have to recover from any unhealthy situations. Setting the length of Suspect Mode too short reduces this recovery time and increases the likelihood of the nodes being marked `admindown` prematurely.

recheckfreq: Suspect Mode rechecks the health of the nodes in `suspect` state at a frequency specified by `recheckfreq`. This value is in seconds.

For a detailed description about NHC actions during the recheck process, see the `intro_NHC(8)` man page.

Default: `300`

NHC Messages

NHC messages may be found on the SMW in `/var/opt/cray/log/`*`sessionid`*`/nhc-`*`YYYYMMDD`* with '<node_health:*M.m*>' in the message, where *M* is the major and *m* is the minor NHC revision number. All NHC messages are visible in the console file.

NHC prints a summary message per node at the end of Normal Mode and Suspect Mode when at least one test has failed on a node. For example:

```
<node_health:3.1> APID:100 (xtnhc) FAILURES: The following tests have failed in
normal mode:
<node_health:3.1> APID:100 (xtnhc) FAILURES: (Admindown) Apinit_Ping
<node_health:3.1> APID:100 (xtnhc) FAILURES: (Admindown) Plugin /example/plugin
<node_health:3.1> APID:100 (xtnhc) FAILURES: (Log Only ) Filesystem_Test on /
mydir
<node_health:3.1> APID:100 (xtnhc) FAILURES: (Admindown) Free_Memory_Check
<node_health:3.1> APID:100 (xtnhc) FAILURES: End of list of 5 failed test(s)
```

The `xtcheckhealth` error and warning messages include node IDs and application IDs and are written to the console file on the SMW; for example:

```
[2010-04-05 23:07:09][c1-0c2s0n0]<node_health:3.0> APID:2773749
(check_apid) WARNING: Failure: File /dev/cpuset/2773749/tasks exists and is not
empty. \
The following processes are running under expired APID
2773749:
```

```
[2010-04-05 23:07:09][c1-0c2s0n1]<node_health:3.0> APID:2773749
(check_apid) WARNING: Pid:   300 Name:           (marys_program) State: D
```

The `xtcleanup_after` script writes its normal launch information to the `/var/log/xtcheckhealth_log` file, which resides on the login nodes. The `xtcleanup_after` launch information includes the time that `xtcleanup_after` was launched and the time `xtcleanup_after` called `xtcheckhealth`.

The `xtcleanup_after` script writes error output (launch failure information) to the `/var/log/xtcheckhealth_log` file, to the console file on the SMW, and to the syslog.

Example `xtcleanup_after` output follows:

```
Thu Apr 22 17:48:18 CDT 2010 <node_health> (xtcleanup_after)
/opt/cray/nodehealth/3.0-1.0000.20840.30.8.ss/bin/xtcheckhealth -a 10515
-e 1 /tmp/apsysLVNqO9 /etc/opt/cray/nodehealth/nodehealth.conf
```

Recover from a Login Node Crash when a Login Node will not be Rebooted

About this task

If a login node crashes while `xtcheckhealth` binaries on that login node are monitoring compute nodes that are in `suspect` state, those `xtcheckhealth` binaries will die when the login node crashes. When the login node that crashed is rebooted, a recovery action takes place. When the login node boots, the `node_health_recovery` binary starts up. This script checks for all compute nodes that are in `suspect` state and were last set to `suspect` state by this login node. The script then determines the APID of the application that was running on each of these compute nodes at the time of the crash. The script then launches an `xtcheckhealth` binary to monitor each of these compute nodes. One `xtcheckhealth` binary is launched per compute node monitored.

If the `Application_Exited_Check` test is enabled in the configuration file (default), `xtcheckhealth` is launched with this APID to test for processes that may have been left behind by that application. Otherwise, NHC does not run the `Application_Exited_Check` test and will not check for leftover processes, but will run any other NHC tests that are enabled in the configuration file.

Nodes are changed from `suspect` state to `up` or `admindown`, depending upon whether they fail any health checks. No system administrator intervention should be necessary.

NHC automatically recovers the nodes in `suspect` state when the crashed login node is rebooted because the recovery feature runs on the rebooted login node. If the crashed login node is not rebooted, then manual intervention is required to rescue the nodes from `suspect` state. This manual recovery can commence as soon as the login node has crashed. To recover from a login node crash during the case in which a login node will not be rebooted, the `nhc_recovery` binary is provided to help release the compute nodes owned by the crashed login node; see *Recover from a Login Node Crash when a Login Node will not be Rebooted*. Also, see the `nhc_recovery(8)` man page for a description of the `nhc_recovery` binary usage.

Procedure

1. Create a file, *`nodelistfile`*, that contains a list of the nodes in the system that are currently in Suspect Mode. The file must be a list of NIDs, one per line; do not include a blank line at the end of the file.

2. List all of the `suspect` nodes in the system and the login nodes to which they belong.

```
smw:~# nhc_recovery -d nodelistfile
```

3. Parse the `nhc_recovery` output for the NID of the login node that crashed, creating a file, *`computenodes`*, that contains all of the compute nodes owned by the crashed login node.

4. Use the *`computenodes`* file to create *`nodelist`* files containing nodes that share the same APID (to determine the nodes from the crashed login node). For example, the files can be named `nodelistfile-APID1`, `nodelistfile-APID2`, `nodelistfile-APID3`, etc.

5. Release all of the `suspect` compute nodes owned by the crashed login node.

   ```
   smw:~# nhc_recovery -r computenodes
   ```

   All of these compute nodes are released in the database, but they are all still in `suspect` state.

6. Determine what to do with these `suspect` nodes from the following three options:

   - (Cray recommends this option) Rerun NHC on a non-crashed login node to recover the nodes listed in step *4* on page 194. Invoke NHC for each `nodelist-APID` file. Supply the APID that corresponds to the *`nodelistfile`*; an iteration count of `0` (zero), which is the value normally supplied to NHC by ALPS; and an application exit code of `1` as the APID argument. An exit code of `1` ensures that NHC will run regardless of the value of the `runtests` variable (`always` or `errors`) in the NHC configuration file. For example:

     ```
     smw:~# xtcleanup_after -s nodelist-APID1  APID1  0  1
     smw:~# xtcleanup_after -s nodelist-APID2  APID2  0  1
     smw:~# xtcleanup_after -s nodelist-APID3  APID3  0  1
     .
     .
     .
     ```

   - Set the `suspect` nodes to `admindown` and determine their fate by further analysis.
   - Set the `suspect` nodes back to `up`, keeping in mind that they were in Suspect Mode for a reason.