Linux on IBM Z and LinuxONE

Device Drivers, Features, and Commands on Red Hat Enterprise Linux 9.1

SC34-7750-00

Note

Before using this document, be sure to read the information in “Notices” on page 807.

This edition applies to Red Hat® Enterprise Linux® 9.1 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this publication

This publication describes the device drivers, features, and commands available to Red Hat Enterprise Linux 9.1 for the control of IBM Z® devices and attachments. Unless stated otherwise, in this book the terms *device drivers* and *features* are understood to refer to device drivers and features for Red Hat Enterprise Linux 9.1 for IBM Z.

For details about IBM® tested Linux environments, see www.ibm.com/systems/z/os/linux/resources/testedplatforms.html.

Unless stated otherwise, all IBM z/VM® related information in this document assumes a current z/VM version, see www.vm.ibm.com/techinfo/lpmigr/vmleos.html.

For a support matrix, see the Capabilities and Limits section in the Red Hat Enterprise Linux 9.1 release notes at https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux. The same Web page contains Technical Notes with details on various features and known issues.

You can find the newest version of this publication at https://www.ibm.com/docs/en/linux-on-systems?topic=distributions-red-hat-enterprise-linux

How this publication is organized

The first part of this publication contains general and overview information for the z/Architecture® specific device drivers for Red Hat Enterprise Linux 9.1 for IBM Z.

Part two contains chapters about device drivers and features that are used in the context of booting and shutting down Linux.

Part three contains chapters specific to individual storage device drivers.

Part four contains chapters specific to individual network device drivers.

Part five contains chapters about device drivers and features that help to manage the resources of the real or virtual hardware.

Part six contains chapters that describe device drivers and features in support of z/VM virtual server integration.

Part seven contains chapters that describe device drivers and features in support of KVM virtual server integration. Topics cover both Linux as a KVM host and Linux as a KVM guest.

Part eight contains chapters about device drivers and features that support security aspects of Red Hat Enterprise Linux 9.1 for IBM Z.

Part nine contains chapters about assessing the performance of Red Hat Enterprise Linux 9.1 for IBM Z.

Part ten contains chapters about device drivers and features that are used in the context of diagnostics and problem solving.

Part eleven contains chapters with reference information about commands, kernel parameters, and Linux use of z/VM DIAG calls.

Who should read this publication

Most of the information in this publication is intended for system administrators who want to configure Red Hat Enterprise Linux 9.1 for IBM Z.

The following general assumptions are made about your background knowledge:

- You have an understanding of basic computer architecture, operating systems, and programs.

- You have an understanding of Linux and IBM Z terminology.
- You are familiar with Linux device driver software.
- You are familiar with the IBM Z devices attached to your system.

Programmers: Some sections are of interest primarily to specialists who want to program extensions to the device drivers and features.

Hypervisor-specific information

This publication provides information for Linux in LPAR mode and for Linux as a guest of z/VM or KVM.

Information in this publication applies to all hypervisor environments, unless indicated otherwise. Parts and chapters that do not apply to all environments state this exception at the beginning. Lesser differences between environments are detailed within the description.

Linux in LPAR mode

Processor Resource/Systems Manager (PR/SM) technology always divides IBM Z hardware resources into one or more logical partitions (LPARs). Linux in LPAR mode runs directly in an LPAR.

Linux on z/VM

The z/VM hypervisor is an IBM Z operating system that can run in an LPAR and provide IBM Z virtual machines. Linux on z/VM runs as a guest in a z/VM guest virtual machine.

z/VM virtualizes z/Architecture and IBM Z devices to its guests, so most of what applies to Linux in LPAR mode also applies to Linux as a z/VM guest.

Depending on your z/VM virtual machine definition and on your z/VM version and service level, a particular z/VM guest might not provide all of the described features. See ibm.com/vm/newfunction/index.html about upcoming and available new function for z/VM.

For information that exclusively applies to Linux on z/VM, see Part 6, “z/VM virtual server integration,” on page 407.

Linux on KVM

Linux in LPAR mode can be set up as a KVM host that provides KVM virtual servers. Linux on KVM runs as a guest in a KVM virtual server. For information about managing KVM virtual servers, see the documentation of your KVM host distribution and *KVM Virtual Server Management*, SC34-2752.

KVM virtual servers on IBM Z present many mainframe devices as generalized virtio devices to their guests. To Linux, these virtio devices resemble virtio devices on other hardware architectures more than the underlying mainframe devices. Therefore, entire parts and chapters of this publication do not apply to Linux on KVM.

Depending on your KVM host and on your virtual server configuration, a particular KVM guest might not provide all of the described features.

For information that exclusively applies to Linux on KVM, see Part 7, “KVM virtual server integration,” on page 455.

Conventions and assumptions used in this publication

This section summarizes the styles, highlighting, and assumptions used throughout the publication.

Authority

Most of the tasks described in this document require a user with root authority. In particular, writing to procfs, and writing to most of the described sysfs attributes requires root authority.

Throughout this document, it is assumed that you have root authority.

Making changes persistent

This document describes how to change settings and options for mainframe computers in sysfs. In most cases, changes in sysfs are not persistent. If you need to make your changes persistent, see *Configuring basic system settings* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_basic_system_settings/) for details about the configuration files to use.

This document describes how to load modules with **modprobe**. Loading a module this way is not persistent across reboots. If you want to load your kernel modules automatically at boot time, see the section on persistent module loading in *Configuring basic system settings* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_basic_system_settings/).

Terminology

In this document, the term *booting* is used for running boot loader code that loads the Linux operating system. *IPL* is used for issuing an IPL command to load boot loader code or a stand-alone dump utility. See also "IPL and booting" on page 93.

sysfs and procfs

In this document, the mount point for the virtual Linux file system sysfs is assumed to be `/sys`. Correspondingly, the mount point for procfs is assumed to be `/proc`.

debugfs

This document assumes that debugfs has been mounted at `/sys/kernel/debug`.

To mount debugfs, you can use this command:

```
# mount -t debugfs none /sys/kernel/debug
```

Documentation directory

This document sometimes refers to files in the Documentation directory in the Linux source tree.

On Red Hat Enterprise Linux 9.1 the full path to this directory is:

```
/usr/share/doc/kernel-doc-<version>/Documentation
```

If this directory is not present, install the `kernel-doc-<version.el7>.noarch` RPM.

Number prefixes

In this publication, KB means 1024 bytes, MB means 1,048,576 bytes, and GB means 1,073,741,824 bytes.

Hexadecimal numbers

Mainframe documents and Linux documents tend to use different styles for writing hexadecimal numbers. Thirty-one, for example, would typically read X'1F' in a mainframe book and 0x1f in a Linux book.

Because the Linux style is required in many commands and is also used in some code samples, the Linux style is used throughout this publication.

Highlighting

This document uses the following highlighting styles:

- Paths and URLs are highlighted in `monospace`.
- Variables are highlighted in *<italics within angled brackets>*.
- Commands in text are highlighted in **`monospace bold`**.
- Input and output as normally seen on a computer screen is shown

```
within a screen frame.
Prompts are shown as hash signs:
#
```

Part 1. General concepts

This information at an overview level describes concepts that apply across different device drivers and kernel features.

Newest version

You can find the newest version of this book at ibm.com/docs/en/linux-on-systems?topic=linuxone-distributions

Chapter 1. How devices are accessed by Linux

Applications on Linux access character and block devices through device nodes, and network devices through network interfaces.

Device names, device nodes, and major/minor numbers

The Linux kernel represents character and block devices as pairs of numbers *<major>*:*<minor>*.

Some major numbers are reserved for particular device drivers. Other major numbers are dynamically assigned to a device driver when Linux boots. For example, major number 94 is always the major number for DASD devices while the device driver for channel-attached tape devices has no fixed major number. A major number can also be shared by multiple device drivers. See `/proc/devices` to find out how major numbers are assigned on a running Linux instance.

The device driver uses the minor number *<minor>* to distinguish individual physical or logical devices. For example, the DASD device driver assigns four minor numbers to each DASD: one to the DASD as a whole and the other three for up to three partitions.

Device drivers assign device names to their devices, according to a device driver-specific naming scheme (see, for example, "DASD naming scheme" on page 142). Each device name is associated with a minor number (see Figure 1 on page 3).

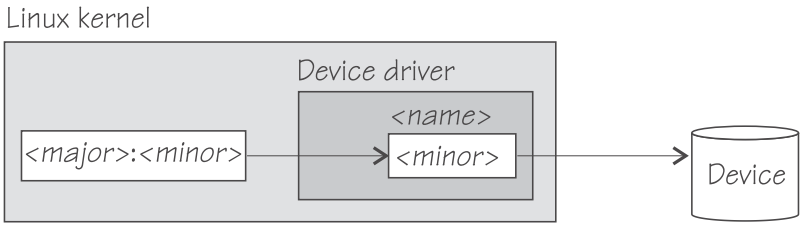


Figure 1. Minor numbers and device names

User space programs access character and block devices through *device nodes* also referred to as *device special files*. When a device node is created, it is associated with a major and minor number (see Figure 2 on page 3).

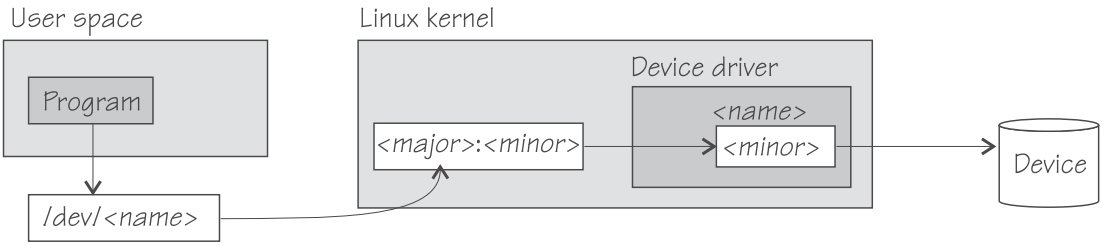


Figure 2. Device nodes

Red Hat Enterprise Linux 9.1 uses udev to create device nodes for you. There is always a device node that matches the device name that is used by the kernel, and additional nodes might be created by special udev rules. See the udev man page for more details.

Network interfaces

The Linux kernel representation of a network device is an interface.

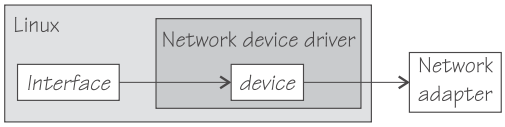


Figure 3. Interfaces

When a network device is defined, it is associated with a real or virtual network adapter (see Figure 3 on page 4). You can configure the adapter properties for a particular network device through the device representation in sysfs (see "Device directories" on page 9).

You activate or deactivate a connection by addressing the interface with **ip** or an equivalent command. All interfaces that are provided by the IBM Z specific network device drivers are interfaces for the Internet Protocol (IP).

On Red Hat Enterprise Linux, you configure network interfaces through the NetworkManager. Use the **nmcli** command line tool to interact with the NetworkManager.

Predictable network interface names

Red Hat Enterprise Linux uses predictable network interface names. These names are stable across reboots and network adapter replacements.

On Red Hat Enterprise Linux 9.1, RoCE devices follow the naming scheme described in "Network interface names" on page 351.

Predictable naming is enabled by default. In this naming scheme, a mainframe network interface name has the following form:

```
<pf><type><bus_id>
```

For example:

```
encf5f0
```

Where:

<pf>
A two-character prefix for the network type. The type can be one of the following:

`en` - Ethernet
`ww` - WAN
`sl` - serial line, such as CTC

<type>
The device type. The device type of channel command word (CCW) devices is `c`. For PCIe devices, the type is `s` or `o`, see "Network interface names" on page 351. For an introduction to mainframe devices in Linux, see "Device categories" on page 7.

<bus_id>
For predictable network names, Red Hat Enterprise Linux 9.1 shortens the device bus-ID by omitting all dots and leading zeroes.

Examples:

- `0.0.0b01` becomes `encb01`
- `0.1.00cd` becomes `enc100cd`

You can use **lszdev -a** or **lscss -a** to obtain a list of the devices in your system.

For more information about the predictable naming scheme, see the *Configuring and managing networking in Red Hat Enterprise Linux 9.1* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_and_managing_networking/index) available at

```
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux
```

Matching devices with the corresponding interfaces

If you define multiple interfaces on a Linux instance, you must keep track of the interface names assigned to your network devices.

Red Hat Enterprise Linux 9.1 uses predictable network names, which make it easy to match network devices with their interface names. The interface names are preserved across reboots.

How to keep track of the mapping between interface names and devices depends on the network device driver. For qeth, you can use the **lszdev qeth** command (see "lszdev - Display IBM Z device configurations" on page 688) to obtain a mapping.

After setting a device online, issue **journalctl** to find the associated interface name in the messages that are issued in response to the device being set online.

For each network device that is online, there is a symbolic link of the form `/sys/class/net/<interface>/device` where *<interface>* is the interface name. This link points to a sysfs directory that represents the corresponding network device. You can read this symbolic link with **readlink** to confirm that an interface name corresponds to a particular network device.

"Device views in sysfs" on page 11 tells you where you can find the device directories with their attributes in sysfs.

Main steps for setting up a network interface

The main steps apply to all network devices drivers that are based on ccwgroup devices (for example, qeth and lcs devices). How to perform a particular step can be different for the different device drivers.

The steps that follow apply to Linux on z/VM and to Linux in LPAR mode. For Linux on KVM, these steps are performed for you on the KVM host. The steps can be different for the different device drivers.

1. Create a network device by combining suitable subchannels into a group device. The device driver then creates directories that represent the device in sysfs.
2. Configure the device through its attributes in sysfs. See "Device views in sysfs" on page 11. Some devices have attributes that can or must be set later when the device is online or when the connection is active.
3. Set the device online. This step associates the device with an interface name and thus makes the device known to the Linux network stack. For devices that are associated with a physical network adapter it also initializes the adapter for the network interface.
4. Configure and activate the interface. This step adds interface properties like IP addresses, netmasks, and MTU to the network interface and moves the network interface into state "up". The interface is then ready for user space (socket) programs to run connections and transfer data across it.

To configure a network device, use tools provided with Red Hat Enterprise Linux. See *Configuring and managing networking in Red Hat Enterprise Linux 9.1* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_and_managing_networking/index)

Chapter 2. Devices in sysfs

Most Linux device drivers create structures in sysfs. These structures hold information about individual devices and are also used to configure and control the devices.

Device categories

The `/sys/devices` directory includes several device categories that are specific to z/Architecture.

Figure 4 on page 7 illustrates a part of sysfs.

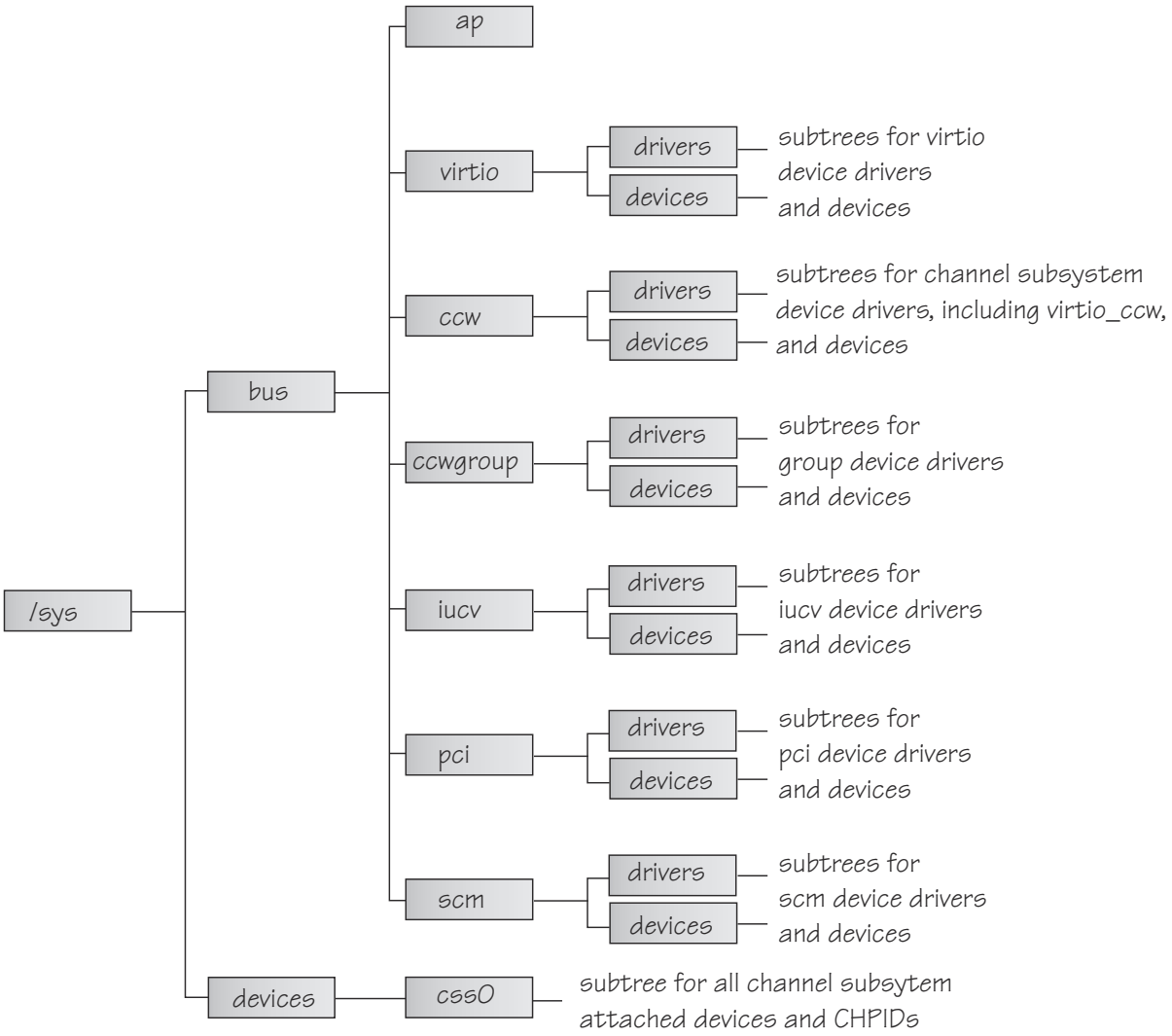


Figure 4. sysfs

`/sys/bus` and `/sys/devices` are common Linux directories. The directories following `/sys/bus` sort the device drivers according to the categories of devices they control. The sysfs branch for a particular category might be missing if there is no device for that category.

AP devices
are adjunct processors used for cryptographic operations.

virtio devices

are virtualized devices as used on KVM guests. This branch lists devices with names `virtio<n>` that represent the virtio aspects of virtio-ccw devices.

The CCW aspects of virtio-ccw devices are represented by corresponding devices in the `/sys/bus/ccw` branch, with device bus-IDs as device names. This publication uses the representation in the `/sys/bus/ccw` branch to work with virtio-ccw devices.

CCW devices
are devices that can be addressed with channel-command words (CCWs). These devices use a single subchannel on the mainframe's channel subsystem.

CCW group devices
are devices that use multiple subchannels on the mainframe's channel subsystem.

IUCV devices
are devices for virtual connections between z/VM guest virtual machines within an IBM mainframe. IUCV devices do not use the channel subsystem.

PCI devices
represent PCIe devices, for example, a 10GbE RoCE Express device. In sysfs, PCIe devices are listed in the `/pci` directory rather than the `/pcie` directory.

Table 1 on page 8 lists the z/Architecture specific device drivers that have representation in sysfs:

Table 1. Device drivers with representation in sysfs

Device driver	Category	sysfs directories
3215 console	CCW	`/sys/bus/ccw/drivers/3215`
3270 console	CCW	`/sys/bus/ccw/drivers/3270`
DASD	CCW	`/sys/bus/ccw/drivers/dasd-eckd` `/sys/bus/ccw/drivers/dasd-fba`
SCSI-over-Fibre Channel	CCW	`/sys/bus/ccw/drivers/zfcp`
Channel-attached tape	CCW	`/sys/bus/ccw/drivers/tape_34xx` `/sys/bus/ccw/drivers/tape_3590`
virtio CCW transport device driver	CCW	`/sys/bus/ccw/drivers/virtio_ccw`
Storage class memory supporting Flash Express	SCM	`/sys/bus/scm/drivers/scm_block`
Cryptographic	AP	`/sys/bus/ap/drivers/cex4card` `/sys/bus/ap/drivers/cex4queue`
DCSS	n/a	`/sys/devices/dcssblk`
z/VM recording	IUCV	`/sys/bus/iucv/drivers/vmlogrdr`
qeth (OSA-Express features and HiperSockets)	CCW group	`/sys/bus/ccwgroup/drivers/qeth`
LCS	CCW group	`/sys/bus/ccwgroup/drivers/lcs`
CTCM	CCW group	`/sys/bus/ccwgroup/drivers/ctcm`
10GbE RoCE Express devices (mlx4_en)	PCI	`sys/bus/pci/drivers/mlx4_core`
10 GbE RoCE Express2 devices (mlx5_core)	PCI	`sys/bus/pci/drivers/mlx5_core`
Internal Shared Memory	PCI	`/sys/bus/pci/drivers/ism`
NVMe	PCI	`/sys/bus/pci/drivers/nvme`

Some device drivers do not relate to physical devices that are connected through the channel subsystem. Their representation in sysfs differs from the CCW and CCW group devices, for example, the Cryptographic device drivers have their own category, AP.

The following sections provide more details about devices and their representation in sysfs.

Device directories

Each device that is known to Linux is represented by a directory in sysfs.

For CCW and CCW group devices the name of the directory is a *bus ID* that identifies the device within the scope of a Linux instance. For a CCW device, the bus ID is the device's device number with a leading "0.*<n>*.", where *<n>* is the subchannel set ID. For example, 0.1.0ab1.

CCW group devices are associated with multiple device numbers. For CCW group devices, the bus ID is the primary device number with a leading "0.*<n>*.", where *<n>* is the subchannel set ID.

"Device views in sysfs" on page 11 tells you where you can find the device directories with their attributes in sysfs. Red Hat Enterprise Linux 9.1 uses configuration files to control devices. For example, network devices have interface scripts called `/etc/sysconfig/network-scripts/ifcfg-<interface-name>`. See the *Configuring and managing networking in Red Hat Enterprise Linux 9.1* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_and_managing_networking/index) for details about configuration files.

Device attributes

The device directories contain attributes. You control a device by writing values to its attributes.

Some attributes are common to all devices in a device category, other attributes are specific to a particular device driver. The following attributes are common to all CCW devices:

online
You use this attribute to set the device online or offline. To set a device online, write the value 1 to its online attribute. To set a device offline, write the value 0 to its online attribute.

cutype
specifies the control unit type and model, if applicable. This attribute is read-only.

cmb_enable
enables I/O data collection for the device. See "Enabling, resetting, and switching off data collection" on page 540 for details.

devtype
specifies the device type and model, if applicable. This attribute is read-only.

availability
indicates whether the device can be used. The following values are possible:

good
This is the normal state. The device can be used.

boxed
DASD only: The device is locked by another operating system instance and cannot be used until the lock is surrendered or the DASD is accessed by force (see "Accessing DASD by force" on page 151).

no device
Applies to disconnected devices only. The device disappears after a machine check and the device driver requests to keep the device online anyway. Changes back to "good" when the device returns after another machine check and the device driver accepts the device back.

no path
Applies to disconnected devices only. After a machine check or a logical vary off, no path remains to the device. However, the device driver keeps the device online. Changes back to "good" when

the path returns after another machine check or logical vary on and the device driver accepts the device back.

modalias
contains the module alias for the device. It is of the format:

```
ccw:t<cu_type>m<cu_model>
```

or

```
ccw:t<cu_type>m<cu_model>dt<dev_type>dm<dev_model>
```

Setting attributes

Directly write to attributes or, for CCW devices, use the **chccwdev** command to set attribute values.

About this task

Because the KVM hypervisor hides many aspects of physical devices that back virtio devices, the scope for setting device attributes for these devices on KVM guests is limited.

Procedure

- You can set a writable attribute by writing the designated value to the corresponding attribute file.
- For CCW devices, you can also use the **chzdev** or the **chccwdev** command (see “chzdev - Configure IBM Z devices” on page 590 and “chccwdev - Set CCW device attributes” on page 575) to set attributes.

  With a single **chzdev** or **chccwdev** command you can:

  – Set an attribute for multiple devices
  – Set multiple attributes for a device, including setting the device online
  – Set multiple attributes for multiple devices

Working with newly available devices

Errors can occur if you try to work with a device before its sysfs representation is completely initialized.

About this task

When new devices become available to a running Linux instance, some time elapses until the corresponding device directories and their attributes are created in sysfs. Errors can occur if you attempt to work with a device for which the sysfs structures are not present or are not complete. These errors are most likely to occur and most difficult to handle when you are configuring devices with scripts.

Procedure

Use the following steps before you work with a newly available device to avoid such errors:

1. Attach the device, for example, with a z/VM CP ATTACH command or by dynamically attaching a device to a KVM virtual server.
2. Assure that the sysfs structures for the new device are complete:

   ```
   # echo 1 > /proc/cio_settle
   ```

   This command returns control after all pending updates to sysfs are complete.

   Tip: For CCW devices, you can omit this step if you then use **chccwdev** (see “chccwdev - Set CCW device attributes” on page 575) to work with the devices. **chccwdev** triggers cio_settle for you and waits for cio_settle to complete.

3. Assure that udev actions for the new device are complete.

```
# udevadm settle
```

The **settle** command returns control after all pending events are complete.

Results

You can now work with the new device. For example, you can set the device online or set attributes for the device.

Device views in sysfs

sysfs provides multiple views of device specific data.

The most important views are:

- “Device driver view” on page 11
- “Device category view” on page 12
- “Device view” on page 12
- “Channel subsystem view” on page 13

Many paths in sysfs contain device bus-IDs to identify devices. Device bus-IDs of subchannel-attached devices are of the form:

```
0.<n>.<devno>
```

where *<n>* is the subchannel set-ID and *<devno>* is the device number.

Device driver view

This view groups devices by the device drivers that control them.

The device driver view is of the form:

```
/sys/bus/<bus>/drivers/<driver>/<device_bus_id>
```

where:

<bus>
is the device category, for example, ccw or ccwgroup.

<driver>
is a name that specifies an individual device driver or the device driver component that controls the device (see Table 1 on page 8).

<device_bus_id>
identifies an individual device (see “Device directories” on page 9).

Note: DCSSs are not represented in this view.

Examples

- This example shows the path for an ECKD type DASD device:

```
/sys/bus/ccw/drivers/dasd-eckd/0.0.b100
```

- This example shows the path for a qeth device:

```
/sys/bus/ccwgroup/drivers/qeth/0.0.a100
```

- This example shows the path for a cryptographic device (a CEX7A card):

```
/sys/bus/ap/drivers/cex7a/card3b
```

Device category view

This view groups devices by major categories that can span multiple device drivers.

The device category view does not sort the devices according to their device drivers. All devices of the same category are contained in a single directory. The device category view is of the form:

```
/sys/bus/<bus>/devices/<device_bus_id>
```

where:

<bus>
is the device category, for example, ccw or ccwgroup.

<device_bus_id>
identifies an individual device (see “Device directories” on page 9).

Note:

- DCSSs are not represented in this view.
- `/sys/bus/ccw/devices` includes virtio CCW devices.

Examples

- This example shows the path for a CCW device.

```
/sys/bus/ccw/devices/0.0.b100
```

- This example shows the path for a CCW group device.

```
/sys/bus/ccwgroup/devices/0.0.a100
```

- This example shows the path for a cryptographic device:

```
/sys/bus/ap/devices/card3b
```

Device view

This view sorts devices according to their device drivers, but independent from the device category. It also includes logical devices that are not categorized.

The device view is of the form:

```
/sys/devices/<driver>/<device>
```

where:

<driver>
is a name that specifies an individual device driver or the device driver component that controls the device.

<device>
identifies an individual device. The name of this directory can be a device bus-ID or the name of a DCSS or IUCV device.

Examples

- This example shows the path for a qeth device.

```
/sys/devices/qeth/0.0.a100
```

- This example shows the path for a DCSS block device.

```
/sys/devices/dcssblk/mydcss
```

Channel subsystem view

The channel subsystem view shows the relationship between subchannels and devices.

The channel subsystem view is of the form:

```
/sys/devices/css0/<subchannel>
```

where:

<subchannel>
is a subchannel number with a leading "0.*<n>*.", where *<n>* is the subchannel set ID.

I/O subchannels show the devices in relation to their respective subchannel sets and subchannels. An I/O subchannel is of the form:

```
/sys/devices/css0/<subchannel>/<device_bus_id>
```

where:

<subchannel>
is a subchannel number with a leading "0.*<n>*.", where *<n>* is the subchannel set ID.

<device_bus_id>
is a device number with a leading "0.*<n>*.", where *<n>* is the subchannel set ID (see “Device directories” on page 9).

Examples

- This example shows a CCW device with device number 0xb100 that is associated with a subchannel 0x0001.

```
/sys/devices/css0/0.0.0001/0.0.b100
```

- This example shows a CCW device with device number 0xb200 that is associated with a subchannel 0x0001 in subchannel set 1.

```
/sys/devices/css0/0.1.0001/0.1.b200
```

- The entries for a group device show as separate subchannels. If a CCW group device uses three subchannels 0x0002, 0x0003, and 0x0004 the subchannel information could be:

```
/sys/devices/css0/0.0.0002/0.0.a100
/sys/devices/css0/0.0.0003/0.0.a101
/sys/devices/css0/0.0.0004/0.0.a102
```

  Each subchannel is associated with a device number. Only the primary device number is used for the bus ID of the device in the device driver view and the device view.

- This example lists the information available for a non-I/O subchannel with which no device is associated:

```
ls /sys/devices/css0/0.0.ff00/
bus  driver  modalias  subsystem  type  uevent
```

Subchannel attributes

There are sysfs attributes that represent subchannel properties, including common attributes and information specific to the subchannel type.

Subchannels have two common attributes:

type
The subchannel type, which is a numerical value, for example:

- 0 for an I/O subchannel
- 1 for a CHSC subchannel
- 3 for an EADM subchannel

modalias
The module alias for the device of the form css:t<*n*>, where <*n*> is the subchannel type (for example, 0 or 1).

These two attributes are the only ones that are always present. Some subchannels, like I/O subchannels, might contain devices and further attributes.

Apart from the bus ID of the attached device, I/O subchannel directories typically contain these attributes:

chpids
is a list of the channel-path identifiers (CHPIDs) through with the device is connected. See also "Channel path ID information" on page 15.

pimpampom
provides the path installed, path available, and path operational masks. See *z/Architecture Principles of Operation*, SA22-7832 for details about the masks.

Channel path measurement

For Linux in LPAR mode and Linux on z/VM, a sysfs attribute controls the channel path measurement facility of the channel subsystem.

```
/sys/devices/css0/cm_enable
```

With the `cm_enable` attribute you can enable and disable the extended channel-path measurement facility. It can take the following values:

0
Deactivates the measurement facility and remove the measurement-related attributes for the channel paths. No action if measurements are not active.

1
Attempts to activate the measurement facility and create the measurement-related attributes for the channel paths. No action if measurements are already active.

If a machine does not support extended channel-path measurements the `cm_enable` attribute is not created.

Two sysfs attributes are added for each channel path object:

cmg
Specifies the channel measurement group or unknown if no characteristics are available.

shared
Specifies whether the channel path is shared between LPARs or unknown if no characteristics are available.

If measurements are active, two more sysfs attributes are created for each channel path object:

measurement
A binary sysfs attribute that contains the extended channel-path measurement data for the channel path. It consists of eight 32-bit values and must always be read in its entirety, or 0 will be returned.

measurement_chars
A binary sysfs attribute that is either empty, or contains the channel measurement group dependent characteristics for the channel path, if the channel measurement group is 2 or 3. If not empty, it consists of five 32-bit values.

Examples

- To turn measurements on issue:

```
# echo 1 > /sys/devices/css0/cm_enable
```

- To turn measurements off issue:

```
# echo 0 > /sys/devices/css0/cm_enable
```

Channel path ID information

All CHPIDs that are known to Linux are shown alongside the subchannels in the `/sys/devices/css0` directory.

The directories that represent the CHPIDs have the form:

```
/sys/devices/css0/chp0.<chpid>
```

where *<chpid>* is a two digit hexadecimal CHPID.

Example: `/sys/devices/css0/chp0.4a`

Setting a CHPID logically online or offline

Directories that represent CHPIDs contain a `status` attribute that you can use to set the CHPID logically online or offline.

Before you begin

Do not set all CHPIDs that connect a vital device offline. For example, Linux will crash if you set all CHPIDs for the root device offline.

About this task

When a CHPID has been set logically offline from a particular Linux instance, the CHPID is, in effect, offline for this Linux instance. A CHPID that is shared by multiple operating system instances can be logically online to some instances and offline to others. A CHPID can also be logically online to Linux while it has been varied off at the SE.

Procedure

Issue a command of this form:

```
# echo <value> > /sys/devices/css0/chp0.<CHPID>/status
```

where:

<CHPID>
is a two digit hexadecimal CHPID.

<value>
is either on or `off`.

Examples

- To set a CHPID 0x4a logically offline issue:

```
# echo off > /sys/devices/css0/chp0.4a/status
```

- To read the `status` attribute to confirm that the CHPID is logically offline issue:

```
# cat /sys/devices/css0/chp0.4a/status
offline
```

- To set the same CHPID logically online issue:

```
# echo on > /sys/devices/css0/chp0.4a/status
```

- To read the `status` attribute to confirm that the CHPID is logically online issue:

```
# cat /sys/devices/css0/chp0.4a/status
online
```

Configuring a CHPID on LPAR

For Linux in LPAR mode, directories that represent CHPIDs contain a `configure` attribute that you can use to query and change the configuration state of I/O channel-paths.

About this task

The following configuration changes are supported:

- From standby to configured ("configure")
- From configured to standby ("deconfigure")

Procedure

Issue a command of this form:

```
# echo <value> > /sys/devices/css0/chp0.<CHPID>/configure
```

where:

<CHPID>
is a two digit hexadecimal CHPID.

<value>
is either 1 or 0.

To query and set the configure value using commands, see “chchp - Change channel path status” on page 577 and “lschp - List channel paths” on page 653.

Examples

- To set a channel path with the ID 0x40 to standby issue:

```
# echo 0 > /sys/devices/css0/chp0.40/configure
```

This operation is equivalent to performing a Configure Channel Path Off operation on the hardware management console.

- To read the configure attribute to confirm that the channel path has been set to standby issue:

```
# cat /sys/devices/css0/chp0.40/configure
0
```

- To set the same CHPID to configured issue:

```
# echo 1 > /sys/devices/css0/chp0.40/configure
```

This operation is equivalent to performing a Configure Channel Path On operation on the hardware management console.

- To read the status attribute to confirm that the CHPID has been set to configured issue:

```
# cat /sys/devices/css0/chp0.40/configure
1
```

Finding the physical channel associated with a CHPID

Use the mapping of physical channel IDs (PCHID) to CHPIDs to find the hardware from the CHPID number or the CHPID numbers from the PCHID.

About this task

A CHPID is associated with either a physical port or with an internal connection defined inside the mainframe, such as HiperSockets. See Figure 5 on page 17. You can determine the PCHID or internal channel ID number associated with a CHPID number.

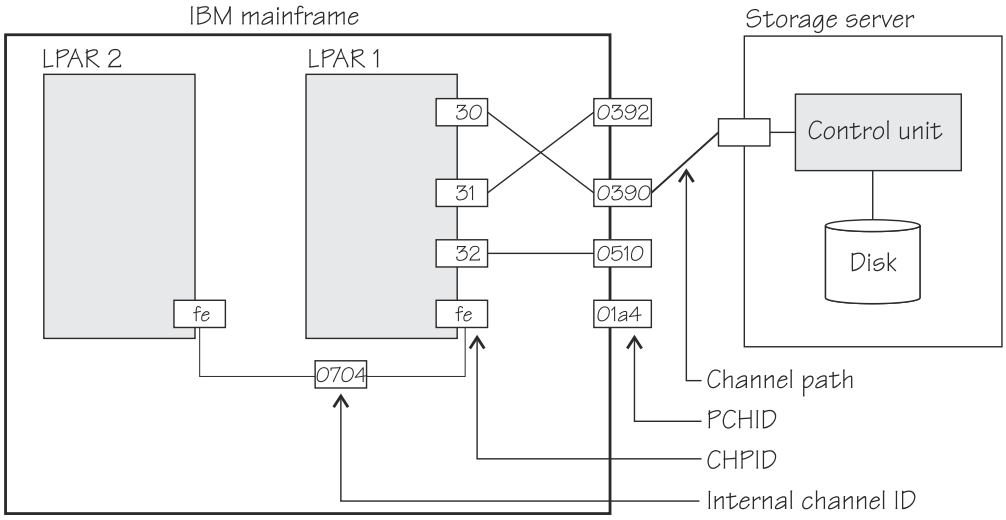


Figure 5. Relationships between CHPIDs, PCHIDs, and internal channel ID numbers.

Knowing the PCHID number can be useful in the following situations:

- When Linux indicates that a CHPID is in an error state, you can use the PCHID number to identify the associated hardware.
- When a hardware interface requires service action, the PCHID mapping can be used to determine which CHPIDs and I/O devices will be affected.

The internal channel ID number can be useful to determine which CHPIDs are connected to the same communication path, such as a HiperSockets link.

Procedure

To find the physical channel ID corresponding to a CHPID, either:

- Display the mapping of all CHPIDs to PCHIDs. Issue the **lschp** command:

```
# lschp
```

- Find the channel-ID related files for the CHPID.

  These sysfs files are located under `/sys/devices/css0/chp0.<num>`, where *<num>* is the two-digit, lower-case, hexadecimal CHPID number. There are two attribute files:

  chid
  The channel ID number.

chid_external
A flag indicating whether this CHPID is associated with an internal channel ID (value 0) or a physical channel ID (value 1).

The sysfs attribute files are not created when no channel ID information is available to Linux. For Linux in LPAR mode, this information is always available. For Linux on z/VM and Linux on KVM, the availability depends on the configuration and on the hypervisor version.

Example

The **lschp** command shows channel ID information in a column labeled PCHID. Internal channel IDs are enclosed in brackets. If no channel ID information is available, the column will show "-".

```
# lschp
CHPID  Vary  Cfg.  Type  Cmg  Shared  PCHID
=========================================
0.30   1     1     1b    2    1       0390
0.31   1     1     1b    2    1       0392
0.32   1     1     1b    2    1       0510
0.33   1     1     1b    2    1       0512
0.34   1     0     1b    -    -       0580
0.fc   1     1     24    3    1      (0702)
0.fd   1     1     24    3    1      (0703)
0.fe   1     1     24    3    1      (0704)
```

This example shows that CHPID 30 is associated with PCHID 0390, while CHPID fe is associated with internal channel ID 0704.

Alternatively, check the channel ID sysfs files, for example for CHPID 30:

```
# cat /sys/devices/css0/chp0.30/chid
0390
# cat /sys/devices/css0/chp0.30/chid_external
1
```

Checking the FCES status of a CHPID

For Linux on IBM Z, directories that represent CHPIDs contain a read-only attribute, `esc`, that you can use to query the Fibre Channel Endpoint Security capability of I/O channel-paths.

About this task

The `esc` sysfs attribute can have the following values:

0
FCES is not supported.

1
The channel path supports authentication.

2 or 3
The channel path supports authentication and encryption.

Procedure

- To read the FCES status of a CHPID, issue:

```
# cat /sys/devices/css0/chp0.<CHPID>/esc
```

For example:

```
# cat /sys/devices/css0/chp0.34/esc
2
```

CCW hotplug events

A hotplug event is generated when a CCW device appears or disappears with a machine check.

The hotplug events provide the following variables:

CU_TYPE
for the control unit type of the device that appeared or disappeared.

CU_MODEL
for the control unit model of the device that appeared or disappeared.

DEV_TYPE
for the type of the device that appeared or disappeared.

DEV_MODEL
for the model of the device that appeared or disappeared.

MODALIAS
for the module alias of the device that appeared or disappeared. The module alias is the same value that is contained in `/sys/devices/css0/<subchannel_id>/<device_bus_id>/modalias` and is of the format`ccw:t<cu_type>m<cu_model>` or `ccw:t<cu_type>m<cu_model>dt<dev_type>dm<dev_model>`

Hotplug events can be used, for example, for:

- Automatically setting devices online as they appear
- Automatically loading driver modules for which devices have appeared

Chapter 3. Device auto-configuration for Linux in LPAR mode

As of z14 and LinuxONE II, you can store device configuration data for Linux in LPAR mode on the Support Element (SE).

You provide this configuration data through a hardware management console (HMC) interface in Dynamic Partition Manager (DPM) mode. This data can then be processed automatically by Linux during the boot process.

Making devices available to Linux

Devices must be configured on the hardware and in Linux before they can be used.

Defining devices to an LPAR

Typical IBM Z and LinuxONE systems run numerous operating system instances in parallel and connect to a considerable number of storage, network, and other peripheral devices. In this environment, device access must be controlled.

- Workload isolation demands selective and controlled device access.
- Operating systems expend cycles, time, and memory to manage each device. For example, on Linux, udev creates structures for each registered device.

Data centers with discrete host systems can use physical cabling between hosts and peripheral devices to manage device access. On IBM Z and LinuxONE systems with their logical partitions (LPARs), much of this cabling would need to be within the hardware system itself.

Instead of cables,a *hardware configuration* controls which LPAR has access to which I/O device. The hardware configuration is specified in an input/output configuration data set (IOCDS). Traditionally, IOCDSs are created with the hardware configuration definition (HCD) program.

DPM: The IBM Z or LinuxONE firmware automatically processes the device-configuration data you provide on the HMC interface, and creates and activates a corresponding IOCDS for you.

Controlling device availability on Linux

The hardware configuration already limits the I/O devices that are available to a Linux instance. The `cio_ignore` feature provides another control point on Linux. With `cio_ignore`, you can create and maintain a list of devices to be ignored by Linux.

DPM: If `cio_ignore` is active, the list of devices to be ignored by Linux is automatically adjusted, at boot-time, to accommodate all devices that are configured on the HMC interface. If available to the hardware, these devices become available to Linux and are set online.

To enforce the current `cio_ignore` list, you can use the `rd.zdev=no-auto` kernel parameter to disregard auto-configuration for devices on Linux.This parameter also affects the initial online state of PCIe devices and cryptographic devices, see “rd.zdev=no-auto - Override initial device availability for DPM mode” on page 792.

Configuring devices on Linux

On a running Linux instance, you can use the **chzdev** command to configure individual devices. With the **lszdev** command you can display the device settings. These tools distinguish different types of configurations.

Active configuration
The current configuration, which might include settings that do not persist across reboots.

Persistent configuration
The configuration to be applied when the Linux instance is booted.

DPM only: Auto-configuration
The configuration as specified on the HMC interface.

chzdev provides a richer set of configuration options than the HMC interface. The active and persistent settings are often a fine-tuned version of the auto-configuration.

Overriding the auto-configuration

You can override the auto-configuration for a device with a persistent configuration.

For devices that come online early in the boot process, use the `zdev:early` device attribute to ensure that this persistent configuration is available at this early stage (see "chzdev - Configure IBM Z devices" on page 590).

Managing auto-configuration data

Use the **lszdev** and **chzdev** commands version 2.5 or later to manage auto-configuration data.

Displaying auto-configuration data

The **lszdev** command can display auto-configuration data.

Use the **lszdev** command with the `--auto-conf` option to display a list of devices for which auto-configuration data is available.

Example:

```
# lszdev --auto-conf
TYPE       ID        AUTO
dasd-eckd  0.0.ec30  yes
dasd-eckd  0.0.ec31  yes
```

Auto-configuration settings can be overridden with settings in the persistent configuration. Omit the `--auto-conf` option to find out for which devices auto-configuration is effective:

Example:

```
# lszdev
TYPE         ID                          ON   PERS   NAMES
dasd-eckd    0.0.ec30                    yes  yes    dasda
dasd-eckd    0.0.ec31                    yes  auto   dasdb
dasd-eckd    0.0.ec32                    yes  no     dasdc
qeth         0.0.f5f0:0.0.f5f1:0.0.f5f2  yes  no     encf5f0
generic-ccw  0.0.0009                    yes  no
```

In the example, auto-configuration data is effective for only one device, `0.0.ec31`. Effective auto-configuration data is indicated through the value `auto` in the PERS column of the command output.

The **lszdev** output for detailed information about a device includes a separate column, AUTOCONF, for auto-configuration data, if available.

Example:

```
# lszdev -i 0.0.ec31
DEVICE dasd-eckd 0.0.ec31
Names                : -
Modules              : dasd_eckd_mod dasd_mod
Online               : no
Exists               : yes
Persistent           : no
Auto-configured      : yes

ATTRIBUTE                    ACTIVE   PERSISTENT  AUTOCONF
cmb_enable                   "0"      -           -
eer_enabled                  "0"      -           -
erplog                       "0"      -           -
failfast                     "0"      -           -
last_known_reservation_state "none"   -           -
online                       "1"      -           "1"
raw_track_access             "0"      -           -
readonly                     "0"      -           -
reservation_policy           "ignore" -           -
use_diag                     "0"      -           -
```

If the AUTOCONF column is omitted, no auto-configuration data is available for this device. You can force the column with the `--auto-conf` option.

You can access the raw auto-configuration data through sysfs at `/sys/firmware/sclp_sd/config/data`. For example, you can use this sysfs attribute as a source for importing auto-configuration data with the **chzdev** command:

```
# chzdev --import /sys/firmware/sclp_sd/config/data --auto-conf
```

Modifying the auto-configuration

Persistent changes to the auto-configuration can be made only through the hardware interface through which the original device configuration is specified. Such changes are applied with the next reboot.

To refresh the raw auto-configuration in sysfs at `/sys/firmware/sclp_sd/config/data`, target an echo command at `/sys/firmware/sclp_sd/config/reload`.

```
# echo > /sys/firmware/sclp_sd/config/reload
```

You can use **chzdev** with the `-d` and `--auto-conf` options to temporarily remove the auto-configuration for a device.

Example:

```
# chzdev -d --auto-conf 0.0.ec31
Deconfiguring devices in the auto-configuration only
ECKD DASD 0.0.ec31 deconfigured
```

Auto-configuration settings are then not applied when the device appears. These configuration changes do not remove the corresponding configuration data on the SE. The auto-configuration data for the device is restored with the next reboot.

Overriding the auto-configuration for devices that are used early in the boot process

With **chzdev**, you can override settings from the auto-configuration in the active configuration or persistently. Some persistent settings for devices that are set online early in the boot process must be included in the initial RAM disk.

Use the **chzdev** command to set the `zdev:early` device attribute for such devices.

Example:

```
# chzdev -e dasd-fba e030 zdev:early=1
FBA DASD 0.0.e030 configured
Note: The initial RAM-disk must be updated for these changes to take effect:
- FBA DASD 0.0.e030
Update initial RAM-disk now? (yes/no) yes
```

Do not indiscriminately include configuration settings in the initial RAM disk. To remove settings for a device, remove the `zdev:early` attribute from the device settings.

Example:

```
# chzdev -e dasd-fba e030 --remove-attribute zdev:early
FBA DASD 0.0.e030 configured
Note: The initial RAM-disk must be updated for these changes to take effect:
- FBA DASD 0.0.e030
Update initial RAM-disk now? (yes/no) yes
```

Use the **`lszdev`** command to list all devices that are configured with the `zdev:early` attribute.

Example:

```
# lszdev --by-attr zdev:early=1
TYPE     ID                                                ON  PERS NAMES
dasd-fba 0.0.e030                                          yes yes  dasda
zfcp-lun 0.0.1911:0x50050763070845e3:0x4082409f00000000 no  yes
```

Chapter 4. Kernel and module parameters

Kernel and module parameters are used to configure the kernel and kernel modules.

Individual kernel parameters or module parameters are single keywords, or keyword-value pairs of the form keyword=*<value>* with no blank. Blanks separate consecutive parameters.

Kernel parameters and module parameters are encoded as strings of ASCII characters. For tape or the z/VM reader as a boot device, the parameters can also be encoded in EBCDIC.

Use *kernel parameters* to configure the base kernel and any optional kernel parts that have been compiled into the kernel image. Use *module parameters* to configure separate kernel modules. Do not confuse kernel and module parameters. Although a module parameter can have the same syntax as a related kernel parameter, kernel and module parameters are specified and processed differently.

Where possible, this document describes kernel parameters with the device driver or feature to which they apply. Kernel parameters that apply to the base kernel or cannot be attributed to a particular device driver or feature are described in Chapter 63, "Selected kernel parameters," on page 779. You can also find descriptions for most of the kernel parameters in `Documentation/admin-guide/kernel-parameters.txt` in the Linux source tree.

Separate kernel modules must be loaded before they can be used. Many modules are loaded automatically by Red Hat Enterprise Linux 9.1 when they are needed. To keep the module parameters in the context of the device driver or feature module to which they apply, this document describes module parameters as part of the syntax you would use to load the module with modprobe.

To find the separate kernel modules for Red Hat Enterprise Linux 9.1, list the contents of the subdirectories of `/lib/modules/<kernel-release>` in the Linux file system. In the path, *<kernel-release>* denotes the kernel level. You can query the value for *<kernel-release>* with **uname -r**.

Specifying kernel parameters

Different methods are available for passing kernel parameters to Linux.

- Including kernel parameters in a boot configuration
- Using a kernel parameter file
- Specifying kernel parameters when booting Linux

Kernel parameters that you specify when booting Linux are not persistent. To define a permanent set of kernel parameters for a Linux instance, include these parameters in the boot configuration.

Note: Parameters that you specify on the kernel parameter line might interfere with parameters that Red Hat Enterprise Linux 9.1 sets for you. Read `/proc/cmdline` to find out which parameters were used to start a running Linux instance.

Including kernel parameters in a boot configuration

Use the zipl tool to create Linux boot configurations for IBM mainframe systems.

Which sources of kernel parameters you can use depends on the mode in which you run **zipl**. See "zipl modes and syntax overview" on page 58 for details.

A boot configuration can include up to 895 characters of kernel parameters. See also "How kernel parameters from different sources are combined" on page 28.

Running zipl in configuration-file mode

In configuration-file mode, you issue the **zipl** command with command arguments that identify a section in a **zipl** configuration file or a Boot Loader Specification (BLS) snippet..

The possible sources of kernel parameters depend on where the details of the boot configuration are specified, in a zipl configuration-file section or in a BLS snippet.

zipl configuration-file section

Boot configurations in a zipl configuration-file section have three potential sources of kernel parameters, as illustrated in Figure 6 on page 26.

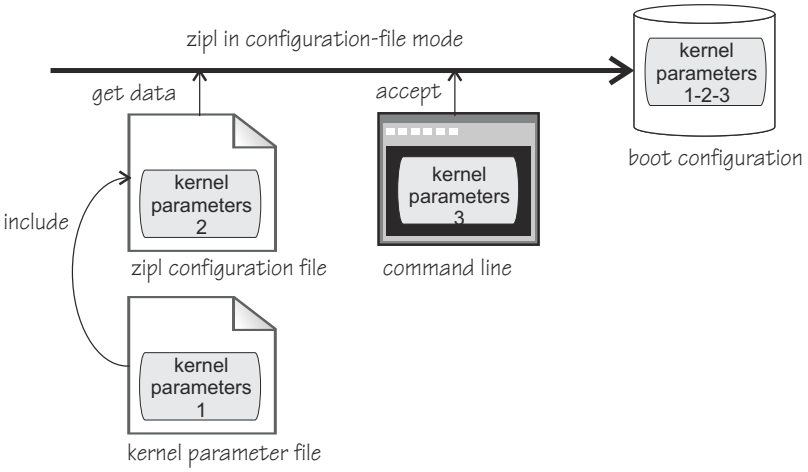


Figure 6. Sources of kernel parameters: zipl configuration-file section

zipl concatenates the kernel parameters from these sources in the following order:

1. Parameters that are specified in the kernel parameter file
2. Parameters that are specified in the **zipl** configuration-file
3. Parameters that are specified on the command line

BLS snippet

Boot configurations in a BLS snippet have two potential sources of kernel parameters, as illustrated in Figure 7 on page 26.

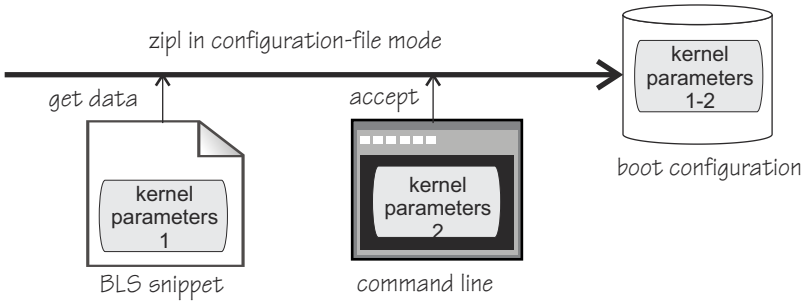


Figure 7. Sources of kernel parameters: BLS snippet

zipl concatenates the kernel parameters from these sources in the following order:

1. Parameters that are specified in the BLS snippet
2. Parameters that are specified on the command line

See "zipl modes and syntax overview" on page 58 for details about the **zipl** command modes.

Running zipl in command-line mode

In command-line mode, you specify the details about the boot configuration to be created as arguments for the **zipl** command.

As shown in Figure 8 on page 27, there are two sources of kernel parameters for **zipl** in command-line mode.

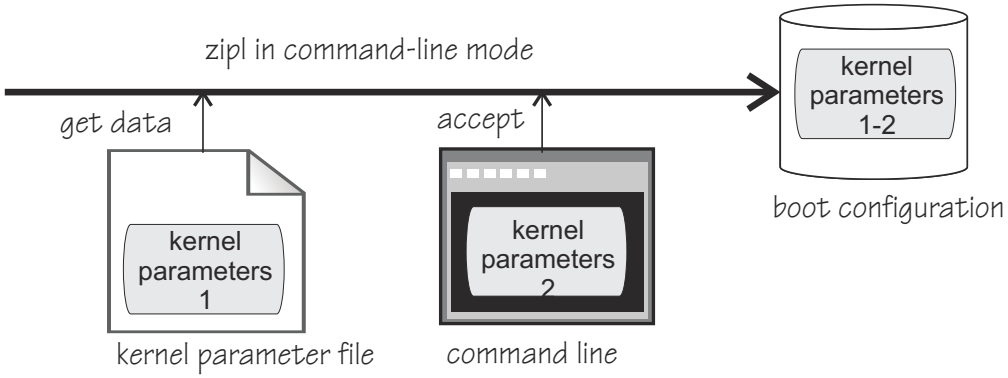


Figure 8. Sources of kernel parameters for zipl in command-line mode

In command-line mode, **zipl** concatenates the kernel parameters in the order:

1. Parameters that are specified in the kernel parameter file
2. Parameters that are specified on the command line

See "zipl modes and syntax overview" on page 58 for details about the **zipl** command modes.

Using a kernel parameter file in the z/VM reader

For booting Linux from the z/VM reader, you can use a kernel parameter file in the reader.

See "Booting from the z/VM reader" on page 114 for more details.

Specifying kernel parameters when booting Linux

Depending on the boot device and whether you boot Linux in a z/VM guest virtual machine or in LPAR mode, you can provide kernel parameters when you start the boot process.

zipl interactive boot menu on DASD
When booting Linux on z/VM or in LPAR mode with a zipl interactive boot menu on a DASD boot device, you can display the menu and specify kernel parameters as you select a boot configuration. See "Example for a DASD menu configuration on z/VM" on page 111 and "Example for a DASD menu configuration (LPAR)" on page 98 for details.

z/VM guest virtual machine with a CCW boot device
When booting Linux in a z/VM guest virtual machine from a CCW boot device, you can use the PARM parameter of the IPL command to specify kernel parameters. CCW boot devices include DASD, tape, the z/VM reader, and NSS.

For details, see the subsection of "Booting Linux in a z/VM guest virtual machine" on page 109 that applies to your boot device.

z/VM guest virtual machine with an FC-attached SCSI disk boot device
When booting Linux in a z/VM guest virtual machine from an FC-attached SCSI disk boot device, you can use the SET LOADDEV command with the SCPDATA option to specify kernel parameters. See "Booting as a z/VM guest from a SCSI device" on page 112 for details.

LPAR mode with a FC-attached SCSI disk boot device
When booting Linux in LPAR mode from an FC-attached SCSI disk boot device, you can specify kernel parameters in the **Operating system specific load parameters** field on the HMC Load panel. See Figure 33 on page 99.

Kernel parameters as entered from a CMS or CP session are interpreted as lowercase on Linux.

Adding kernel parameters to a boot configuration

When booting a Linux instance, you can specify kernel parameters that are used in addition to parameters in the boot configuration.

By default, the kernel parameters you specify when booting are concatenated to the end of the kernel parameters in your boot configuration. In total, the combined kernel parameter string that is used for booting can be up to 4096 characters.

If kernel parameters are specified in a combination of methods, they are concatenated in the following order:

1. Kernel parameters that have been included in the boot configuration with zipl
2. DASD only: zipl kernel parameters that are specified with the interactive boot menu
3. Depending on where you are booting Linux:
   - z/VM: kernel parameters that are specified with the PARM parameter for CCW boot devices; kernel parameters specified as SCPDATA for SCSI boot devices
   - LPAR: kernel parameters that are specified on the HMC Load panel for FCP-attached SCSI disk boot devices

If the combined kernel parameter string contains conflicting settings, the last specification in the string overrides preceding ones. Thus, you can specify a kernel parameter when booting to override an unwanted setting in the boot configuration.

Example

If the kernel parameters in your boot configuration include `possible_cpus=8` but you specify `possible_cpus=2` when booting, Linux uses `possible_cpus=2`.

Replacing all kernel parameters in a boot configuration

Kernel parameters that you specify when booting can completely replace the kernel parameters in your boot configuration.

To replace all kernel parameters in your boot configuration, specify the new parameter string with a leading equal sign (=).

Note: This feature is intended for expert users who want to test a set of parameters. By replacing all parameters, you might inadvertently omit parameters that the boot configuration requires. Furthermore, you might omit parameters other than kernel parameters that Red Hat Enterprise Linux 9.1 includes in the parameter string for use by the init process.

Read `/proc/cmdline` to find out with which parameters a running Linux instance was started (see also “Displaying the current kernel parameter line” on page 29).

How kernel parameters from different sources are combined

If kernel parameters are specified in a combination of methods, they are concatenated in a specific order.

1. Kernel parameters that have been included in the boot configuration (see “Including kernel parameters in a boot configuration” on page 25).

   The kernel parameters in the boot configuration cannot exceed 895 characters. If more then 895 characters are specified, the excessive characters are truncated.
2. LPAR or z/VM: Kernel parameters that you specify through the HMC or through z/VM interfaces (see “Specifying kernel parameters when booting Linux” on page 27).

   For DASD boot devices you can specify up to 64 characters (z/VM only); for FC-attached SCSI disk boot devices you can specify up to 3452 characters.

In total, the combined kernel parameter string that is passed to the Linux kernel for booting can be up to 4096 characters.

Multiple specifications for the same parameter

For some kernel parameters, multiple instances in the kernel parameter string are treated cumulatively. For example, multiple specifications for `cio_ignore=` are all processed and combined.

Conflicting kernel parameters

If the kernel parameter string contains kernel parameters with mutually exclusive settings, the last specification in the string overrides preceding ones. Thus, you can specify a kernel parameter when booting to override an unwanted setting in the boot configuration.

Example: If the kernel parameters in your boot configuration include `possible_cpus=8` but you specify `possible_cpus=2` when booting, Linux uses `possible_cpus=2`.

Parameters other than kernel parameters

Parameters on the kernel parameter string that the kernel does not recognize as kernel parameters are ignored by the kernel and made available to user space programs. How multiple specifications and conflicts are resolved for such parameters depends on the program that evaluates them.

Examples for kernel parameters

Typical parameters that are used for booting Red Hat Enterprise Linux 9.1 configure the console.

conmode=*<mode>*, condev=*<cuu>*, console=*<name>*
to set up the Linux console. See "Console kernel parameter syntax" on page 42 for details.

See Chapter 63, "Selected kernel parameters," on page 779 for more examples of kernel parameters.

Displaying the current kernel parameter line

Read `/proc/cmdline` to find out with which kernel parameters a running Linux instance was booted.

About this task

Apart from kernel parameters, which are evaluated by the Linux kernel, the kernel parameter line can contain parameters that are evaluated by user space programs, for example, modprobe.

See also "Displaying current IPL parameters" on page 115 about displaying the parameters that were used to IPL and boot the running Linux instance.

Procedure

- Read `/proc/cmdline`.
  For example:

```
# cat /proc/cmdline
     vconsole.keymap=us
     cio_ignore=all,!condev
     crashkernel=auto
     rd.zfcp=0.0.1707,0x500507630513c1ae,0x402140b600000000
     rd.zfcp=0.0.1807,0x500507630508c1ae,0x402140b600000000
     vconsole.font=latarcyrheb-sun16
     LANG=en_US.UTF-8
     BOOT_IMAGE=0
```

Kernel parameters for rebooting

When rebooting, you can use the current kernel parameters or an alternative set of kernel parameters. By default, Linux uses the current kernel parameters for rebooting. See "Rebooting from an alternative

source" on page 117 about how to set up Linux to use different kernel parameters for re-IPL and the associated reboot.

Specifying parameters for modules

How to specify parameters for modules depends on how the module is loaded, for example, automatically, through a tool, or from the command line.

You can specify parameters for modules with the **modprobe** command or on the kernel parameter line. You can specify certain parameters for modules in a boot configuration. Avoid specifying the same parameter through multiple means.

Specifying module parameters with modprobe

If you load a module explicitly with a modprobe command, you can specify the module parameters as command arguments.

Module parameters that are specified as arguments to modprobe are effective until the module is unloaded only.

Note: Parameters that you specify as command arguments might interfere with parameters that Red Hat Enterprise Linux 9.1 sets for you.

Specifying parameters on the kernel parameter line

Parameters that the kernel does not recognize as kernel parameters are ignored by the kernel and made available to user space programs.

One of these programs is modprobe, which Red Hat Enterprise Linux 9.1 uses to load modules for you. modprobe interprets module parameters that are specified on the kernel parameter line if they are qualified with a leading module prefix and a dot.

For example, you can include a specification with `dasd_mod.dasd=` on the kernel parameter line. modprobe evaluates this specification as the `dasd=` module parameter when it loads the dasd_mod module.

Including parameters for modules in a boot configuration

Parameters for modules that are required early during the boot process must be included in the boot configuration.

About this task

Red Hat Enterprise Linux 9.1 uses an initial file system (initramfs) when booting. The initramfs does not contain device specifications. Instead, it takes parameters from **dracut** during the boot process. **dracut** obtains the parameters by parsing the kernel parameter line for parameters with an "rd." prefix.

Anaconda writes information about devices that must be accessible during the boot process to `zipl.conf` for you. An example is the device with the root file system.

Procedure

Follow these steps to provide parameters on a kernel command line to be evaluated, for example, by **dracut**:

1. With an "rd." prefix, specify the parameters in `zipl.conf`. For example, to specify a DASD, use `rd.dasd=`. The parameters are lower case and are case-sensitive. See the **dracut** man page, dracut.cmdline(7), for more details about parameters with an "rd." prefix.
2. Run **zipl** to include the new parameter line in your boot configuration.

Displaying information about module parameters

Loaded modules can export module parameter settings to sysfs.

The parameters for modules are available as sysfs attributes of the form:

```
/sys/module/<module_name>/parameters/<parameter_name>
```

Before you begin

You can display information about modules that fulfill these prerequisites:

- The module must be loaded.
- The module must export the parameters to sysfs.

Procedure

To find and display the parameters for a module, follow these steps:

1. Optional: Confirm that the module of interest is loaded by issuing a command of this form:

   ```
   # lsmod | grep <module_name>
   ```

   where *<module_name>* is the name of the module.
2. Optional: Get an overview of the parameters for the module by issuing a command of this form:

   ```
   # modinfo <module_name>
   ```
3. Check if the module of interest exports parameters to sysfs. Issue a command of the form:

   ```
   # ls /sys/module/<module_name>/parameters
   ```
4. If the previous command listed parameters, you can display the value for the parameter you are interested in. Issue a command of the form:

   ```
   # cat /sys/module/<module_name>/parameters/<parameter_name>
   ```

Example

- To list the module parameters for the ap module, issue:

```
# ls /sys/module/ap/parameters
 domain
 ...
```

- To display the value of the domain parameter, issue:

```
# cat /sys/module/ap/parameters/domain
1
```

Part 2. Booting and shutdown

These device drivers and features are useful for booting and shutting down instances of Red Hat Enterprise Linux 9.1.

Newest version

You can find the newest version of this book at
ibm.com/docs/en/linux-on-systems?topic=linuxone-distributions

Restrictions

For prerequisites and restrictions see the IBM Z architecture specific information in the Red Hat Enterprise Linux 9.1 release notes at
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

Chapter 5. Console device drivers

The console device drivers support terminal devices for basic Linux control, for example, for booting Linux, for troubleshooting, and for displaying Linux kernel messages.

Linux in LPAR mode

The only interface to a Linux instance in an LPAR before the boot process is completed is the Hardware Management Console (HMC), see Figure 9 on page 35. After the boot process has completed, you typically use a network connection to access Linux through a user login, for example, in an SSH session. The possible connections depend on the configuration of your particular Linux instance.

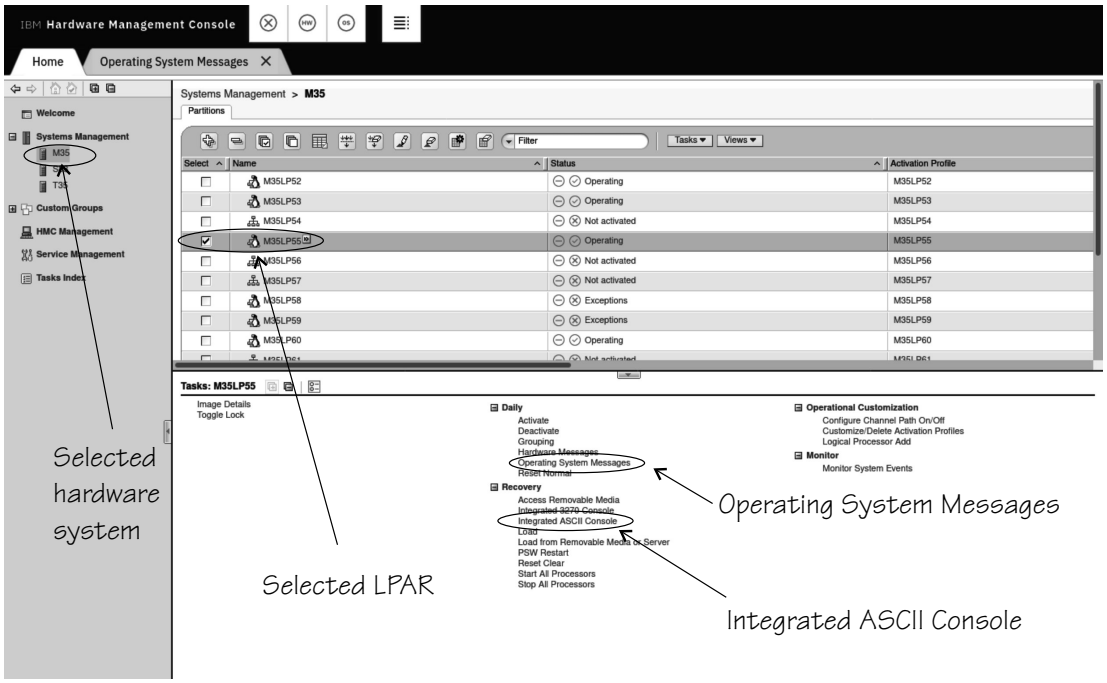


Figure 9. Hardware Management Console

Linux on z/VM

With Linux on z/VM, you typically use a 3270 terminal or terminal emulator to log in to z/VM first. From the 3270 terminal you IPL the Linux boot device. Again, after boot you typically use a network connection to access Linux through a user login rather than a 3270 terminal.

Linux on KVM

You initiate the boot process for Linux as a KVM guest on IBM Z when you start the KVM virtual server through a **virsh** command on the KVM host. The `--console` of the **virsh start** command option gives you access to a terminal that displays the kernel messages.

After the boot process has completed, a guest is usually accessed through a user login, for example, in an SSH session. The possible connections depend on the configuration of your particular Linux instance.

Console features

The console device drivers support several types of terminal devices.

HMC applets
You can use two applets.

Operating System Messages
This applet provides a line-mode terminal for Linux in LPAR mode. See Figure 10 on page 36 for an example.

Integrated ASCII Console
This applet provides a full-screen mode terminal for Linux in LPAR mode and for Linux on z/VM.

These HMC applets are accessed through the service-call logical processor (SCLP) console interface.

3270 terminal
This terminal can be based on physical 3270 terminal hardware or a 3270 terminal emulation.

z/VM can use the 3270 terminal as a 3270 device or perform a protocol translation and use it as a 3215 device. As a 3215 device it is a line-mode terminal for the United States code page (037).

virsh command on the KVM host
For Linux on KVM, you can access the console through a virsh command on the KVM host. See “Using virsh on a KVM host” on page 40.

The console device drivers support these terminals as output devices for Linux kernel messages.

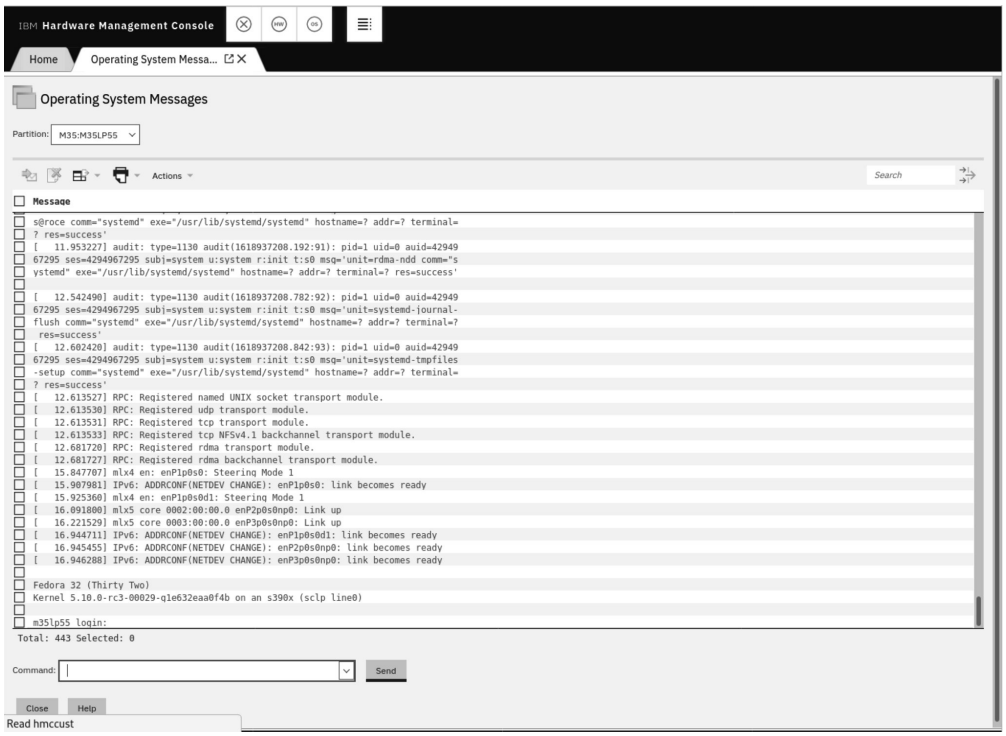


Figure 10. Linux kernel messages on the HMC Operating System Messages applet

What you should know about the console device drivers

The console concepts, naming conventions, and terminology overview help you to understand the tasks you might have to perform with console and terminal devices.

Console terminology

Terminal and *console* have special meanings in Linux.

Linux terminal
An input/output device through which users interact with Linux and Linux applications. Login programs and shells typically run on Linux terminals and provide access to the Linux system.

Linux console
An output-only device to which the Linux kernel can write kernel messages. Linux console devices can be associated with Linux terminal devices. Thus, console output can be displayed on a Linux terminal.

Mainframe terminal
Any device that gives a user access to operating systems and applications that run on a mainframe. A mainframe terminal can be a physical device such as a 3270 terminal hardware that is linked to the mainframe through a controller. It can also be a terminal emulator on a workstation that is connected through a network. For example, you access z/OS® through a mainframe terminal.

Hardware Management Console (HMC)
A device that gives a system programmer control over IBM Z hardware resources, for example, LPARs. The HMC is a web application on a web server that is connected to the Support Element (SE). The HMC can be accessed from the SE but more commonly is accessed from a workstation within a secure network.

On the mainframe, the Linux console and Linux terminals can both be connected to a mainframe terminal.

Before you have a Linux terminal - the zipl boot menu

Do not confuse the zipl boot menu with a Linux terminal.

Depending on your setup, a zipl boot menu might be displayed when you perform an IPL. The zipl boot menu is part of the boot loader that loads the Linux kernel and is displayed before a Linux terminal is set up. The zipl boot menu is very limited in its functions. For example, there is no way to specify uppercase letters because all input is converted to lowercase characters. For more details about booting Linux, see Chapter 7, “Booting Linux,” on page 93. For more details about the zipl boot menu, see Chapter 6, “Initial program loader for IBM Z - zipl,” on page 57.

Device and console names

Each terminal device driver can provide a single console device.

Table 2 on page 37 lists the terminal device drivers with the corresponding device names and console names.

Table 2. Device and console names

Device driver	Device name	Console name
SCLP line-mode terminal device driver	sclp_line0	ttyS0
SCLP VT220 terminal device driver	ttysclp0	ttyS1
3215 line-mode terminal device driver	ttyS0	ttyS0
3270 terminal device driver	3270/tty1 to 3270/tty<*N*>	tty3270
z/VM IUCV HVC device driver	hvc0 to hvc7	hvc0
virtio-console device driver	hvc0 to hvc<*n*>	hvc0

As shown in Table 2 on page 37, the console with name ttyS0 can be provided either by the SCLP console device driver or by the 3215 line-mode terminal device driver. The system environment and settings determine which device driver provides ttyS0. For details, see the information about the conmode kernel parameter in “Console kernel parameter syntax” on page 42.

Of the terminal devices that are provided by the z/VM IUCV HVC device driver only hvc0 is associated with a console.

Of the 3270/tty<*N*> terminal devices only 3270/tty1 is associated with a console.

Device nodes

Applications, for example, login programs, access terminal devices by device nodes.

For example, with the default `conmode` and `hvc_iucv` settings, udev creates the following device nodes:

Table 3. Device nodes created by udev			
Device driver	**On LPAR**	**On z/VM**	**On KVM**
SCLP line-mode terminal device driver	`/dev/sclp_line0`	`/dev/sclp_line0`	`/dev/sclp_line0`
SCLP VT220 terminal device driver	`/dev/ttysclp0`	`/dev/ttysclp0`	`/dev/ttysclp0`
3215 line-mode terminal device driver	n/a	`/dev/ttyS0`	n/a
3270 terminal device driver	`/dev/3270/tty1` to `/dev/3270/tty<N>`	`/dev/3270/tty1` to `/dev/3270/tty<N>`	`/dev/3270/tty1` to `/dev/3270/tty<N>`
z/VM IUCV HVC device driver	n/a	`/dev/hvc0`	n/a
virtio-console device driver	n/a	n/a	`/dev/hvc0` to `/dev/hvc<n>`

Apart from the standard device nodes, there is also a generic device node, `/dev/console`, that maps to the current console. The console device driver itself presents `/dev/console` as a pure input device to the user space. However, through its association with the terminal device driver, it becomes bidirectional.

Terminal modes

The Linux terminals that are provided by the console device drivers include line-mode terminals, block-mode terminals, and full-screen mode terminals.

On a full-screen mode terminal, pressing any key immediately results in data being sent to the terminal. Also, terminal output can be positioned anywhere on the screen. This feature facilitates advanced interactive capability for terminal-based applications like the vi editor.

On a line-mode terminal, the user first types a full line, and then presses Enter to indicate that the line is complete. The device driver then issues a read to get the completed line, adds a new line, and hands over the input to the generic TTY routines.

The terminal that is provided by the 3270 terminal device driver is a traditional IBM mainframe block-mode terminal. Block-mode terminals provide full-screen output support and users can type input in predefined fields on the screen. Other than on typical full-screen mode terminals, no input is passed on until the user presses Enter. The terminal that is provided by the 3270 terminal device driver provides limited support for full-screen applications. For example, the ned editor is supported, but not vi.

Table 4 on page 38 summarizes when to expect which terminal mode.

Table 4. Terminal modes			
Accessed through	**Environment**	**Device driver**	**Mode**
Operating System Messages applet on the HMC	LPAR	SCLP line-mode terminal device driver	Line mode
z/VM emulation of the HMC **Operating System Messages** applet	z/VM	SCLP line-mode terminal device driver	Line mode
Integrated ASCII Console applet on the HMC	z/VM or LPAR	SCLP VT220 terminal device driver	Full-screen mode
KVM host (for example, using the **`virsh console`** command)	KVM	SCLP line-mode terminal device driver	Line mode

Table 4. Terminal modes (continued)			
Accessed through	**Environment**	**Device driver**	**Mode**
KVM host (for example, using the **virsh console** command)	KVM	SCLP VT220 terminal device driver	Full-screen mode
3270 terminal hardware or emulation	z/VM with CONMODE=3215 or KVM	3215 line-mode terminal device driver	Line mode
3270 terminal hardware or emulation	z/VM with CONMODE=3270 or KVM	3270 terminal device driver	Block mode
iucvconn program	z/VM	z/VM IUCV HVC device driver	Full-screen mode
KVM host (for example, using the **virsh console** command)	KVM	virtio-console device driver	Full-screen mode

The 3270 terminal device driver provides three different views. See "Switching the views of the 3270 terminal device driver" on page 49 for details.

How console devices are accessed

How you can access console devices depends on your environment.

The diagrams in the following sections omit device drivers that are not relevant for the particular access scenario.

Using the HMC for Linux in an LPAR

You can use two applets on the HMC to access terminal devices on Linux instances that run directly in an LPAR.

Figure 11 on page 39 shows the possible terminal devices for Linux instances that run directly in an LPAR.

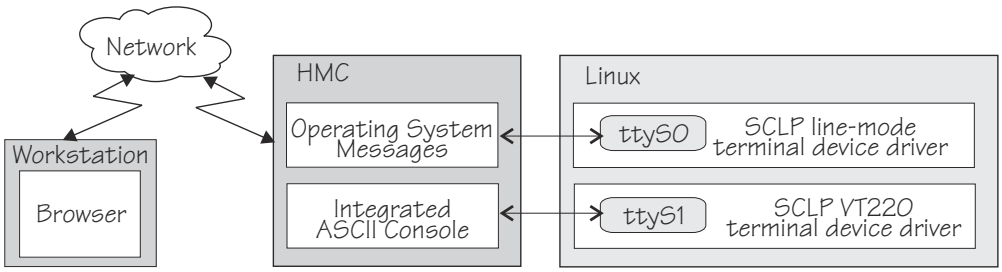


Figure 11. Accessing terminal devices on Linux in an LPAR from the HMC

The **Operating System Messages** applet accesses the device that is provided by the SCLP line-mode terminal device driver. The **Integrated ASCII console** applet accesses the device that is provided by the SCLP VT220 terminal device driver.

Using the HMC for Linux on z/VM

You can use the HMC **Integrated ASCII Console** applet to access terminal devices on Linux instances that run as z/VM guests.

While the ASCII system console is attached to the z/VM guest virtual machine where the Linux instance runs, you can access the ttyS1 terminal device from the HMC **Integrated ASCII Console** applet (see Figure 12 on page 40).

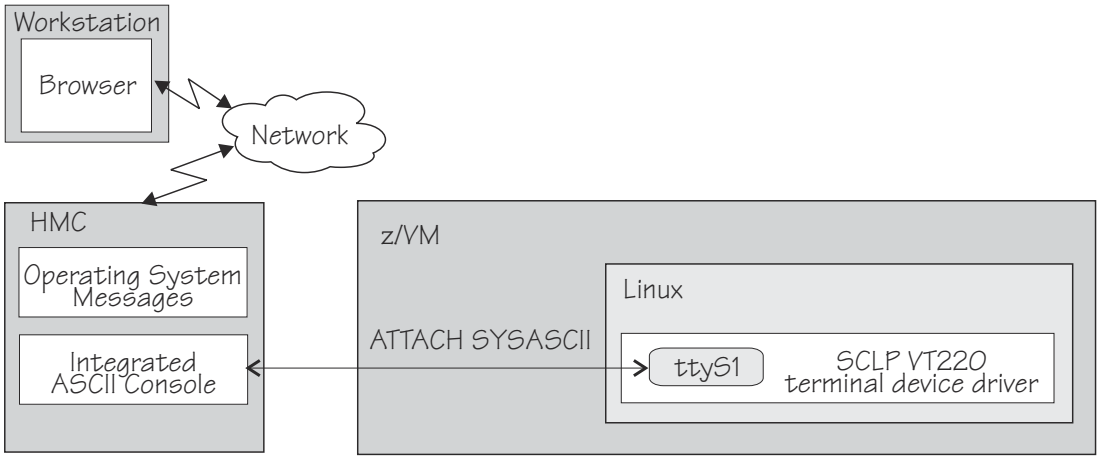


Figure 12. Accessing terminal devices from the HMC for Linux on z/VM

Use the CP ATTACH SYSASCII command to attach the ASCII system console to your z/VM guest virtual machine.

Using virsh on a KVM host

You can use the **`virsh console`** command on a KVM host to access an `sclp` or `virtio` based terminal on a KVM guest.

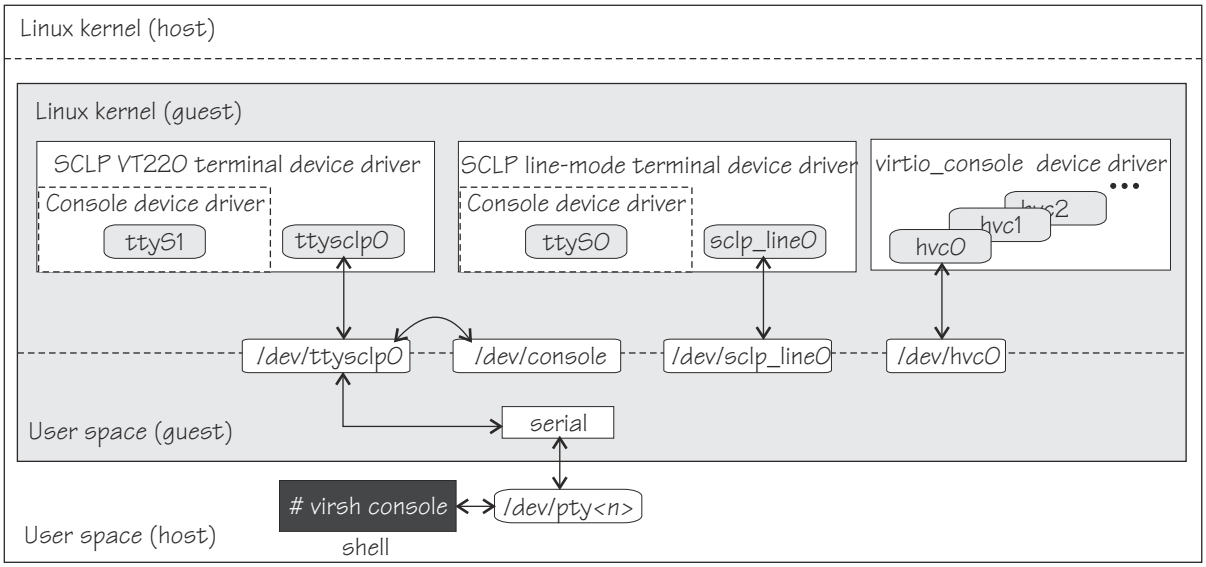


Figure 13. Using virsh to access a KVM guest console

Figure 13 on page 40, shows a KVM guest with three device drivers that can provide a console. The terminal that is accessed by the **`virsh console`** command depends on the guest configuration. For details, see *KVM Virtual Server Management*, SC34-2752.

In a common setup, the **`virsh console`** command opens a connection to the device that is provided by the SCLP VT220 terminal device driver. This device also becomes associated with the generic `/dev/console` device node.

Whether your Linux instance uses this device as the device to which Linux kernel messages are written depends on the Linux configuration. Use the `console=` parameter to control which devices are activated to receive Linux kernel messages (see in "Console kernel parameter syntax" on page 42).

Using a 3270 terminal emulation for Linux on z/VM

For Linux on z/VM, you can use 3270 terminal emulation to access a console device.

Figure 14 on page 41 illustrates how z/VM can handle the 3270 communication.

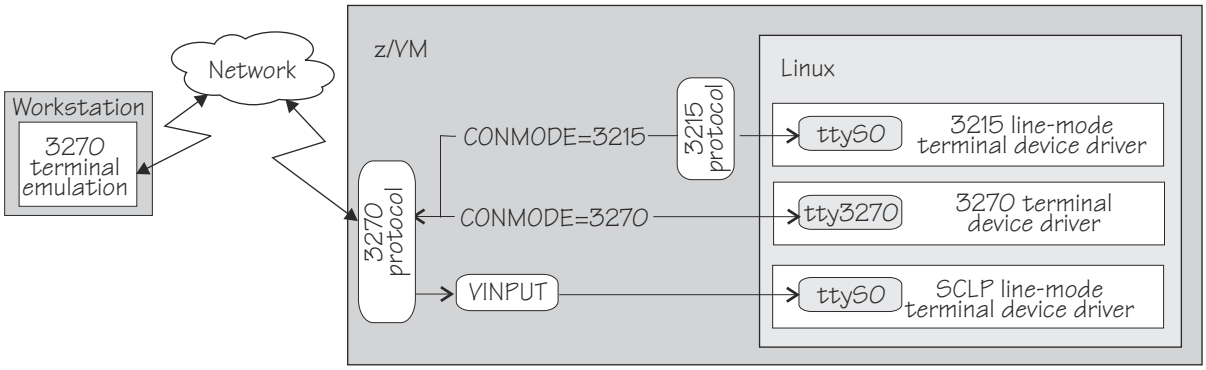


Figure 14. Accessing terminal devices from a 3270 device

Note: Figure 14 on page 41 shows two console devices with the name ttyS0. Only one of these devices can be present at any one time.

CONMODE=3215
translates between the 3270 protocol and the 3215 protocol and connects the 3270 terminal emulation to the 3215 line-mode terminal device driver in the Linux kernel.

CONMODE=3270
connects the 3270 terminal emulation to the 3270 terminal device driver in the Linux kernel.

VINPUT
is a z/VM CP command that directs input to the ttyS0 device provided by the SCLP line-mode terminal device driver. In a default z/VM environment, ttyS0 is provided by the 3215 line-mode terminal device driver. You can use the conmode kernel parameter to make the SCLP line-mode terminal device driver provide ttyS0 (see "Console kernel parameter syntax" on page 42).

The terminal device drivers continue to support 3270 terminal hardware, which, if available at your installation, can be used instead of a 3270 terminal emulation.

Using a 3270 terminal emulation for Linux on KVM

For Linux on IBM Z as a KVM guest, you can use a 3270 terminal emulation to access a console device through the 3270 or 3215 terminal device driver.

"Using a 3270 terminal emulation for Linux on KVM" on page 41 illustrates how Linux on KVM can handle the 3270 communication.

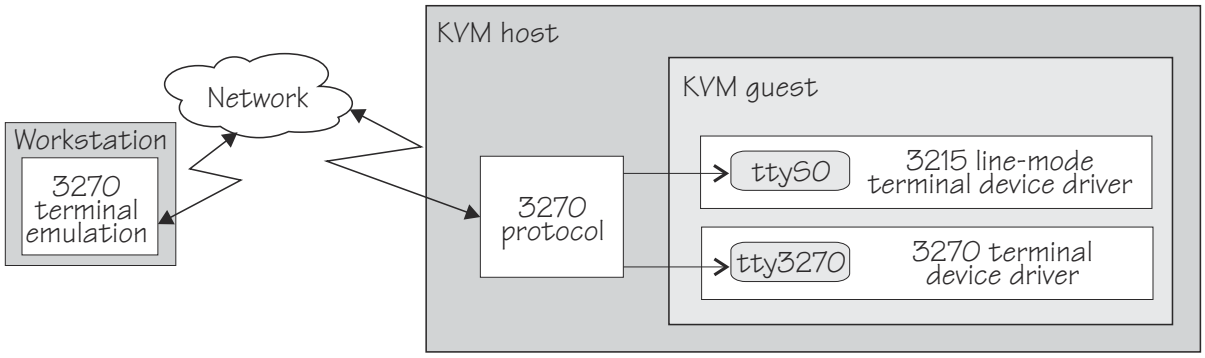


Figure 15. Accessing terminal devices from virt-manager

Using iucvconn on Linux on z/VM

On Linux on z/VM, you can access the terminal devices that are provided by the z/VM IUCV Hypervisor Console (HVC) device driver.

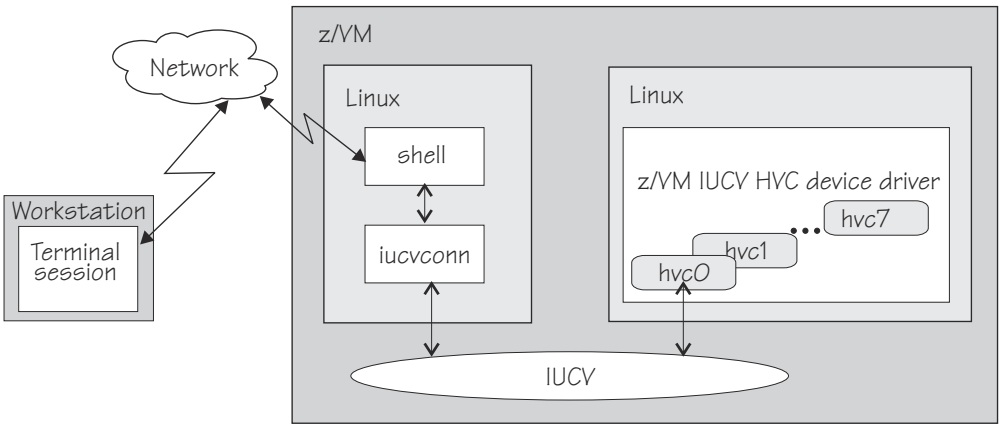


Figure 16. Accessing terminal devices from a peer Linux instance

As illustrated in Figure 16 on page 42, you access the devices with the iucvconn program from another Linux instance. Both Linux instances are guests of the same z/VM system. IUCV provides the communication between the two Linux instances. With this setup, you can access terminal devices on Linux instances with no external network connection.

Note: Of the terminal devices that are provided by the z/VM IUCV HVC device driver only hvc0 can be activated to receive Linux kernel messages.

Setting up the console device drivers

You configure the console device drivers through kernel parameters. You also might have to enable user logins on terminals and ensure that the TERM environment variable has a suitable value.

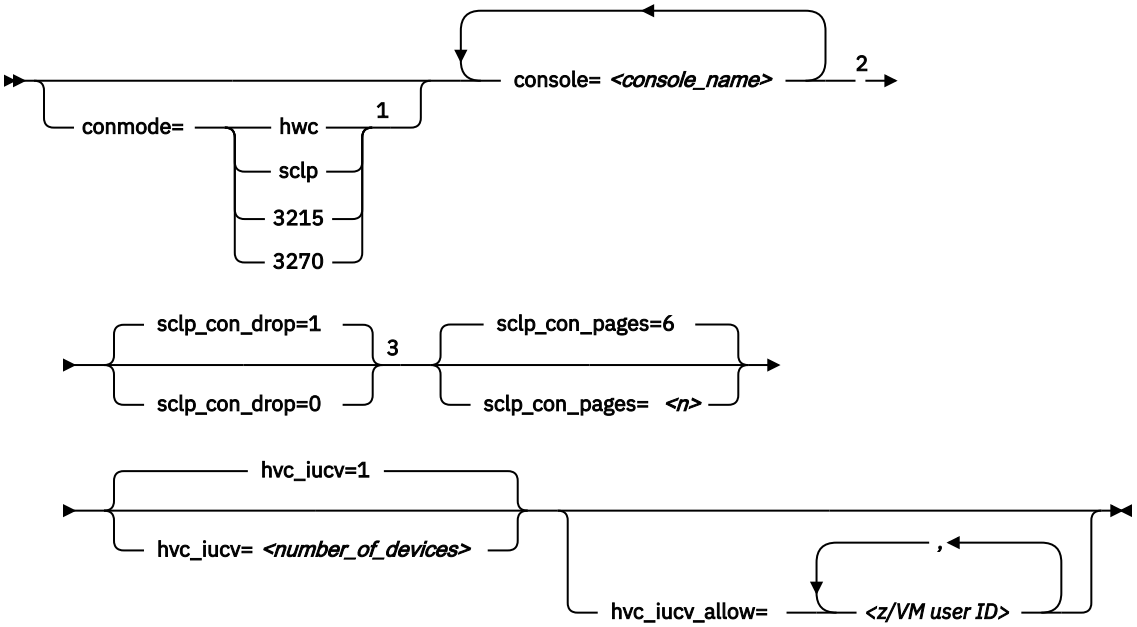
Console kernel parameter syntax

Use the console kernel parameters to configure the console device drivers, line-mode terminals, and HVC terminal devices.

The `sclp_con_pages=` and `sclp_con_drop=` parameters apply only to the SCLP line-mode terminal device driver and the VT220 terminal device driver.

The hvc_iucv= and hvc_iucv_allow= kernel parameters apply only to terminal devices that are provided by the z/VM IUCV HVC device driver.

Console kernel parameter syntax



Notes:

[1] Specify the `conmode=` parameter for Linux on z/VM only.
[2] If you specify both the `conmode=` and the `console=` parameter, specify them in the sequence that is shown, `conmode=` first.
[3] The `sclp_con_pages=` and `sclp_con_drop=` parameters apply only to the SCLP line-mode terminal device driver and to the SCLP VT220 terminal device driver.

where:

conmode
specifies which one of the line-mode or block-mode terminal devices is present and provided by which device driver.

A Linux kernel might include multiple console device drivers that can provide a line-mode terminal:

- SCLP line-mode terminal device driver
- 3215 line-mode terminal device driver
- 3270 terminal device driver

On a running Linux instance, only one of these device drivers can provide a device. Table 5 on page 43 shows how the device driver that is used by default depends on the environment.

Table 5. Default device driver for the line-mode terminal device

Mode	Default
LPAR	SCLP line-mode terminal device driver
z/VM	3215 line-mode terminal device driver or 3270 terminal device driver, depending on the z/VM guest's console settings (the `CONMODE` field in the output of `#CP QUERY TERMINAL`). If the device driver you specify with the conmode= kernel parameter contradicts the CONMODE z/VM setting, z/VM is reconfigured to match the specification for the kernel parameter.
KVM	SCLP line-mode terminal device driver

You can use the `conmode=` parameter to override the default for Linux on z/VM. Do not change the default for Linux on KVM or for Linux in LPAR mode.

sclp or hwc
specifies the SCLP line-mode terminal device driver.

You need this specification if you want to use the z/VM CP VINPUT command ("Using a z/VM emulation of the HMC Operating System Messages applet" on page 52).

3270
specifies the 3270 device driver.

3215
specifies the 3215 device driver.

console=*<console_name>*
specifies the console devices to be activated to receive Linux kernel messages. If present, ttyS0 is always activated to receive Linux kernel messages and, by default, it is also the *preferred* console.

By default, ttyS0 is also the *preferred* console for Linux on z/VM and for Linux in LPAR mode. For Linux on KVM, the default order for the preferred console is ttyS1, followed by ttyS0, followed by hvc0.

The preferred console is used as an initial terminal device, beginning at the stage of the boot process when the initialization procedures run. Messages that are issued by programs that are run at this stage are therefore only displayed on the preferred console. Multiple terminal devices can be activated to receive Linux kernel messages but only one of the activated terminal devices can be the preferred console.

If you specify `conmode=3270`, there is no console with the name `ttyS0`.

If you want console devices other than ttyS0 to be activated to receive Linux kernel messages, specify a console statement for each of these other devices. The last console statement designates the preferred console.

If you specify one or more console parameters and you want to keep ttyS0 as the preferred console, add a console parameter for ttyS0 as the last console parameter. Otherwise, you do not need a console parameter for ttyS0.

<console_name> is the console name that is associated with the terminal device to be activated to receive Linux kernel messages. Of the terminal devices that are provided by the z/VM IUCV HVC device driver only hvc0 can be activated. Specify the console names as shown in Table 2 on page 37.

sclp_con_drop
governs the behavior of the SCLP line-mode terminal device driver and VT220 terminal device driver if either of them runs out of output buffer pages. The trade-off is between slowing down Linux and losing console output. Possible values are `0` and `1` (default).

0
assures complete console output by pausing until used output buffer pages are written to an output device and can be reused without loss.

1
avoids system pauses by overwriting used output buffer pages, even if the content was never written to an output device.

You can use the `sclp_con_pages=` parameter to set the number of output buffers.

sclp_con_pages=*<n>*
specifies the number of 4-KB memory pages to be used as the output buffer for the SCLP line-mode and VT220 terminal. Depending on the line length, each output buffer can hold multiple lines. Use many buffer pages for a kernel with frequent phases of producing console output faster than it can be written to the output device.

Depending on the setting for the `sclp_con_drop=`, running out of pages can slow down Linux or cause it to lose console output.

The value is a positive integer. The default is 6.

hvc_iucv=*<number_of_devices>*
specifies the number of terminal devices that are provided by the z/VM IUCV HVC device driver. *<number_of_devices>* is an integer in the range 0 - 8. Specify 0 to switch off the z/VM IUCV HVC device driver.

hvc_iucv_allow=*<z/VM user ID>*,*<z/VM user ID>*, ...
specifies an initial list of z/VM guest virtual machines that are allowed to connect to HVC terminal devices. If this parameter is omitted, any z/VM guest virtual machine that is authorized to establish the required IUCV connection is also allowed to connect. On the running system, you can change this list with the **chiucvallow** command. See *How to Set up a Terminal Server Environment on z/VM*, SC34-2596 for more information.

Examples

- To activate `ttyS1` in addition to `ttyS0`, and to use `ttyS1` as the preferred console, add the following specification to the kernel command line:

```
console=ttyS1
```

- To activate `ttyS1` in addition to `ttyS0`, and to keep `ttyS0` as the preferred console, add the following specification to the kernel command line:

```
console=ttyS1 console=ttyS0
```

- To use an emulated HMC Operating System Messages applet in a z/VM environment specify:

```
conmode=sclp
```

- To activate `hvc0` in addition to `ttyS0`, use `hvc0` as the preferred console, configure the z/VM IUCV HVC device driver to provide four devices, and limit the z/VM guest virtual machines that can connect to HVC terminal devices to `lxtserv1` and `lxtserv2`, add the following specification to the kernel command line:

```
console=hvc0 hvc_iucv=4 hvc_iucv_allow=lxtserv1,lxtserv2
```

- The following specification selects the SCLP line-mode terminal and configures 32 4-KB pages (128 KB) for the output buffer. If buffer pages run out, the SCLP line-mode terminal device driver does not wait for pages to be written to an output device. Instead of pausing, it reuses output buffer pages at the expense of losing content.

```
console=sclp sclp_con_pages=32 sclp_con_drop=1
```

Setting up a z/VM guest virtual machine for iucvconn

Because the iucvconn program uses z/VM IUCV to access Linux, you must set up your z/VM guest virtual machine for IUCV.

See “Setting up your z/VM guest virtual machine for IUCV” on page 334 for details about setting up the z/VM guest virtual machine.

For information about accessing Linux through the iucvtty program rather than through the z/VM IUCV HVC device driver, see *How to Set up a Terminal Server Environment on z/VM*, SC34-2596 or the man pages for the **iucvtty** and **iucvconn** commands.

Setting up a line-mode terminal

The line-mode terminals are primarily intended for booting Linux.

The preferred user access to a running instance of Red Hat Enterprise Linux 9.1 is through a user login that runs, for example, in an SSH session. See “Terminal modes” on page 38 for information about the available line-mode terminals.

Tip: If the terminal does not provide the expected output, ensure that `dumb` is assigned to the `TERM` environment variable. For example, enter the following command on the bash shell:

```
# export TERM=dumb
```

Setting up a full-screen mode terminal

The full-screen terminal can be used for full-screen text editors, such as `vi`, and terminal-based full-screen system administration tools.

See “Terminal modes” on page 38 for information about the available full-screen mode terminals.

Tip: If the terminal does not provide the expected output, ensure that `linux` is assigned to the `TERM` environment variable. For example, enter the following command on the bash shell:

```
# export TERM=linux
```

Setting up a terminal provided by the 3270 terminal device driver

The terminal that is provided by the 3270 terminal device driver is not a line-mode terminal, but it is also not a typical full-screen mode terminal.

The terminal provides limited support for full-screen applications. For example, the ned editor is supported, but not vi.

Tip: If the terminal does not provide the expected output, ensure that `linux` is assigned to the `TERM` environment variable. For example, enter the following command on the bash shell:

```
# export TERM=linux
```

Enabling user logins

Instantiate getty services for terminals to allow users access.

For the default terminals (see “Device nodes” on page 37) systemd starts a getty. In particular, you do not need to enable user logins for the following terminals:

- `hvc0`
- `3270/tty1`
- SCLP-based terminals

For other terminals, such as hvc1 to hvc7 or dynamically attached 3270 terminals, you must create a symbolic link.

Procedure

To create a symbolic link and start a getty on terminal hvc1:

1. Create a new terminal instance, hvc1, for a serial getty. For example, issue:

   ```
   # systemctl enable serial-getty@hvc1.service
   ```

2. Start the new instance with systemctl.
   For example:

   ```
   # systemctl start serial-getty@hvc1.service
   ```

   The terminal instance starts automatically at system start when the `getty.target` is processed.

Preventing respawns for non-operational HVC terminals

If you enable user logins on a terminal that is not available or not operational, systemd keeps respawning the getty service.

About this task

If user logins are enabled on unavailable HVC terminals `hvc1` to `hvc7`, systemd might keep respawning the getty program. To be free to change the conditions that affect the availability of these terminals, use the ttyrun service to enable user logins for them. HVC terminals are operational only in a z/VM environment, and they depend on the `hvc_iucv=` kernel parameter (see "Console kernel parameter syntax" on page 42).

Any other unavailable terminals with enabled user login, including `hvc0`, do not cause problems with systemd.

Procedure

Perform these steps to use a ttyrun service for enabling user logins on a terminal:

1. Enable the ttyrun service by issuing a command of this form:

```
# systemctl enable ttyrun-getty@hvc<n>.service
```

   where hvc<*n*> specifies one of the terminals `hvc1` to `hvc7`.
2. Optional: Start the new service by issuing a command of this form:

```
# systemctl start ttyrun-getty@hvc<n>.service
```

Results

At the next system start, systemd starts the ttyrun service for hvc<*n*>. The ttyrun service starts a getty only if this terminal is available.

Example

For `hvc1`, issue:

```
# systemctl enable ttyrun-getty@hvc1.service
# systemctl start ttyrun-getty@hvc1.service
```

Setting up the code page for an x3270 emulation on Linux

For accessing z/VM from Linux through the x3270 terminal emulation, you must add a number of settings to the `.Xdefaults` file to get the correct code translation.

Add these settings:

```
   ! X3270 keymap and charset settings for Linux
   x3270.charset: us-intl
   x3270.keymap: circumfix
   x3270.keymap.circumfix: :<key>asciicircum: Key("^")\n
```

Working with Linux terminals

You might have to work with different types of Linux terminals, and use special functions on these terminals.

- "Using the terminal applets on the HMC" on page 48
- "Accessing terminal devices over z/VM IUCV" on page 48

Using the terminal applets on the HMC

You should be aware of some aspects of the line-mode and the full-screen mode terminal when working with the corresponding applets on the HMC.

The following statements apply to both the line-mode terminal and the full-screen mode terminal on the HMC:

- On an HMC, you can open each applet only once.
- Within an LPAR, there can be only one active terminal session for each applet, even if multiple HMCs are used.
- A particular Linux instance supports only one active terminal session for each applet.
- Slow performance of the HMC is often due to a busy console or increased network traffic.

The following statements apply to the full-screen mode terminal only:

- Output that is written by Linux while the terminal window is closed is not displayed. Therefore, a newly opened terminal window is always blank. For most applications, like login or shell prompts, it is sufficient to press Enter to obtain a new prompt.
- The terminal window shows only 24 lines and does not provide a scroll bar. To scroll up, press Shift+PgUp; to scroll down, press Shift+PgDn.

Security hint: Always end a terminal session by explicitly logging off (for example, type "exit" and press Enter). Simply closing the applet leaves the session active and the next user to open the applet resumes the existing session without a logon.

Accessing terminal devices over z/VM IUCV

Use z/VM IUCV to access hypervisor console (HVC) terminal devices, which are provided by the z/VM IUCV HVC device driver.

About this task

For information about accessing terminal devices that are provided by the iucvtty program see *How to Set up a Terminal Server Environment on z/VM*, SC34-2596.

You access HVC terminal devices from a Linux instance where the iucvconn program is installed. The Linux instance with the terminal device to be accessed and the Linux instance with the iucvconn program must both run as guests of the same z/VM system. The two guest virtual machines must be configured such that IUCV communication is permitted between them.

Procedure

Perform these steps to access an HVC terminal device over z/VM IUCV:

1. Open a terminal session on the Linux instance where the iucvconn program is installed.
2. Enter a command of this form:

```
# iucvconn <guest_ID> <terminal_ID>
```

where:

<guest_ID>
specifies the z/VM guest virtual machine on which the Linux instance with the HVC terminal device to be accessed runs.

<terminal_ID>
specifies an identifier for the terminal device to be accessed. HVC terminal device names are of the form hvc*n* where *n* is an integer in the range 0-7. The corresponding terminal IDs are lnxhvc*n*.

Example: To access HVC terminal device hvc0 on a Linux instance that runs on a z/VM guest virtual machine LXGUEST1, enter:

```
# iucvconn LXGUEST1 lnxhvc0
```

For more details and further parameters of the **iucvconn** command, see the **iucvconn** man page or *How to Set up a Terminal Server Environment on z/VM*, SC34-2596.

3. Press Enter to obtain a prompt.

   Output that is written by Linux while the terminal window is closed is not displayed. Therefore, a newly opened terminal window is always blank. For most applications, like login or shell prompts, it is sufficient to press Enter to obtain a new prompt.

Security hint

Always end terminal sessions by explicitly logging off (for example, type **exit** and press Enter). If logging off results in a new login prompt, press Control and Underscore (Ctrl+_), then press D to close the login window. Simply closing the terminal window for a hvc0 terminal device that was activated for Linux kernel messages leaves the device active. The terminal session can then be reopened without a login.

Switching the views of the 3270 terminal device driver

The 3270 terminal device driver provides a terminal with three different views for Linux on z/VM.

Use function key 3 (PF3) to switch between the views (see Figure 17 on page 49).

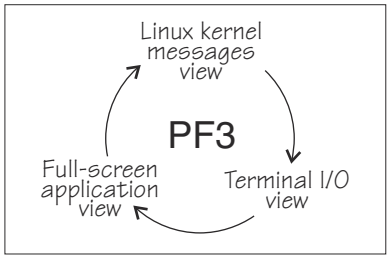


Figure 17. Switching views of the 3270 terminal device driver

The Linux kernel messages view is available only if the terminal device is activated for Linux kernel messages. The full-screen application view is available only if there is an application that uses this view, for example, the ned editor.

Be aware that the 3270 terminal provides only limited full-screen support. The full-screen application view of the 3270 terminal is not intended for applications that require vt220 capabilities. The application itself must create the 3270 data stream.

For the Linux kernel messages view and the terminal I/O view, you can use the PF7 key to scroll backward and the PF8 key to scroll forward. The scroll buffers are fixed at four pages (16 KB) for the Linux kernel messages view and five pages (20 KB) for the terminal I/O view. When the buffer is full and more terminal data needs to be printed, the oldest lines are removed until there is enough room. The number of lines in the history, therefore, vary. Scrolling in the full-screen application view depends on the application.

You cannot issue z/VM CP commands from any of the three views that are provided by the 3270 terminal device driver. If you want to issue CP commands, use the PA1 key to switch to the CP READ mode.

Setting a CCW terminal device online or offline

The 3270 terminal device driver uses CCW devices and provides them as CCW terminal devices.

About this task

This section applies to Linux on z/VM. A CCW terminal device can be:

- The tty3270 terminal device that can be activated for receiving Linux kernel messages.

  If this device exists, it comes online early during the Linux boot process. In a default z/VM environment, the device number for this device is 0009. In sysfs, it is represented as `/sys/bus/ccw/drivers/3270/0.0.0009`. You need not set this device online and you must not set it offline.
- CCW terminal devices through which users can log in to Linux with the CP DIAL command.

  These devices are defined with the CP DEF GRAF command. They are represented in sysfs as `/sys/bus/ccw/drivers/3270/0.`*<n>*`.`*<devno>* where *<n>* is the subchannel set ID and *<devno>* is the virtual device number. By setting these devices online, you enable them for user logins. If you set a device offline, it can no longer be used for user login.

See *z/VM: CP Commands and Utilities Reference*, SC24-6268 for more information about the DEF GRAF and DIAL commands.

Procedure

You can use the **chccwdev** command (see "chccwdev - Set CCW device attributes" on page 575) to set a CCW terminal device online or offline. Alternatively, you can write `1` to the device's online attribute to set it online, or `0` to set it offline.

Examples

- To set a CCW terminal device `0.0.7b01` online, issue:

```
# chccwdev -e 0.0.7b01
```

  Alternatively issue:

```
# echo 1 > /sys/bus/ccw/drivers/3270/0.0.7b01/online
```

- To set a CCW terminal device `0.0.7b01` offline, issue:

```
# chccwdev -d 0.0.7b01
```

  Alternatively issue:

```
# echo 0 > /sys/bus/ccw/drivers/3270/0.0.7b01/online
```

Entering control and special characters on line-mode terminals

Line-mode terminals do not have a control (Ctrl) key. Without a control key, you cannot enter control characters directly.

Also, pressing the Enter key adds a newline character to your input string. Some applications do not tolerate such trailing newline characters.

Table 6 on page 51 summarizes how you can use the caret character (^) to enter some control characters and to enter strings without appended newline characters.

Table 6. Control and special characters on line-mode terminals

For the key combination	Enter	Usage
Ctrl+C	^c	Cancel the process that is running in the foreground of the terminal.
Ctrl+D	^d	Generate an end of file (EOF) indication.
Ctrl+Z	^z	Stop a process.
n/a	^n	Suppresses the automatic generation of a new line. Thus, you can enter single characters; for example, the characters that are needed for yes/no answers in some utilities.

Note: For a 3215 line-mode terminal in 3215 mode, you must use United States code page (037).

Using the magic sysrequest feature

The Linux on IBM Z terminal device drivers support the magic sysrequest functions.

- To call the magic sysrequest functions on the VT220 terminal or on hvc0, enter the single character Ctrl+o followed by the character for the particular function.

  You can call the magic sysrequest functions from the hvc0 terminal device if it is present and is activated to receive Linux kernel messages.
- To call the magic sysrequest functions on a line-mode terminal, enter the 2 characters "^-" (caret and hyphen) followed by a third character that specifies the particular function.

Table 7 on page 51 provides an overview of the commands for the magic sysrequest functions:

Table 7. Magic sysrequest functions

On line-mode terminals, enter	On hvc0 and the VT220 terminal, enter	To
^-b	Ctrl+o b	Re-IPL immediately (see “lsreipl - List IPL and re-IPL settings” on page 672).
^-s	Ctrl+o s	Emergency sync all file systems.
^-u	Ctrl+o u	Emergency remount all mounted file systems read-only.
^-t	Ctrl+o t	Show task info.
^-m	Ctrl+o m	Show memory.
^- followed by a digit (0 - 9)	Ctrl+o followed by a digit (0 - 9)	Set the console log level.
^-e	Ctrl+o e	Send the TERM signal to end all tasks except init.
^-i	Ctrl+o i	Send the KILL signal to end all tasks except init.
^-p	Ctrl+o p	See “Obtaining details about the CPU-measurement facilities” on page 545.

Note: In Table 7 on page 51 Ctrl+o means pressing O while holding down the control key.

Table 7 on page 51 lists the main magic sysrequest functions that are known to work on Red Hat Enterprise Linux 9.1. For a more comprehensive list of functions, see `Documentation/sysrq.txt` in the Linux source tree. Some of the listed functions might not work on your system.

Activating and deactivating the magic sysrequest feature

Use the `sysrq` procfs attribute to activate or deactivate the magic sysrequest feature.

Procedure

Issue the following command to activate the magic sysrequest function:

```
echo 1 > /proc/sys/kernel/sysrq
```

Issue the following command to deactivate the magic sysrequest feature:

```
echo 0 > /proc/sys/kernel/sysrq
```

Alternatively you can use **`sysctl`** to activate and deactivate the magic sysrequest feature. To check how the magic sysrequest function is set, issue:

```
# sysctl kernel.sysrq
kernel.sysrq = 1
```

In Red Hat Enterprise Linux 9.1 the magic sysrequest function is turned on by default. To turn it off using sysctl, issue:

```
# sysctl -w kernel.sysrq=0
```

Triggering magic sysrequest functions from procfs

You can trigger the magic sysrequest functions through procfs.

Procedure

Write the character for the particular function to `/proc/sysrq-trigger`.

You can use this interface even if the magic sysrequest feature is not activated as described in “Activating and deactivating the magic sysrequest feature” on page 52.

Example

To set the console log level to 9, enter:

```
# echo 9 > /proc/sysrq-trigger
```

Using a z/VM emulation of the HMC Operating System Messages applet

You can use the **Operating System Messages** applet emulation; for example, if the 3215 terminal is not operational.

About this task

The preferred terminal devices for Linux instances that run as z/VM guests are provided by the 3215 or 3270 terminal device drivers.

The emulation requires a terminal device that is provided by the SCLP line-mode terminal device driver. To use the emulation, you must override the default device driver for z/VM environments (see “Console kernel parameter syntax” on page 42).

For the emulation, you use the z/VM CP VINPUT command instead of the graphical user interface at the Support Element or HMC. Type any input to the operating system with a leading CP VINPUT.

The examples in the sections that follow show the input line of a 3270 terminal or terminal emulator (for example, x3270). Omit the leading #CP if you are in CP read mode. For more information about VINPUT, see *z/VM: CP Commands and Utilities Reference*, SC24-6268.

Priority and non-priority commands

VINPUT commands require a VMSG (non-priority) or PVMSG (priority) specification.

Operating systems that accept this specification, process priority commands with a higher priority than non-priority commands.

The hardware console driver can accept both if supported by the hardware console within the specific machine or virtual machine.

Linux does not distinguish between priority and non-priority commands.

Example

The specifications:

```
#CP VINPUT VMSG LS -L
```

and

```
#CP VINPUT PVMSG LS -L
```

are equivalent.

Case conversion

All lowercase characters are converted by z/VM to uppercase. To compensate for this effect, the console device driver converts all input to lowercase.

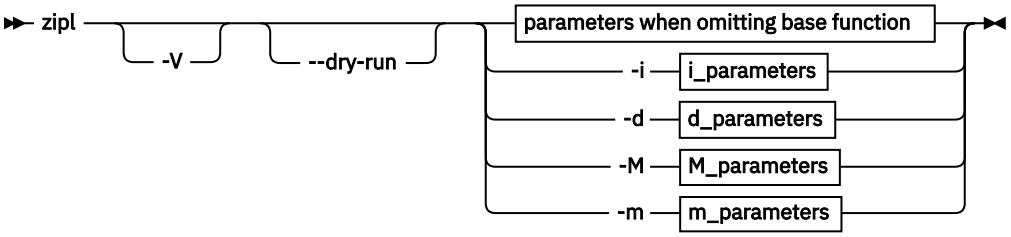
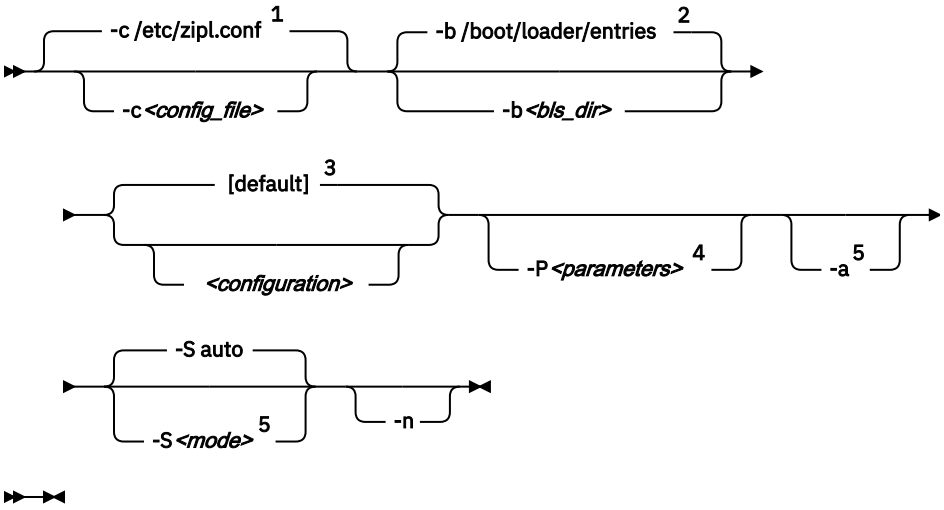
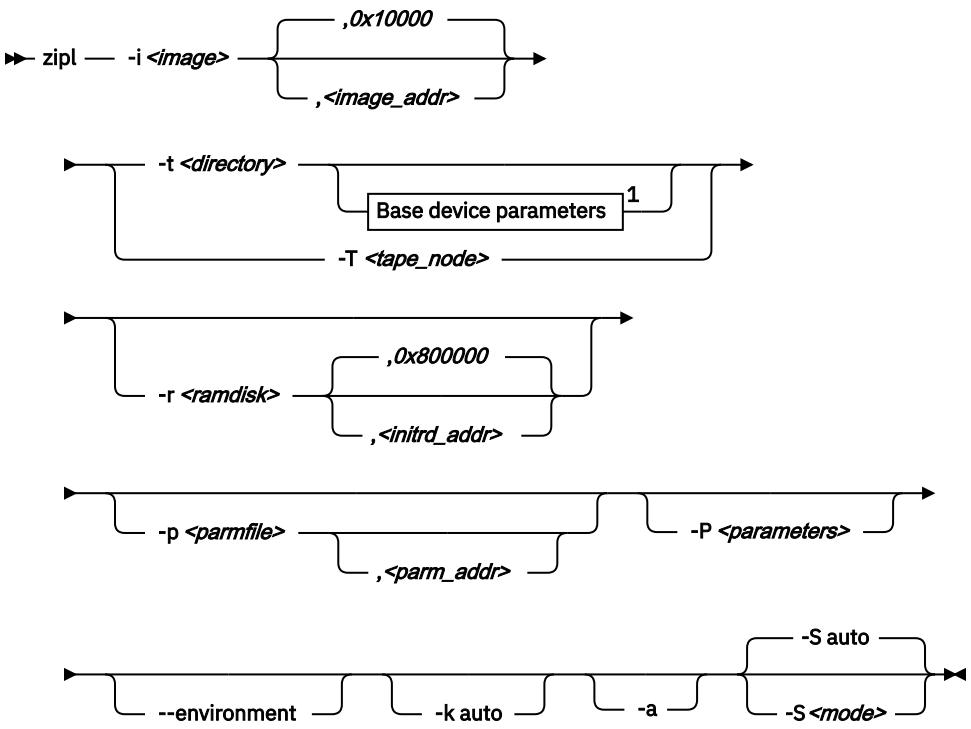
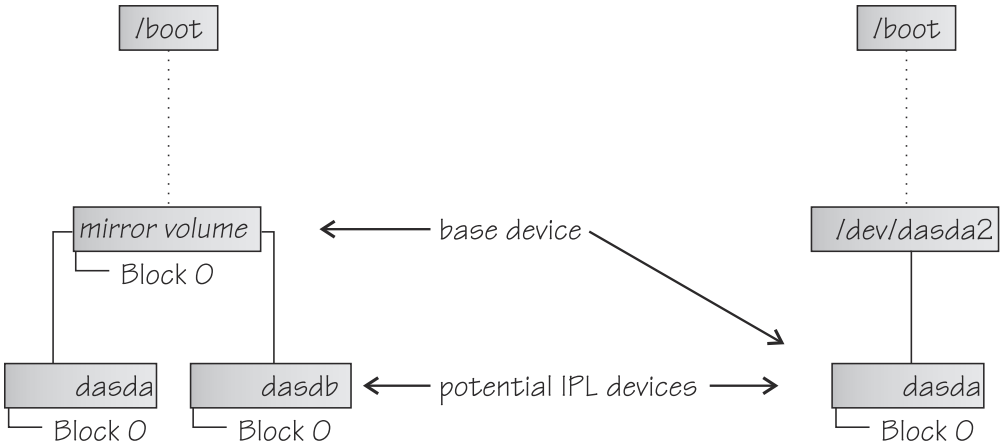
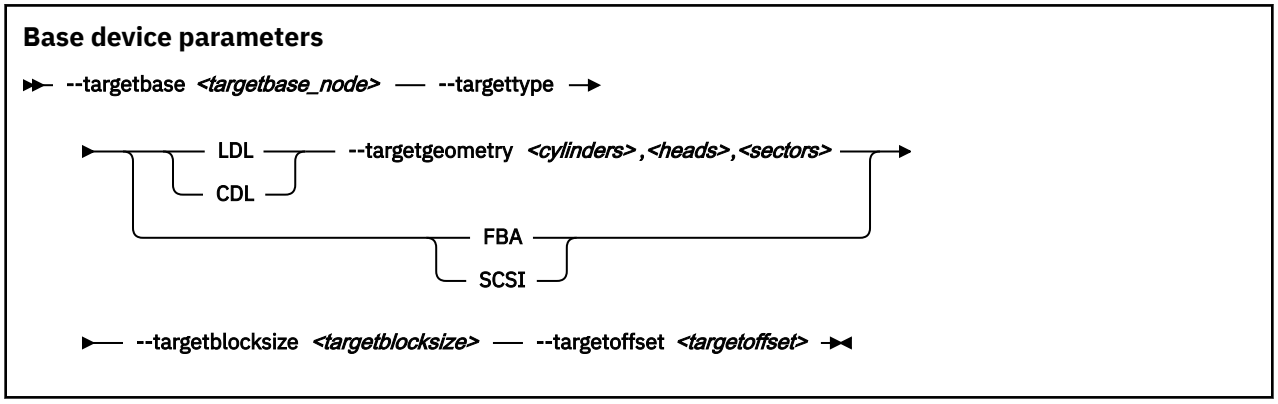
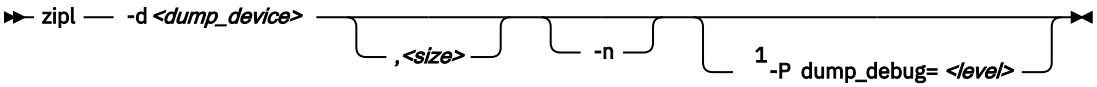
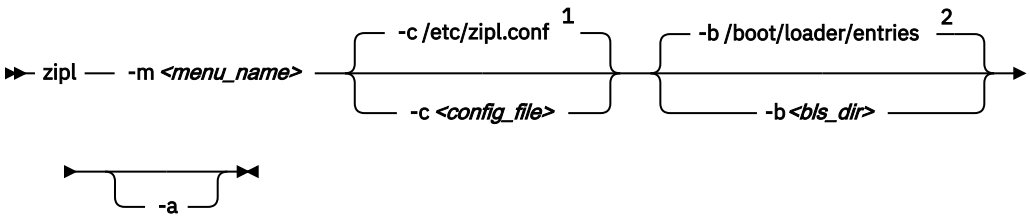
For example, if you type `VInput VMSG echo $PATH`, the device driver gets `ECHO $PATH` and converts it into `echo $path`.

Linux and bash are case-sensitive and require some specifications with uppercase characters. To include uppercase characters in a command, use the percent sign (%) as a delimiter. The console device driver interprets characters that are enclosed by percent signs as uppercase.

Examples

In the following examples, the first line shows the user input. The second line shows what the device driver receives after the case conversion by CP. The third line shows the command that is processed by bash:

- ```
  #cp vinput vmsg ls -l
  CP VINPUT VMSG LS -L
  ls -l
  ...
  ```
- The following input would result in a bash command that contains a variable $path, which is not defined in lowercase:

```
#cp vinput vmsg echo $PATH
CP VINPUT VMSG ECHO $PATH
echo $path
...
```
```

To obtain the correct bash command enclose the uppercase string with the conversion escape character:

```
#cp vinput vmsg echo $%PATH%
CP VINPUT VMSG ECHO $%PATH%
echo $PATH
...
```

Using the escape character

The quotation mark (") is the standard CP escape character. To include the escape character in a command that is passed to Linux, you must type it twice.

Example

The following command passes a string in double quotation marks to be echoed.

```
#cp vinput pvmsg echo ""here is ""$0
CP VINPUT PVMSG ECHO "HERE IS "$0
echo "here is "$0
here is -bash
```

In the example, $0 resolves to the name of the current process.

Using the end-of-line character

To include the end-of-line character in the command that is passed to Linux, you must specify it with a leading escape character.

If you are using the standard settings according to “Using a 3270 terminal in 3215 mode” on page 55, you must specify "# to pass # to Linux.

If you specify the end-of-line character without a leading escape character, z/VM CP interprets it as an end-of-line character that ends the **VINPUT** command.

Example

In this example a number sign is intended to mark the begin of a comment in the bash command. This character is misinterpreted as the beginning of a second command.

```
#cp vinput pvmsg echo ""%N%umber signs start bash comments"" #like this one
CP VINPUT PVMSG ECHO "%N%UMBER SIGNS START BASH COMMENTS"
LIKE THIS ONE
HCPCMD001E Unknown CP command: LIKE
...
```

The escape character prevents the number sign from being interpreted as an e character.

```
#cp vinput pvmsg echo ""%N%umber signs start bash comments"" "#like this one
VINPUT PVMSG ECHO "%N%UMBER SIGNS START BASH COMMENTS" #LIKE THIS ONE
echo "Number signs start bash comments" #like this one
Number signs start bash comments
```

Simulating the Enter and Spacebar keys

You can use the **CP VINPUT** command to simulate the Enter and Spacebar keys.

Simulate the Enter key by entering a blank followed by \n:

```
#CP VINPUT VMSG \n
```

Simulate the Spacebar key by entering two blanks followed by \n:

```
#CP VINPUT VMSG  \n
```

Using a 3270 terminal in 3215 mode

The z/VM control program (CP) defines five characters as line-editing symbols. Use the **CP QUERY TERMINAL** command to see the current settings.

The default line-editing symbols depend on your terminal emulator. You can reassign the symbols by changing the settings of LINEND, TABCHAR, CHARDEL, LINEDEL, or ESCAPE with the **CP TERMINAL** command. Table 8 on page 55 shows the most commonly used settings:

Table 8. Line edit characters

Character	Symbol	Usage
#	LINEND	The end of line character. With this character, you can enter several logical lines at once.
|	TABCHAR	The logical tab character.
@	CHARDEL	The character delete symbol deletes the preceding character.
[or ¢	LINEDEL	The line delete symbol deletes everything back to and including the previous LINEND symbol or the start of the input. "[" is common for ASCII terminals and "¢" for EBCDIC terminals.
"	ESCAPE	The escape character. With this character, you can enter a line-edit symbol as a normal character.

To enter a line-edit symbol, you must precede it with the escape character. In particular, to enter the escape character you must type it twice.

Examples

The following examples assume the settings of Table 8 on page 55 with the opening square bracket character ([) as the "delete line" character.

- To specify a tab character, specify:

```
"|
```

- To specify a double quotation mark character, specify:

```
""
```

- If you type the character string:

```
#CP HALT#CP ZIPL 190[#CP IPL 1@290 PARM vmpoff=""MSG OP REBOOT"#IPL 290""
```

  the actual commands that are received by CP are:

```
CP HALT
CP IPL 290 PARM vmpoff="MSG OP REBOOT#IPL 290"
```

Chapter 6. Initial program loader for IBM Z - zipl

Use **zipl** to prepare a boot device (with a Linux program loader) or to prepare a dump device.

Linux on IBM Z as a KVM guest does not support dump devices with stand-alone dump tools. Instead of preparing a dump device with the zipl tool you can also use the kdump infrastructure. To use kdump, no preparation with zipl is necessary. For more information about the kdump infrastructure and the dump tools that **zipl** installs, see *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751.

You can simulate a **zipl** command to test a configuration before you apply the command to an actual device (see dry-run).

Tip: Use the -V option to confirm that the **zipl** command works as intended.

zipl supports the following devices for Linux on z/VM and Linux in LPAR mode:

- Enhanced Count Key Data (ECKD) DASDs with fixed block Linux disk layout (LDL)
- ECKD DASDs with z/OS-compliant compatible disk layout (CDL)
- Fixed Block Access (FBA) DASDs
- Magnetic tape subsystems compatible with IBM3480, IBM3490, or IBM3590
- SCSI disk with PC-BIOS disk layout or GPT layout
- PCIe-attached NVMe devices.

For supported IPL devices for Linux on KVM, see *KVM Virtual Server Management*, SC34-2752.

Usage

The **zipl** tool has base functions that can be called from the command line or in configuration-file mode. There are generic parameters and parameters that are specific to particular base functions.

zipl base functions

For each base function, there is a short and a long command-line option and, with one exception, a corresponding configuration-file option.

Table 9. zipl base functions

Base function	Command line short option	Command line long option	Configuration file option	Environment
Install a boot loader See “Preparing a boot device” on page 60 for details.	`-i`	`--image`	`image=`	LPAR z/VM KVM
Prepare a DASD, SCSI, NVMe, or tape dump device See “Preparing a dump device” on page 67 for details.	`-d`	`--dumpto`	`dumpto=`	LPAR z/VM
Prepare a list of ECKD volumes for a multi-volume dump See “Preparing a multi-volume dump on ECKD DASD” on page 69 for details.	`-M`	`--mvdump`	`mvdump=`	LPAR z/VM

Table 9. zipl base functions (continued)

Base function	Command line short option	Command line long option	Configuration file option	Environment
Install a menu configuration See “Installing a menu configuration” on page 70 for details.	`-m`	`--menu`	(None)	LPAR z/VM KVM

zipl modes and syntax overview

zipl can operate in command-line mode or in configuration-file mode.

Command-line mode

To run **zipl** in command-line mode, specify one of the following base functions:

-i
see “Preparing a boot device” on page 60

-d
see “Preparing a dump device” on page 67

-M
see “Preparing a multi-volume dump on ECKD DASD” on page 69

Configuration-file mode

To run **zipl** in configuration-file mode, omit the base function or specify the **-m** base function (see “Installing a menu configuration” on page 70).

In this mode, zipl processes a zipl configuration file and, optionally, one or more Boot Loader Specification (BLS) snippets. BLS snippets are always processed as part of a menu configuration.

For more information about the zipl configuration file, see “zipl configuration file structure” on page 75.

For more information about BLS snippets, see “BLS configuration snippets” on page 79.

zipl syntax overview



parameters when omitting base function



Notes:

[1] You can change the default configuration file with the ZIPLCONF environment variable.
[2] You can change the default directory with the BOOT environment variable.
[3] If no configuration is specified, **zipl** uses the configuration in the [defaultboot] section of the configuration file (see "zipl configuration file structure" on page 75).
[4] In a boot configuration only.
[5] In a boot configuration or a menu configuration only.

Where:

-c *<config_file>*
specifies the zipl configuration file to be used.

-b *<bls_dir>*
specifies a directory to be searched for files with BLS snippets.

<configuration>
identifies a particular IPL or menu configuration in a zipl configuration-file.

-P *<parameters>*

can optionally be used to provide kernel parameters in a boot configuration section. See "How kernel parameters from different sources are combined" on page 63 for information about how kernel parameters specified with the **-P** option are combined with any kernel parameters specified in the configuration file.

If you provide multiple parameters, separate them with a blank and enclose them within single quotation marks (') or double quotation marks (").

-a
in a boot configuration section, adds kernel image, kernel parameter file, and initial RAM disk to the bootmap file. Use this option when these files are spread across multiple disks to ensure that they are available at IPL time. Specifying this option significantly increases the size of the bootmap file that is created in the target directory.

-S *<mode>* or --secure *<mode>*
SCSI IPL disk device for LPAR only: Controls the format of the boot data that zipl writes to the IPL device. *<mode>* takes the following values:

auto
Uses the secure-boot enabled format if the zipl command is issued on a mainframe with secure-boot support. This is the default.

1
Enforces the secure-boot enabled format regardless of mainframe support. Use this option to prepare boot devices for systems other than the one you are working on. Disks with this format cannot be booted on machines z14 or earlier.

0
Enforces the traditional format, that does not support secure boot, regardless of mainframe support. Disks with this format can be booted on all machines but cannot be used for secure boot.

For more information about secure boot, see “Secure boot” on page 102.

-n
suppresses confirmation prompts that require operator responses to allow unattended processing (for example, for processing DASD or tape dump configuration sections).

-V
provides verbose command output.

--dry-run
simulates a **zipl** command. Use this option to test a configuration without overwriting data on your device.

During simulation, **zipl** performs all command processing and issues error messages where appropriate. Data is temporarily written to the target directory and is cleared up when the command simulation is completed.

-v
displays version information.

-h
displays help information.

The basic functions and their parameters are described in detail in the following sections.

See “Parameter overview” on page 71 for a summary of the short and long command-line options and their configuration file equivalents.

Examples

- To process the default configuration in the default configuration file (`/etc/zipl.conf`, unless specified otherwise with the environment variable ZIPLCONF) issue:

```
# zipl
```

- To process the default configuration in a configuration file `/etc/myxmp.conf` issue:

```
# zipl -c /etc/myxmp.conf
```

- To process a configuration [myconf] in the default configuration file issue:

```
# zipl myconf
```

- To process a configuration [myconf] in a configuration file `/etc/myxmp.conf` issue:

```
# zipl -c /etc/myxmp.conf myconf
```

- To simulate processing a configuration [myconf] in a configuration file `/etc/myxmp.conf` issue:

```
# zipl --dry-run -c /etc/myxmp.conf myconf
```

Preparing a boot device

Use **zipl** with the **-i** (**--image**) command-line option or with the **image=** configuration-file option to prepare a boot device.

zipl command line syntax for preparing a boot device



Notes:

[1] Additional parameters that are used only if **-t** specifies a logical device as a target. See "Using base device parameters" on page 65.

To prepare a device as a boot device, you must specify:

The location *<image>*
of the Linux kernel image on the file system.

A target *<directory>* or *<tape_node>*
zipl installs the boot loader code on the device that contains the specified directory *<directory>* or to the specified tape device *<tape_node>*.

For KVM guests, the target device can be a virtual block device or a VFIO pass-trough DASD. For details, see *KVM Virtual Server Management*, SC34-2752.

Optionally, you can also specify:

A kernel image address *<image_addr>*
to which the kernel image is loaded at IPL time. The default address is 0x10000.

The RAM disk location *<ramdisk>*
of an initial RAM disk image (initrd) on the file system.

A RAM disk image address *<initrd_addr>*
to which the RAM disk image is loaded at IPL time. If you do not specify this parameter, **zipl** investigates the location of other components and calculates a suitable address for you.

Kernel parameters
to be used at IPL time. If you provide multiple parameters, separate them with a blank and enclose them within single quotation marks (') or double quotation marks (").

You can specify parameters *<parameters>* directly on the command line. Instead or in addition, you can specify a location *<parmfile>* of a kernel parameter file on the file system. See "How kernel parameters from different sources are combined" on page 63 for a discussion of how **zipl** combines multiple kernel parameter specifications.

A parameter address *<parm_addr>*
to which the kernel parameters are loaded at IPL time. The default address is 0x1000.

An option -k auto
to install a kdump kernel that can be used as a stand-alone dump tool. You can IPL this kernel in an LPAR or guest virtual machine to create a dump of a previously running operating system instance that has been configured with a reserved memory area for kdump. For Linux, this memory area is reserved with the `crashkernel=` kernel parameter.

Note: For SCSI disks, the accumulated size of the kernel and ramdisk must not exceed 16 MB.

An option -a
to add the kernel image, kernel parameter file, and initial RAM disk to the bootmap file. Use this option when these files are spread across multiple disks to ensure that they are available at IPL time. This option is available on the command line only. Specifying this option significantly increases the size of the bootmap file that is created in the target directory.

The secure boot mode *<mode>*
SCSI IPL disk or NVMe IPL device for LPAR only: Controls the zipl secure boot support. *<mode>* takes the following values:

auto
Uses the secure-boot enabled format if the **zipl** command is issued on a mainframe with secure-boot support. This is the default.

1
Enforces the secure-boot enabled format regardless of mainframe support. Use this option to prepare boot devices for systems other than the one you are working on. Disks with this format cannot be booted on machines z14 or earlier.

0
Enforces the traditional format, that does not support secure boot, regardless of mainframe support. Disks with this format can be booted on all machines but cannot be used for secure boot.

For more information about secure boot, see "Secure boot" on page 102.

See "Parameter overview" on page 71 for a summary of the parameters. This summary includes the long options that you can use on the command line.

zipl configuration file syntax

Figure 18 on page 62 summarizes how you can specify a boot configuration within a zipl configuration file section. Required specifications are shown in bold. See "zipl configuration file structure" on page 75 for a more comprehensive discussion of the configuration file.

```
[<section_name>]
image=<image>,<image_addr>
ramdisk=<ramdisk>,<initrd_addr>
parmfile=<parmfile>,<parm_addr>
parameters=<parameters>
# Next line for devices other than tape only
target=<directory>
# Next line for tape devices only
tape=<tape_node>
# Next line for stand-alone kdump only
kdump=auto
# Next line for secure boot only
secure=<mode>
```

Figure 18. Syntax for preparing a boot device: zipl configuration-file section

Example

The following command identifies the location of the kernel image as `/boot/mnt/image-2`, identifies the location of an initial RAM disk as `/boot/mnt/initrd`, specifies a kernel parameter file `/boot/mnt/parmf-2`, and writes the required boot loader code to `/boot`. At IPL time, the initial RAM disk is to be loaded to address 0x900000, rather than an address that is calculated by **zipl**. Kernel image, initial RAM disk, and the kernel parameter file are to be copied to the bootmap file on the target directory `/boot` rather than being referenced.

```
# zipl -i /boot/mnt/image-2 -r /boot/mnt/initrd,0x900000 -p /boot/mnt/parmf-2 -t /boot -a
```

An equivalent section in a configuration file might look like this example:

```
[boot2]
image=/boot/mnt/image-2
ramdisk=/boot/mnt/initrd,0x900000
paramfile=/boot/mnt/parmf-2
target=/boot
```

There is no configuration file equivalent for option **-a**. To use this option for a boot configuration in a configuration file, it must be specified with the **zipl** command that processes the configuration.

If the configuration file is called `/etc/myxmp.conf`:

```
# zipl -c /etc/myxmp.conf boot2 -a
```

How kernel parameters from different sources are combined

zipl allows for multiple sources of kernel parameters when preparing boot devices.

In command-line mode, you can use two possible sources of kernel parameters. The parameters are processed in the following order:

1. Parameters in the kernel parameter file (specified with the **-p** or **--parmfile** option)
2. Parameters that are specified on the command line (specified with the **-P** or **--parameters** option)

In configuration file mode, the possible sources of kernel parameters depend on where the configuration is specified, in a zipl configuration-file section or in a BLS snippet. The parameters are processed in the following order:

For a zipl configuration-file section

1. Parameters in the kernel parameter file (specified with the **parmfile=** option)
2. Parameters that are specified in the configuration section (specified with the **parameters=** option)
3. Parameters that are specified on the command line (specified with the **-P** or **--parameters** option)

For a BLS snippet

1. Parameters that are specified in the snippet (specified with the **options** option)
2. Parameters that are specified on the command line (specified with the **-P** or **--parameters** option)

Parameters from different sources are concatenated and passed to the kernel in one string. At IPL time, the combined kernel parameter string is loaded to address 0x1000, unless an alternate address is provided.

For more information about the different sources of kernel parameters, see “Including kernel parameters in a boot configuration” on page 25.

Preparing a logical device as a boot device

A *logical device* is a block device that represents one or more real devices.

If your boot directory is on a logical DASD or SCSI device, zipl cannot detect all required information about the underlying real device or devices and needs extra input.

Logical devices can be two DASDs combined into a logical mirror volume. Another examples are a linear mapping of a partition to a real device or a more complex mapping hierarchy. Logical devices are controlled by a device mapper.

Blocks on the logical device must map to blocks on the underlying real device or devices linearly. If two blocks on the logical device are adjacent, they must also be adjacent on the underlying real devices. This requirement excludes mappings such as *striping*.

You always boot from a real device. **zipl** must be able to write to that device, starting at block 0. In a logical device setup, starting at the top of the mapping hierarchy, the first block device that grants access to block 0 (and subsequent blocks) is the *base device*, see Figure 19 on page 64.



Figure 19. Definition of base device

A base device can have the following mappings:

- A mapping to a part of a real device that contains block 0
- A mapping to one complete real device
- A mapping to multiple real devices.

For a mapping to multiple real devices all the real devices must share the device characteristics and contain the same data (for example, a mirror setup). The mapping can also be to parts of the devices if these parts contain block 0. The mapping must not combine multiple devices into one large device.

The **zipl** command needs the device node of the base device and information about the physical characteristics of the underlying real devices. For most logical boot devices, a helper script automatically provides all the required information to **zipl** for you (see "Using a helper script" on page 64).

If you decide not to use the supplied helper script, or want to write your own helper script, you can use parameters to supply the base device information to **zipl**, see "Using base device parameters" on page 65 and "Writing your own helper script" on page 66.

Using a helper script

zipl provides a helper script, `zipl_helper.device-mapper`, that detects the required information and provides it to **zipl** for you.

The helper script is used automatically when you run **zipl** to prepare a boot device. Specify the parameters for the kernel image, parameter file, initial RAM disk, and target as usual. See "Preparing a boot device" on page 60 for details about the parameters.

Assuming an example device for which the location of the kernel image is `/boot/image-5`, the location of an initial RAM disk as `/boot/initrd-5`, a kernel parameter file `/boot/parmf-5`, and which writes the required boot loader code to `/boot` and is a device mapper device, the command then becomes:

```
# zipl -i /boot/image-5 -r /boot/initrd-5 -p /boot/parmf-5 -t /boot
```

The corresponding configuration file section becomes:

```
[boot5]
image=/boot/image-5
ramdisk=/boot/initrd-5
paramfile=/boot/parmf-5
target=/boot
```

Using base device parameters

You can use parameters to supply the base device information to **zipl** directly.

The following command syntax for the base device parameters is used for logical boot devices. It extends the **`zipl`** syntax as shown in "Preparing a boot device" on page 60.



You must specify the following device information:

The device node *<targetbase_node>*
of the base device, either by using the standard device name or in form of the major and minor number, separated by a colon (:).

Example: The device node specification for the device might be `/dev/dm-0` and the equivalent specification with major and minor numbers might be `253:0`.

The device type
of the base device. The following specifications are valid:

LDL
for ECKD type DASD with the Linux disk layout.

CDL
for ECKD type DASD with the compatible disk layout.

FBA
for FBA type DASD.

SCSI
for FCP-attached SCSI disks and for NVMe disks.

LDL and CDL only: The disk geometry *<cylinders>,<heads>,<sectors>*
of the base device in cylinders, heads, and sectors.

The block size *<targetblocksize>*
in bytes per block of the base device.

The offset *<targetoffset>*
in blocks between the start of the physical device and the start of the topmost logical device in the mapping hierarchy.

Figure 20 on page 66 shows how you can specify this information in a configuration file.

```
[<section_name>]
image=<image>,<image_addr>
ramdisk=<ramdisk>,<initrd_addr>
parmfile=<parmfile>,<parm_addr>
parameters=<parameters>
target=<directory>
targetbase=<targetbase_node>
targettype=LDL|CDL|FBA|SCSI
# Next line for target types LDL and CDL only
targetgeometry=<cylinders>,<heads>,<sectors>
targetblocksize=<targetblocksize>
targetoffset=<targetoffset>
```

Figure 20. zipl syntax for preparing a logical device as a boot device - configuration file mode

Example

The example command identifies the location of the kernel image as `/boot/image-5`, identifies the location of an initial RAM disk as `/boot/initrd-5`, specifies a kernel parameter file `/boot/parmf-5`, and writes the required boot loader code to `/boot`.

The command specifies the following information about the base device: the device node is `/dev/dm-3`, the device has the compatible disk layout, there are 6678 cylinders, there are 15 heads, there are 12 sectors, and the topmost logical device in the mapping hierarchy begins with an offset of 24 blocks from the start of the base device.

```
# zipl -i /boot/image-5 -r /boot/initrd-5 -p /boot/parmf-5 -t /boot --targetbase /dev/dm-3 \
# --targettype CDL --targetgeometry 6678,15,12 --targetblocksize=4096 --targetoffset 24
```

Note: Instead of using the continuation sign (\) at the end of the first line, you might want to specify the entire command on a single line.

An equivalent section in a configuration file might look like this example:

```
[boot5]
image=/boot/image-5
ramdisk=/boot/initrd-5
paramfile=/boot/parmf-5
target=/boot
targetbase=/dev/dm-3
targettype=CDL
targetgeometry=6678,15,12
targetblocksize=4096
targetoffset=24
```

Writing your own helper script

You can write your own helper script for device drivers that provide logical devices. The helper script must conform to a set of rules.

- The script must accept the name of the target directory as an argument. From this specification, it must determine a suitable base device. See “Using base device parameters” on page 65.
- The script must write the following base device *<parameter>=<value>* pairs to stdout as ASCII text. Each pair must be written on a separate line.
  - `targetbase=`*<targetbase_node>*
  - `targettype=`*<type>* where type can be LDL, CDL, FBA, or SCSI.

- `targetgeometry=<cylinders>,<heads>,<sectors>` (For LDL and CDL only)
- `targetblocksize=<blocksize>`
- `targetoffset=<offset>`

See "Using base device parameters" on page 65 for the meaning of the base device parameters.

- The script must be named `zipl_helper.<device>` where *<device>* is the device name as specified in `/proc/devices`.
- The script must be in `/lib/s390-tools`.

Preparing a dump device

Use **zipl** with the **-d** (**--dumpto**) command-line option or with the **dumpto=** configuration-file option to prepare a DASD device, SCSI disk, or channel-attached tape dump device.

zipl command line syntax for preparing a dump device



Notes:

[1] For SCSI dump devices only

To prepare a DASD device, SCSI disk, NVMe disk, or channel-attached tape dump device, you must specify:

The device node *<dump_device>*
of the DASD device, SCSI disk partition, NVMe disk partition, or channel-attached tape device to be prepared as a dump device. **zipl** deletes all data on the partition or tape and installs the boot loader code there.

Note:

- If the dump device is an ECKD disk with fixed-block layout (LDL), a dump overwrites the dump utility. You must reinstall the dump utility before you can use the device for another dump.
- If the dump device is a channel-attached tape, SCSI disk, NVMe disk, FBA disk, or ECKD disk with the compatible disk layout (CDL), you do not need to reinstall the dump utility after every dump.
- If the dump device is an NVMe disk and depending on your HMC version, you might have to prepare a partition in namespace 1 to be able to trigger an LPAR dump from the HMC GUI.

Optionally, you can also specify:

An option -n
to suppress confirmation prompts to allow unattended processing (for example, from a script). This option is available on the command line only.

A limit *<size>*
for the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary.

If you limit the dump size below the amount of memory that is used by the system to be dumped, the resulting dump is incomplete.

Note: For SCSI and NVMe dump devices, the "size" option is not available.

SCSI dump tool parameter:

dump_debug=*<level>*
sets the level of debug messages during the dump process. *<level>* is an integer in the range 1 - 6. Use higher numbers for more detailed messages. The default is 2.

DASD device, SCSI disk, NVMe disk, or channel-attached tape dump devices are not formatted with a file system so no target directory can be specified. See *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751 for details about processing these dumps.

See "Parameter overview" on page 71 for a summary of the parameters. The summary includes the long options that you can use on the command line.

Figure 21 on page 68 summarizes how you can specify a DASD device, SCSI disk, or channel-attached tape dump configuration in a configuration file. See "zipl configuration file structure" on page 75 for a more comprehensive discussion of the configuration file.

```
[<section_name>]
dumpto=<dump_device>,<size>
```

Figure 21. zipl syntax for preparing a DASD device, SCSI disk, or channel-attached tape dump device - configuration file mode

DASD example

The following command prepares a DASD partition `/dev/dasdc1` as a dump device and suppresses confirmation prompts that require an operator response:

```
# zipl -d /dev/dasdc1 -n
```

An equivalent section in a configuration file might look like this example:

```
[dumpdasd]
dumpto=/dev/dasdc1
```

There is no configuration file equivalent for option **-n**. To use this option for a DASD or tape dump configuration in a configuration file, it must be specified with the **zipl** command that processes the configuration.

If the configuration file is called `/etc/myxmp.conf`:

```
# zipl -c /etc/myxmp.conf dumpdasd -n
```

SCSI disk example

The following command prepares a SCSI disk partition `/dev/mapper/36005076303ffd40100000000000020c01` as a dump device:

```
# zipl -d /dev/mapper/36005076303ffd40100000000000020c01
```

Note: The trailing 1 in the node name denotes partition 1.

An equivalent section in a configuration file might look like this example:

```
[dumpscsi]
dumpto=/dev/mapper/36005076303ffd40100000000000020c01
```

If the configuration file is called `/etc/myxmp.conf`, the **zipl** command that processes the configuration would be:

```
# zipl -c /etc/myxmp.conf dumpscsi
```

NVMe disk example

The following command prepares an NVMe disk partition `/dev/nvme0n1p1` as a dump device:

```
# zipl -d /dev/nvme0n1p1
```

An equivalent section in a configuration file might look like this example:

```
[dumpnvme]
dumpto=/dev/nvme0n1p1
```

If the configuration file is called `/etc/myxmp.conf`, the **zipl** command that processes the configuration would be:

```
# zipl -c /etc/myxmp.conf dumpnvme
```

Preparing a multi-volume dump on ECKD DASD

Use **zipl** with the **-M** (**--mvdump**) command-line option or with the **mvdump=** configuration-file option to prepare a multi-volume dump on ECKD DASD.

zipl command line syntax for preparing devices for a multi-volume dump

To prepare a set of DASD devices for a multi-volume dump, you must specify:

-M *<dump_device_list>*
: specifies a file that contains the device nodes of the dump partitions, separated by one or more line feed characters (0x0a). **zipl** writes a dump signature to each involved partition and installs the stand-alone multi-volume dump tool on each involved volume. Duplicate partitions are not allowed. A maximum of 32 partitions can be listed. The volumes must be formatted with cdl and use block size 4096.

Optionally, you can also specify:

-f or --force
: to force that no signature checking takes place when dumping. Any data on all involved partitions is overwritten without warning.

-n
: to suppress confirmation prompts to allow unattended processing (for example, from a script). This option is available on the command line only.

<size>
: for the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary.

  If you limit the dump size below the amount of memory that is used by the system to be dumped, the resulting dump is incomplete.

DASD or tape dump devices are not formatted with a file system so no target directory can be specified. See *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751 for details about processing these dumps.

See "Parameter overview" on page 71 for a summary of the parameters. This summary includes the long options that you can use on the command line.

Figure 22 on page 70 summarizes how you can specify a multi-volume DASD dump configuration in a configuration file. See "zipl configuration file structure" on page 75 for a more comprehensive discussion of the configuration file.

```
[<section_name>]
mvdump=<dump_device_list>,<size>
```

Figure 22. zipl syntax for preparing DASD devices for a multi-volume dump - configuration file mode

Example

The following command prepares two DASD partitions `/dev/dasdc1, /dev/dasdd1` for a multi-volume dump and suppresses confirmation prompts that require an operator response:

```
# zipl -M mvdump.conf -n
```

where the `mvdump.conf` file contains the two partitions that are separated by line breaks:

```
/dev/dasdc1
/dev/dasdd1
```

An equivalent section in a configuration file might look like this example:

```
[multi_volume_dump]
mvdump=mvdump.conf
```

There is no configuration file equivalent for option **-n**. To use this option for a multi-volume DASD dump configuration in a configuration file, it must be specified with the **zipl** command that processes the configuration.

If the configuration file is called `/etc/myxmp.conf`:

```
# zipl -c /etc/myxmp.conf multi_volume_dump -n
```

Installing a menu configuration

Use **zipl** with the **-m** (**--menu**) command-line option to install a menu configuration.

To prepare a menu configuration, you need a configuration file that includes at least one menu.

zipl syntax for installing a menu configuration



Notes:

[1] You can change the default configuration file with the ZIPLCONF environment variable.
[2] You can change the default directory with the BOOT environment variable.

Where:

-m or --menu
specifies the menu that defines the menu configuration in the configuration file.

<config_file>
specifies the configuration file where the menu configuration is defined. The default, `/etc/zipl.conf`, can be changed with the ZIPLCONF environment variable.

-b *<bls_dir>*
specifies a directory to be searched for files with BLS snippets.

-a or --add-files
adds the kernel image file, parmfile, and initial RAM disk image to the bootmap files in the respective target directories instead of referencing them. Use this option if the files are spread across disks to ensure that the files are available at IPL time. Specifying this option significantly increases the size of the bootmap file that is created in the target directory.

Example

Using the sample configuration file of Figure 23 on page 78, you could install a menu configuration with:

```
# zipl -m menu1
```

Parameter overview

You might need to know all **`zipl`** options and how to specify them on the command line, in the zipl configuration-file, or in a BLS snippet.

Option	Explanation

Command line:
`-a`
`--add-files`
	Causes kernel image, kernel parameter file, and initial RAM disk to be added to the bootmap file in the target directory rather than being referenced from this file.
	Use this option when these files are spread across multiple disks to ensure that they are available at IPL time. Specifying this option significantly increases the size of the bootmap file that is created in the target directory.

Command line: `-b` *<bls_dir>* `--blsdir=`*<bls_dir>*	Specifies the directory where zipl finds files with BLS snippets. You can change the default directory, `/boot/loader/entries`, with the BOOT environment variable. See “BLS configuration snippets” on page 79.
Command line: `-c` *<config_file>* `--config=`*<config_file>*	Specifies the configuration file. You can change the default configuration file `/etc/zipl.conf` with the environment variable ZIPLCONF.
Command line: *<configuration>*	Specifies a configuration section in a zipl configuration-file or a BLS snippet to be processed. A configuration section in a zipl configuration-file is specified through its section name. A BLS snippet is specified through the value of its `title` option within the snippet.

Option	Explanation

Command line:
`-d <dump_device>[,<size>]`
`--dumpto=<dump_device>[,<size>]`
 	
zipl configuration-file:
`dumpto=<dump_device>[,<size>]`
	Specifies the DASD partition, SCSI disk partition, NVMe disk partition, or tape device to which a dump is to be written after IPL.
	
	The optional size specification limits the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary. If you limit the dump size below the amount of memory that is used by the system to be dumped, the resulting dump is incomplete. If no limit is provided, all of the available physical memory is dumped.
	
	See "Preparing a dump device" on page 67 and *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751 for details.

Command line: `--environment`	Specifies the location of a zipl environment file, see "zipl environment - Variables for the kernel command line" on page 80. The default location is `/etc/ziplenv`.

Command line:
`-h`
`--help`
	Displays help information.

Command line:
`-i <image>[,<image_addr>]`
`--image=<image>[,<image_addr>]`
 	
zipl configuration-file:
`image=<image>[,<image_addr>]`
 	
BLS snippet:
`linux <image>`
	Specifies the location of the Linux kernel image on the file system.
	
	In command-line mode or in a zipl configuration-file section you can optionally specify a memory location after IPL. The default memory address is 0x10000.
	
	See "Preparing a boot device" on page 60 for details.

Command line:
`-k auto`
`--kdump=auto`
 	
zipl configuration-file:
`kdump=auto`
	Installs a kdump kernel that can be used as a stand-alone dump tool. You can IPL this kernel in an LPAR or guest virtual machine to create a dump of a previously running operating system instance that has been configured with a reserved memory area for kdump. For Linux, this memory area is reserved with the `crashkernel=` kernel parameter.
	
	See "Preparing a boot device" on page 60 for details.

Command line:
`-m <menu_name>`
`--menu=<menu_name>`
	Specifies the name of the menu that defines a menu configuration in the configuration file (see "Menu configurations" on page 76).

Option	Explanation

Command line:
`-M <dump_device_list>[,<size>]`
`--mvdump=<dump_device_list>[,<size>]`
 	
zipl configuration-file:
`mvdump=<dump_device_list>[,<size>]`
	Specifies a file with a list of DASD partitions to which a dump is to be written after IPL.
	
	The optional size specification limits the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary. If you limit the dump size below the amount of memory that is used by the system to be dumped, the resulting dump is incomplete. If no limit is provided, all of the available physical memory is dumped.
	
	See "Preparing a multi-volume dump on ECKD DASD" on page 69 and *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751 for details.

Command line:
`-n`
`--noninteractive`
 	
n/a
	Suppresses all confirmation prompts (for example, when preparing a DASD or tape dump device).

Command line:
`-p <parmfile>[,<parm_addr>]`
`--parmfile=<parmfile>[,<parm_addr>]`
 	
zipl configuration-file:
`parmfile=<parmfile>[,<parm_addr>]`
	In a boot configuration, specifies the location of a kernel parameter file.
	
	You can specify multiple sources of kernel parameters. For more information, see "How kernel parameters from different sources are combined" on page 63.
	
	The optional *<parm_addr>* specifies the memory address where the combined kernel parameter list is to be loaded at IPL time.

Command line:
`-P <parameters>`
`--parameters=<parameters>`
 	
zipl configuration-file:
`parameters=<parameters>`
 	
BLS snippet:
`options <parameters>`
	In a boot configuration, specifies kernel parameters.
	
	Individual parameters are single keywords or have the form `key=value`, without spaces. If you provide multiple parameters, separate them with a blank and enclose them within single quotation marks (') or double quotation marks (").
	
	You can specify multiple sources of kernel parameters. For more information, see "How kernel parameters from different sources are combined" on page 63.

Command line:
 	
`-r <ramdisk>[,<initrd_addr>]`
`--ramdisk=<ramdisk>[,<initrd_addr>`
 	
zipl configuration-file:
`ramdisk=<ramdisk>[,<initrd_addr>`
 	
BLS snippet:
`initrd <ramdisk>`
	Specifies the location of the initial RAM disk (initrd) on the file system.
	
	In command-line mode or in a zipl configuration-file section you can optionally specify a memory location after IPL.
	
	If you do not specify a memory address, **zipl** investigates the location of other components and calculates a suitable address for you.

Option	Explanation

`-S <mode>`
`--secure=<mode>`
 	
`secure=auto|0|1`
	In an LPAR boot configuration, controls the format of the boot data that zipl writes to a SCSI IPL disk or NVMe IPL device. You can specify the following values for *<mode>*:
	auto
	Uses the secure-boot enabled format if the zipl command is issued on a mainframe with secure-boot support. This is the default.
	1
	Enforces the secure-boot enabled format regardless of mainframe support. Use this option to prepare boot devices for systems other than the one you are working on. Disks with this format cannot be booted on machines z14 or earlier.
	0
	Enforces the traditional format, that does not support secure boot, regardless of mainframe support. Disks with this format can be booted on all machines but cannot be used for secure boot.
	For more information about secure boot, see "Secure boot" on page 102.

Command line:
`-t <directory>`
`--target=<directory>`
 	
zipl configuration-file:
`target=<directory>`
	Specifies the target directory where **zipl** creates boot-relevant files. The boot loader is installed on the disk that contains the target directory.

Command line:
 	
`--targetbase=<targetbase_node>`
 	
zipl configuration-file:
`targetbase=<targetbase_node>`
	For logical boot devices, specifies the device node of the base device, either by using the standard device name or in form of the major and minor number, separated by a colon (:).
	See "Using base device parameters" on page 65 for details.

Command line:
`--targetblocksize=<targetblocksize>`
 	
zipl configuration-file:
`targetblocksize=<targetblocksize>`
	For logical boot devices, specifies the bytes per block of the base device.
	See "Using base device parameters" on page 65 for details.

Command line:
`--targetgeometry=<cylinders>,<heads>,<sectors>`
 	
zipl configuration-file:
`targetgeometry=<cylinders>,<heads>,<sectors>`
	For logical boot devices that map to ECKD type base devices, specifies the disk geometry of the base device in cylinders, heads, and sectors.
	See "Using base device parameters" on page 65 for details.

Option	Explanation
Command line: `--targetoffset=`*<targetoffset>* zipl configuration-file: `targetoffset=`*<targetoffset>*	For logical boot devices, specifies the offset in blocks between the start of the physical device and the start of the logical device. See "Using base device parameters" on page 65 for details.
Command line: `--targettype=`*<type>* zipl configuration-file: `targettype=`*<type>*	For logical boot devices, specifies the device type of the base device. See "Using base device parameters" on page 65 for details.
Command line: `-T` *<tape_node>* `--tape=`*<tape_node>* zipl configuration-file: `tape=`*<tape_node>*	Specifies the tape device where **`zipl`** installs the boot loader code.

Command line:
`-v`
`--version`
	Prints version information.

Command line:
`-V`
`--verbose`
	Provides more detailed command output.

If you call **`zipl`** in configuration file mode without specifying a configuration file, the default `/etc/zipl.conf` is used. You can change the default configuration file with the environment variable ZIPLCONF.

The default directory for files with BLS snippets is `/boot/loader/entries`. You can change this default with the BOOT environment variable.

zipl configuration file structure

A zipl configuration file comprises a default section and one or more sections with IPL configurations. In addition, there can be sections that define menu configurations.

[defaultboot]
a default section that defines what is to be done if the configuration file is called without a section specification.

[*<configuration>*]
one or more sections that describe IPL configurations.

:*<menu_name>*
optionally, one or more menu sections that describe menu configurations.

A configuration file section consists of a section identifier and one or more option lines. Option lines are valid only as part of a section. Blank lines are permitted, and lines that begin with the number sign (#) are treated as comments and ignored. Option specifications consist of keyword=value pairs. There can but need not be blanks before and after the equal sign (=) of an option specification.

Default section

The default section begins with the `[defaultboot]` section identifier, which is followed by one of the options `default=`, `defaultmenu=`, or `defaultauto`.

default=*<section_name>*
where *<section_name>* is one of the IPL configurations described in the zipl configuration-file. If the zipl configuration-file is called without a section specification, an IPL device is prepared according to this IPL configuration.

defaultmenu=*<menu_name>*
where *<menu_name>* is the name of a menu configuration that is described in the configuration file. If the configuration file is called without a section specification, IPL devices are prepared according to this menu configuration.

Examples

- This default specification points to a boot configuration `boot1` as the default.

```
[defaultboot]
default=boot1
```

- This default specification points to a menu configuration with a menu `menu1` as the default.

```
[defaultboot]
defaultmenu=menu1
```

IPL configurations

An IPL configuration has a section identifier that consists of a section name within square brackets and is followed by one or more option lines.

Each configuration includes one of the following mutually exclusive options that determine the type of IPL configuration:

image=*<image>*
Defines a boot configuration. See "Preparing a boot device" on page 60 for details.

dumpto=*<dump_device>*
Defines a DASD, SCSI, NVMe, or tape dump configuration. See "Preparing a dump device" on page 67 for details.

mvdump=*<dump_device_list>*
Defines a multi-volume DASD dump configuration. See "Preparing a multi-volume dump on ECKD DASD" on page 69 for details.

KVM: For KVM guests, `image=` is the only supported option.

Additional parameters might be required for logical boot devices (see "Preparing a logical device as a boot device" on page 64).

Menu configurations

For DASD and SCSI devices, you can define a menu configuration. A menu configuration has a section identifier that consists of a menu name with a leading colon.

The identifier is followed by one or more lines with references to IPL configurations in the same zipl configuration-file or to BLS snippets. The menu configuration can also include one or more option lines.

target=*<directory>*
specifies a device where a boot loader is installed that handles multiple IPL configurations. For menu configurations, the target options of the referenced IPL configurations are ignored.

<i>=<configuration>
specifies a menu item. A menu includes one and more lines that specify the menu items.

<configuration> is the name of an IPL configuration that is described in the same zipl configuration-file, or it is the title of a BLS snippet. You can specify multiple boot configurations. For SCSI target devices, you can also specify one or more SCSI dump configurations. You cannot include DASD dump configurations as menu items.

<i> is the configuration number. The configuration number sequentially numbers the menu items, beginning with 1 for the first item. When initiating an IPL from a menu configuration, you can specify the configuration number of the menu item you want to use.

default=*<n>*
specifies the configuration number of one of the configurations in the menu to define it as the default configuration. If this option is omitted, the first configuration in the menu is the default configuration.

prompt=*<flag>*
for a DASD target device, determines whether the menu is displayed when an IPL is performed. Menus cannot be displayed for SCSI target devices.

For `prompt=1` the menu is displayed, for `prompt=0` it is suppressed. If this option is omitted, the menu is not displayed. Independent of this parameter, the operator can force a menu to be displayed by specifying `prompt` in place of a configuration number for an IPL configuration to be used.

If the menu of a menu configuration is not displayed, the operator can either specify the configuration number of an IPL configuration or the default configuration is used.

timeout=*<seconds>*
for a DASD target device and a displayed menu, specifies the time in seconds, after which the default configuration is IPLed, if no configuration has been specified by the operator. If this option is omitted or if `0` is specified as the timeout, the menu stays displayed indefinitely on the operator console and no IPL is performed until the operator specifies an IPL configuration.

secure=*<mode>*
In an LPAR boot configuration, controls the format of the boot data that zipl writes to a SCSI IPL device. You can specify the following values for *<mode>*:

auto
Uses the secure-boot enabled format if the zipl command is issued on a mainframe with secure-boot support. This is the default.

1
Enforces the secure-boot enabled format regardless of mainframe support. Use this option to prepare boot devices for systems other than the one you are working on. Disks with this format cannot be booted on machines z14 or earlier.

0
Enforces the traditional format, that does not support secure boot, regardless of mainframe support. Disks with this format can be booted on all machines but cannot be used for secure boot.

For more information about secure boot, see “Secure boot” on page 102.

As for any configuration section, additional parameters might be required for logical boot devices (see “Preparing a logical device as a boot device” on page 64).

Example

Figure 23 on page 78 shows a sample configuration file that defines multiple configuration sections and two menu configurations.

```
[defaultboot]
defaultmenu=menu1

# First boot configuration (DASD)
[boot1]
ramdisk=/boot/initrd
parameters='root=/dev/ram0 ro'
image=/boot/image-1
target=/boot

# Second boot configuration (SCSI)
[boot2]
image=/boot/mnt/image-2
ramdisk=/boot/mnt/initrd,0x900000
parmfile=/boot/mnt/parmf-2
target=/boot

# Third boot configuration (DASD)
[boot3]
image=/boot/mnt/image-3
ramdisk=/boot/mnt/initrd
parmfile=/boot/mnt/parmf-3
target=/boot

# Configuration for dumping to tape
[dumptape]
dumpto=/dev/rtibm0

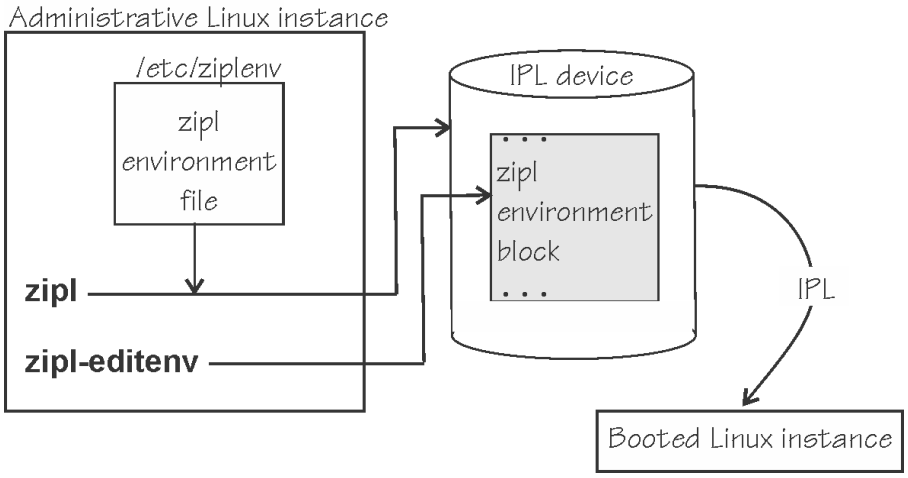
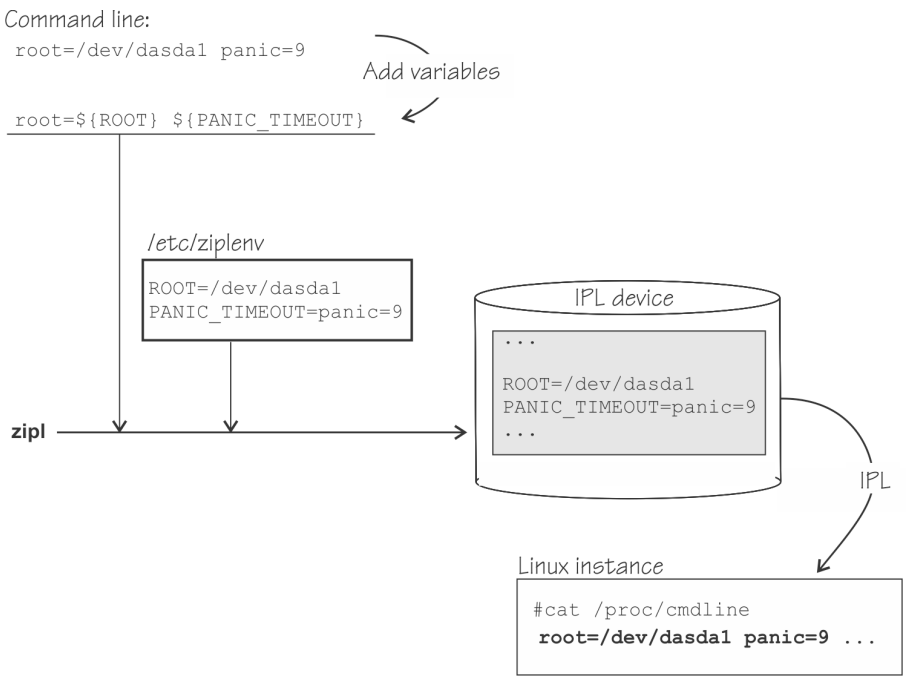
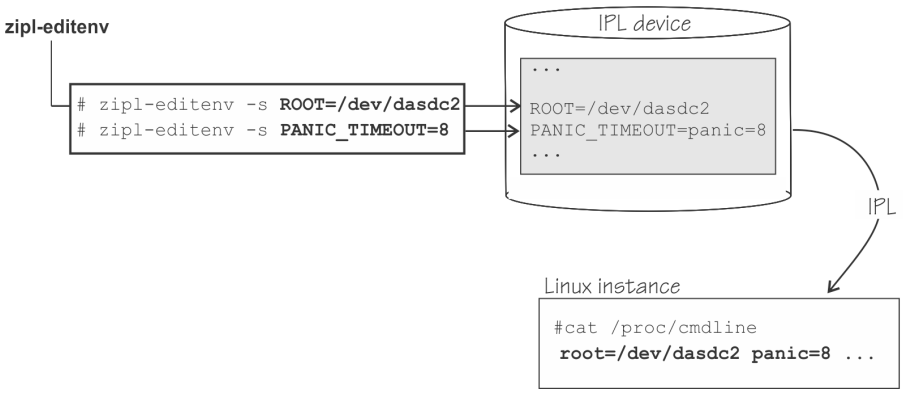
# Configuration for dumping to DASD
[dumpdasd]
dumpto=/dev/dasdc1

# Configuration for multi-volume dumping to DASD
[multi_volume_dump]
mvdump=sample_dump_conf

# Configuration for dumping to SCSI disk
[dumpscsi]
dumpto=/dev/mapper/36005076303ffd40100000000000020c01

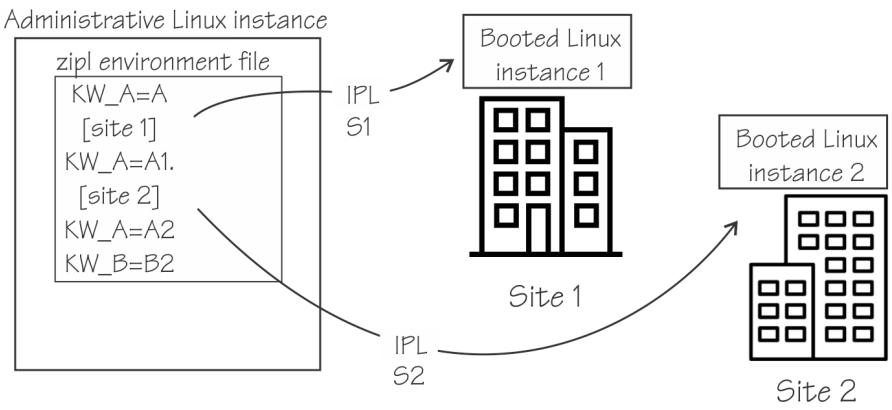
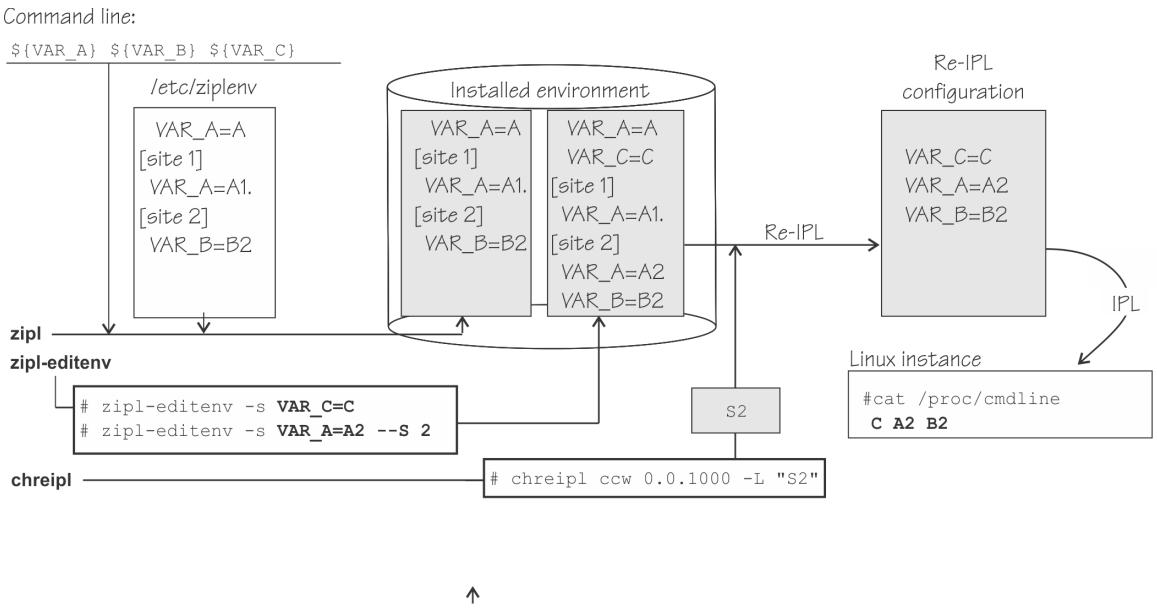
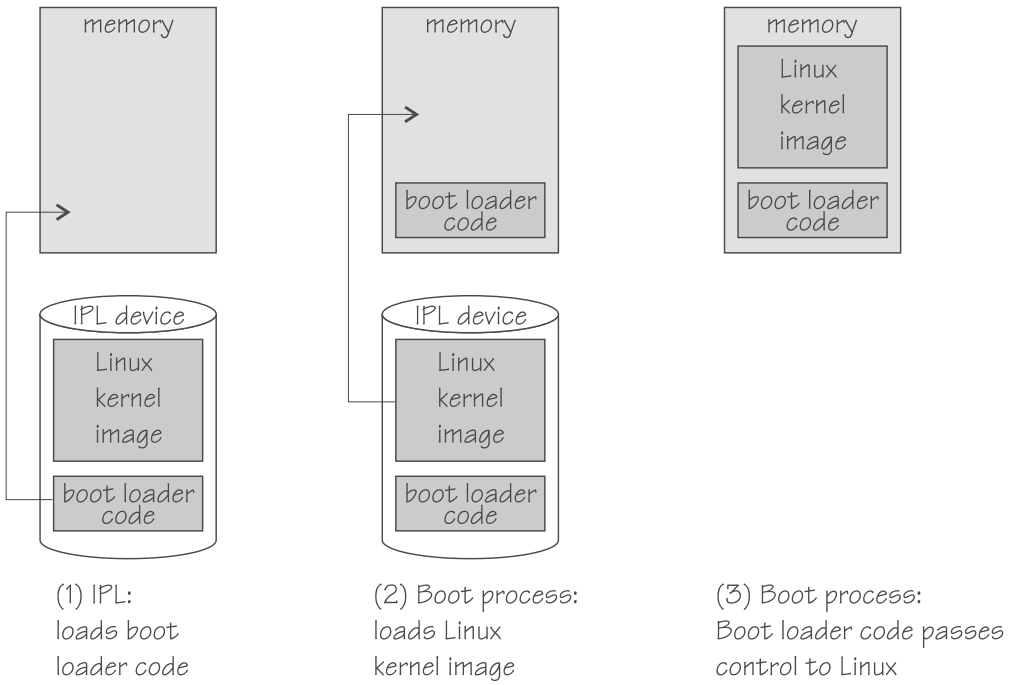
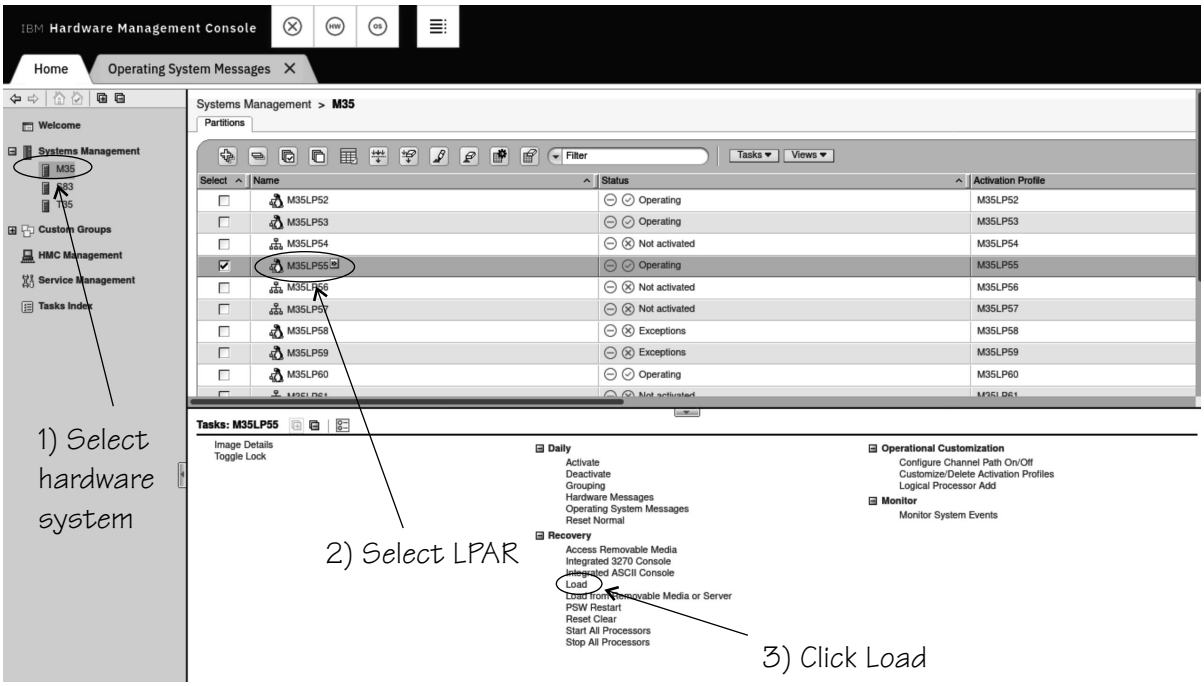
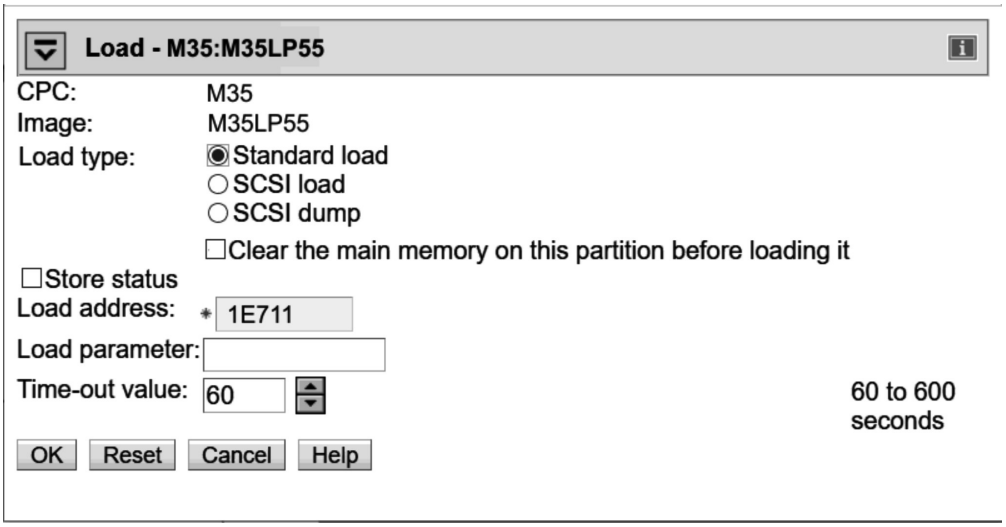
# Configuration for dumping to NVMe
[dumpnvme]
dumpto=/dev/nvme0n1p1

# Menu containing the SCSI boot and SCSI dump configurations
:menu1
1=dumpscsi
2=boot2
target=/boot
default=2

# Menu containing two DASD boot configurations
:menu2
1=boot1
2=boot3
target=/boot
default=1
prompt=1
timeout=30
```

Figure 23. Sample /etc/zipl.conf file

The following commands assume that the configuration file of the sample is the default configuration file.

- Call **zipl** to use the default configuration file settings:

```
# zipl
```

  Result: zipl reads the default option from the `[defaultboot]` section and selects the `:menu1` section. It then installs a menu configuration with a boot configuration and a SCSI dump configuration.
- Call **zipl** to install a menu configuration (see also “Installing a menu configuration” on page 70):

```
# zipl -m menu2
```

Result: **zipl** selects the `:menu2` section. It then installs a menu configuration with two DASD boot configurations. “Example for a DASD menu configuration on z/VM” on page 111 and “Example for a DASD menu configuration (LPAR)” on page 98 illustrate what this menu looks like when it is displayed.

- Call **zipl** to install a boot loader for boot configuration `[boot2]`:

```
# zipl boot2
```

Result: **zipl** selects the `[boot2]` section. It then installs a boot loader that loads copies of `/boot/mnt/image-2`, `/boot/mnt/initrd`, and `/boot/mnt/parmf-2`.

- Call **zipl** to prepare a tape that can be IPLed for a tape dump:

```
# zipl dumptape
```

Result: **zipl** selects the `[dumptape]` section and prepares a dump tape on `/dev/rtibm0`.

- Call **zipl** to prepare a DASD dump device:

```
# zipl dumpdasd -n
```

Result: **zipl** selects the `[dumpdasd]` section and prepares the dump device `/dev/dasdc1`. Confirmation prompts that require an operator response are suppressed.

- Call **zipl** to prepare a SCSI dump device:

```
# mount /dev/sda1 /boot
# mount /dev/sda2 /dumps
# mkdir /dumps/mydumps
# zipl dumpscsi
# umount /dev/sda1
# umount /dev/sda2
```

Result: **zipl** selects the `[dumpscsi]` section and prepares the dump device `/dev/sda1`. The associated dump file is created uncompressed in directory `/mydumps` on the dump partition. If space is required, the lowest-numbered dump file in the directory is deleted.

BLS configuration snippets

Using Boot Loader Specification (BLS) snippets, you can add boot configurations to zipl without editing existing configuration files.

BLS snippets are provided as configuration files in a directory that is shared across all installed operating system instances. You add a boot configuration to zipl by adding a file with a BLS snippet to this directory, `/boot/loader/entries` by default.

Files that contain BLS snippets can have any name, but must have the file extension `.conf`. To avoid naming conflicts and to provide a hint about the content, a common naming convention includes the value of `/etc/machine-id`, the kernel version, and an operating system identifier in the name. For example, one such file might be `/boot/loader/entries/36851bb476df410181cc114154a8a524-4.18.0-80.el8.s390x.conf`.

Red Hat Enterprise Linux creates a file with a BLS snippet for you.

BLS options

Table 10 on page 80 shows the subset of BLS options that are relevant to Linux on IBM Z.

Table 10. BLS parameters and zipl equivalents		
Option	**Description**	**zipl configuration file equivalent**
`title`	A meaningful identifier for the IPL configuration. BLS configuration file `title` specifications together with their zipl configuration file equivalents must be unique within the scope of a **`zipl`** command call. The title must be the first specification within a BLS configuration file.	Section name as specified within square brackets ([ ])
`version`	Specifies a version in human readable format. For example, use the output of the **`uname -r`** command. This item is optional.	none
`linux`	Path to a Linux kernel.	image=
`initrd`	Path to an initial RAM disk. This item is optional.	ramdisk=
`options`	Kernel parameters. This item is optional.	parameters=

Snippet syntax

Lines start with an option keyword, followed by a blank, followed by a value. The first line must specify the `title` option. The configuration file can include empty lines and comment lines. Comment lines start with a number sign (#).

BLS snippet example

```
title  Linux 5.14 kernel
version 5.14.0-127.el9.s390x
linux /boot/vmlinuz-5.14.0-127.el9.s390x
initrd /boot/initramfs-5.14.0-127.el9.s390x.img
options root=/dev/disk/by-path/ccw-0.0.6500-part1 rd.dasd=0.0.6500 cio_ignore=all,!condev ...
```

Complementing BLS snippets through a zipl configuration file

You must use a zipl configuration file to complement the specifications in a BLS snippet with the `target=` parameter. Use the default section of the zipl configuration file to set `target=`.

Depending on your distribution, zipl might be installed with a default configuration file at `/etc/zipl.conf`, with a content similar to the following example:

```
[defaultboot]
defaultauto
prompt=1
timeout=5
target=/boot
secure=auto
```

Optionally, you can specify the `secure=` option.

zipl environment - Variables for the kernel command line

zipl prepares an IPL device by installing boot data and a boot record that points to this data. The boot data includes kernel parameter lines.

The straightforward way to change any installed parameter line is to rerun zipl. However, you can avoid rerunning zipl. For this, define variable parts of your parameter line (such as numerical values of timeouts)

with zipl environment variables and define those variables in a special boot data component, called a zipl environment block.

Once the zipl environment block is installed along with other zipl components, you don't need to rerun zipl to change the variable parts: All you have to do is redefine the variables and update only the installed environment block.

Hence, a zipl environment block contains specifications for resolving variables in the kernel command line. These specifications apply to all menu entries that you create and install with zipl.

The installed zipl environment block is interpreted at boot time. zipl creates the zipl environment block from a zipl environment file on the administrative Linux instance, from which you run zipl. See "Creating variables for the kernel command line" on page 81.

Using the zipl environment feature, you can:

- Modify a zipl environment block without rerunning zipl. For example, see "Modifying a zipl environment block with zipl-editenv" on page 84.
- Define common options for the kernel-command line across multiple boot menu entries, see "Specifying common variables across multiple boot menu entries" on page 85.
- Add placeholder variables for future use, see "Specifying variables for future use" on page 86.
- Define a name space for each failover sites in a zipl environment block, thus providing flexible boot configurations across sites, see "Site-aware zipl environment" on page 87.

Figure 24 on page 81 shows the relationship between the zipl environment file and zipl environment block.



Figure 24. zipl environment block and file

Use the **zipl-editenv** command to modify the zipl environment block directly.

For information about the **zipl-editenv** command, see "zipl-editenv - Edit the zipl environment block" on page 764.

Creating variables for the kernel command line

You can modify the kernel command line, substituting its parts with variables.

About this task

Any expression of the form ${*<keyword>*} in the kernel command line is replaced during boot with the value associated with *<keyword>*, as defined by the installed zipl environment block. If no value associated with <keyword> is found, then the expression is removed from the command line.

The initial mapping between keywords and values is established when the boot record is installed with zipl, see "zipl environment file syntax" on page 83. You can change the mapping by using the **zipl-**

editenv command, which modifies the content of an installed zipl environment block, see “Modifying a zipl environment block with zipl-editenv” on page 84.

In a kernel command line, identify parts that you might want to be a variable. The kernel command line can be a combined string from multiple sources, see “How kernel parameters from different sources are combined” on page 63. Some of these sources support variables:

- The zipl configuration file (`/etc/zipl.conf`)
- Kernel parameter file
- BLS snippets (`boot/loader/entries/…`)
- The interactive DASD boot menu

Example

This example shows how to replace variable parts in the kernel command line with variables. The original installation is unaffected. The example shows a snippet of the kernel parameters, other parameters can be included, for example parameters that are automatically appended by an earlier stage of the boot process.

1. Assume that a parameter specification is as follows:

   ```
   root=/dev/dasda1 panic=9
   ```

2. Assume you might want to boot with another root partition and different values of panic time-out. Then, you would replace "`/dev/dasda1`" and "`panic=9`" with variables. Assume that you choose the keywords ROOT and PANIC_TIMEOUT for them, respectively.

   Replace the parameters in the original command line with variables as follows:

   ```
   root=${ROOT} ${PANIC_TIMEOUT}
   ```

   You can replace a whole parameter as with ${PANIC_TIMEOUT} or just the parameter value as with ${ROOT}. Now you must give the variables values. Use a zipl environment file to do this.

3. Use your favorite text editor to set up a zipl environment file. For details about the file syntax, see “zipl environment file syntax” on page 83.

   Assume you create a zipl environment file `/etc/ziplenv`. Now use the keywords you chose before, ROOT and PANIC_TIMEOUT, to set values. For example:

   ```
   #cat /etc/ziplenv
   ROOT=/dev/dasda1
   PANIC_TIMEOUT=panic=9
   ```

4. Prepare a block device for IPL with **zipl**. Run **zipl**.
5. Reboot the system with the prepared boot configuration, log in, and display the current command line:

   ```
   # cat /proc/cmdline
   root=/dev/dasda1 panic=9 ...
   ```

   Figure 25 on page 83 illustrates the process of replacing specifications in the command line with variables.



Figure 25. Variables on the command line

Results

During the IPL process, the variables ${ROOT} and ${PANIC_TIMEOUT} resolved to `/dev/dasda1` and `panic=9` respectively, according to the keyword definitions in the zipl environment block. This ensures that the original installation works as before, but you can now boot with another root partition and different values of panic timeout by changing the zipl environment block. For an example of how to change the zipl environment block, see "Modifying a zipl environment block with zipl-editenv" on page 84.

zipl environment file syntax

The zipl environment file defines the initial mapping between keywords and values established at boot record installation time.

The default location of the file is `/etc/ziplenv`. To use a file at a different location, use the **zipl** command option `--environment`.

Each line of this file defines the mapping on a single keyword-value pair. On each line, the keyword is the sequence of characters that precedes the first equal sign (=). The sequence of characters after that first equal sign (excluding the new-line character) is identified as the value. Values can be empty, but keywords must consist of at least of one character.

The maximum number of keyword-value pairs for one boot partition is 512.

You can modify a zipl environment file with any text editor. An example file might look as follows:

```
...
ROOT=/dev/dasda1
CRASH=256M
PANIC_TIMEOUT=panic=8
PANIC_TIMEOUT=panic=9
RESERVED=
...
```

The keywords must satisfy the following requirements:

- Consist of at least one character of uppercase letters A - Z, digits 0 - 9, and the "_" (underscore).
- Must not begin with a digit.

Lines beginning with "#" are ignored, as are lines without a keyword.

If lines contain identical keywords, the last line overrides preceding ones. For example, in the example file from before, of the two entries for PANIC_TIMEOUT, the entry `PANIC_TIMEOUT=panic=9` would be used.

Creating a boot record with a missing, or empty, zipl environment file results in an empty zipl environment block in the boot data. At boot time, all variables that cannot be resolved are removed from the command line.

If the zipl environment file defines more than 512 keyword-value pairs, or if the environment defined by that file exceeds one file-system block, **zipl** fails to import the file.

Modifying a zipl environment block with zipl-editenv

Use the **zipl-editenv** command to modify the installed zipl environment block.

About this task

Assume you have the same zipl environment file as before, with the ROOT and PANIC_TIMEOUT keywords defined:

```
#cat /etc/ziplenv
ROOT=/dev/dasda1
PANIC_TIMEOUT=panic=9
```

For details about the file syntax, see “zipl environment file syntax” on page 83.

Further, assume that you ran zipl, and the zipl environment block is created. Now you would like to use another root partition and another value for the panic time-out.

Procedure

1. Optional: Display the current zipl environment block by using the **zipl-editenv** command:

   ```
   # zipl-editenv --list
   ROOT=/dev/dasda1
   PANIC_TIMEOUT=panic=9
   ```

   If no option `-t` is specified, **zipl-editenv** assumes that the environment was installed in the `/boot` directory. To specify a different directory, use the `-t` option.
2. Use the **zipl-editenv** command to change the values for ROOT and PANIC_TIMEOUT.
   For example, to set the root partition to `/dev/dasdc2` and the panic time-out to 8, issue the following commands:

   ```
   # zipl-editenv -s ROOT=/dev/dasdc2
   # zipl-editenv -s PANIC_TIMEOUT=panic=8
   ```

   To check that everything is correct, display the modified zipl environment block:

   ```
   # zipl-editenv --list
   ROOT=/dev/dasdc2
   PANIC_TIMEOUT=panic=8
   ```
3. Reboot the system, log in, and display the current command line:

   ```
   # cat /proc/cmdline
   root=/dev/dasdc2 panic=8 ...
   ```

   The root partition and panic time-out were set to the new values.

   The process for modifying the environment block with **zipl-editenv** is illustrated in Figure 26 on page 85



Figure 26. Changing values for keywords that replace specifications in the kernel command line.

Results

You can repeatedly modify the zipl environment block to IPL the Linux instance with different kernel command lines without rerunning zipl.

What to do next

You can define keyword-value pairs for common parameters for the kernel-command line across multiple boot menu entries, see "Specifying common variables across multiple boot menu entries" on page 85.

You can also equip the zipl environment file with keyword-value pairs for future use, see "Specifying variables for future use" on page 86.

Specifying common variables across multiple boot menu entries

The zipl environment feature is useful in the case of multiple boot menu entries.

Procedure

1. Create a zipl environment file that holds only the keyword-value pairs for common variables. For example:

```
# cat /etc/ziplenv
COMMON=nosmt
```

2. Introduce the `${COMMON}` variable to the kernel parameters in the boot menu configurations.

```
[defaultboot]
defaultauto
target=/boot

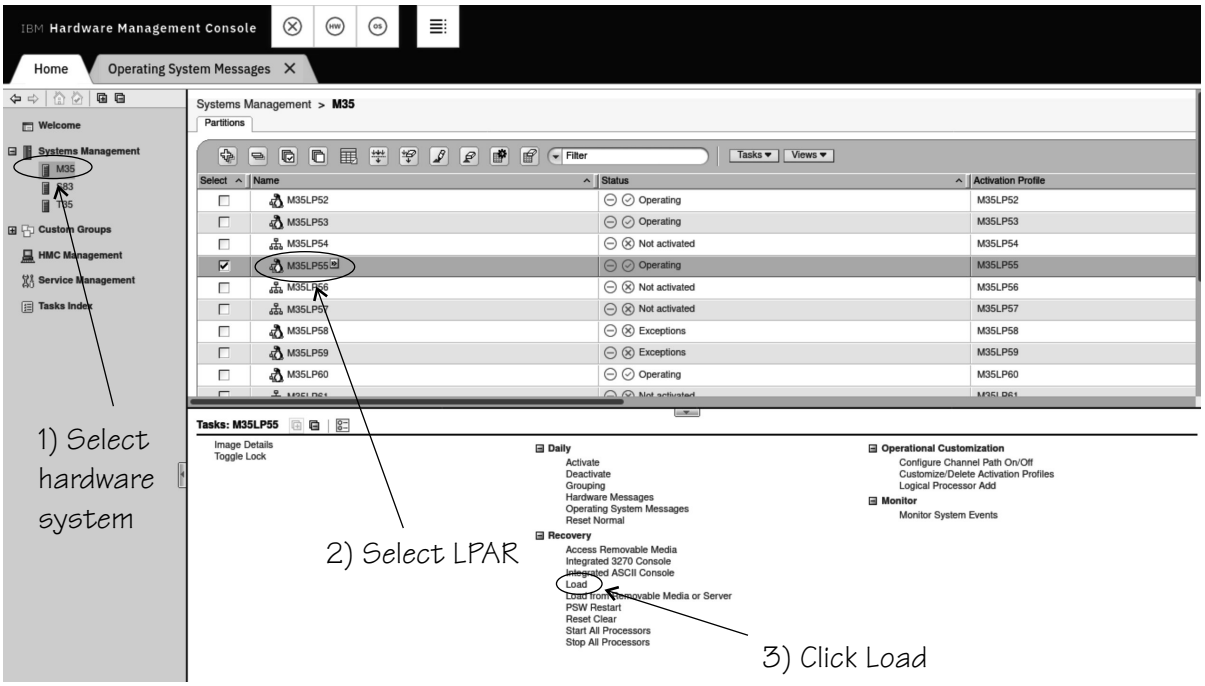
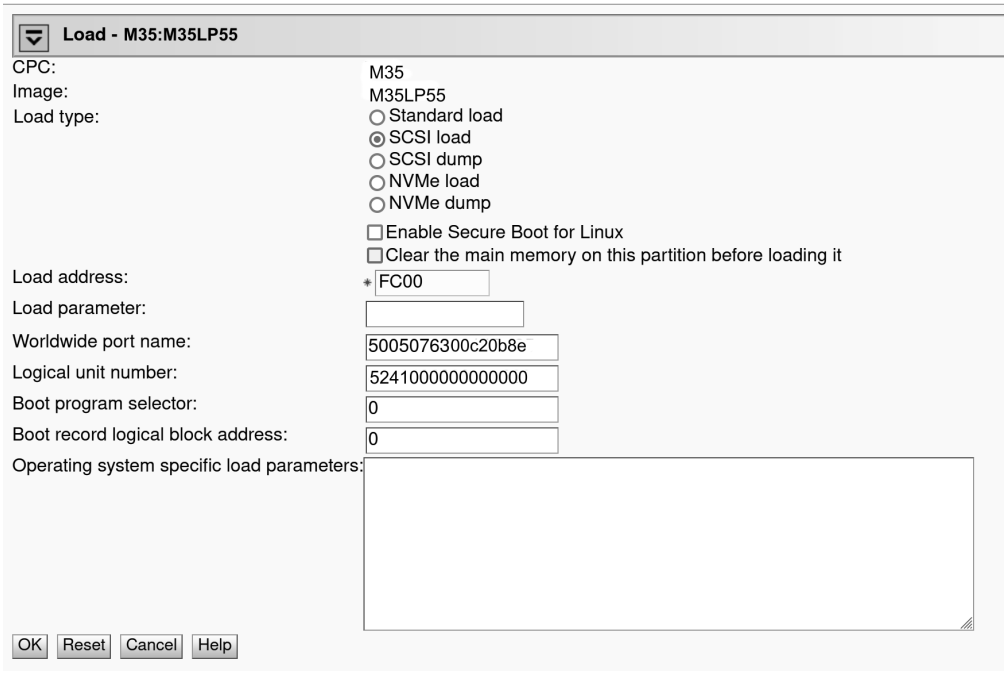
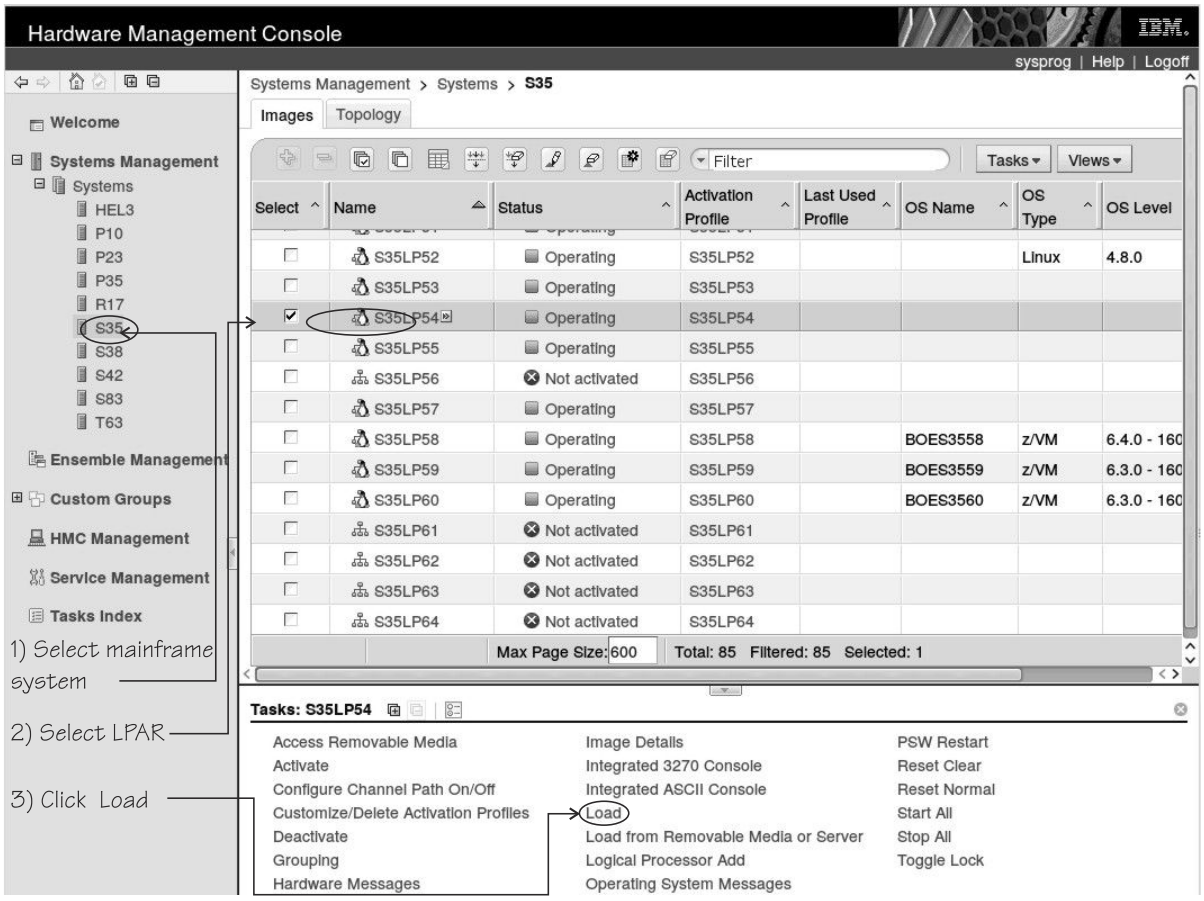
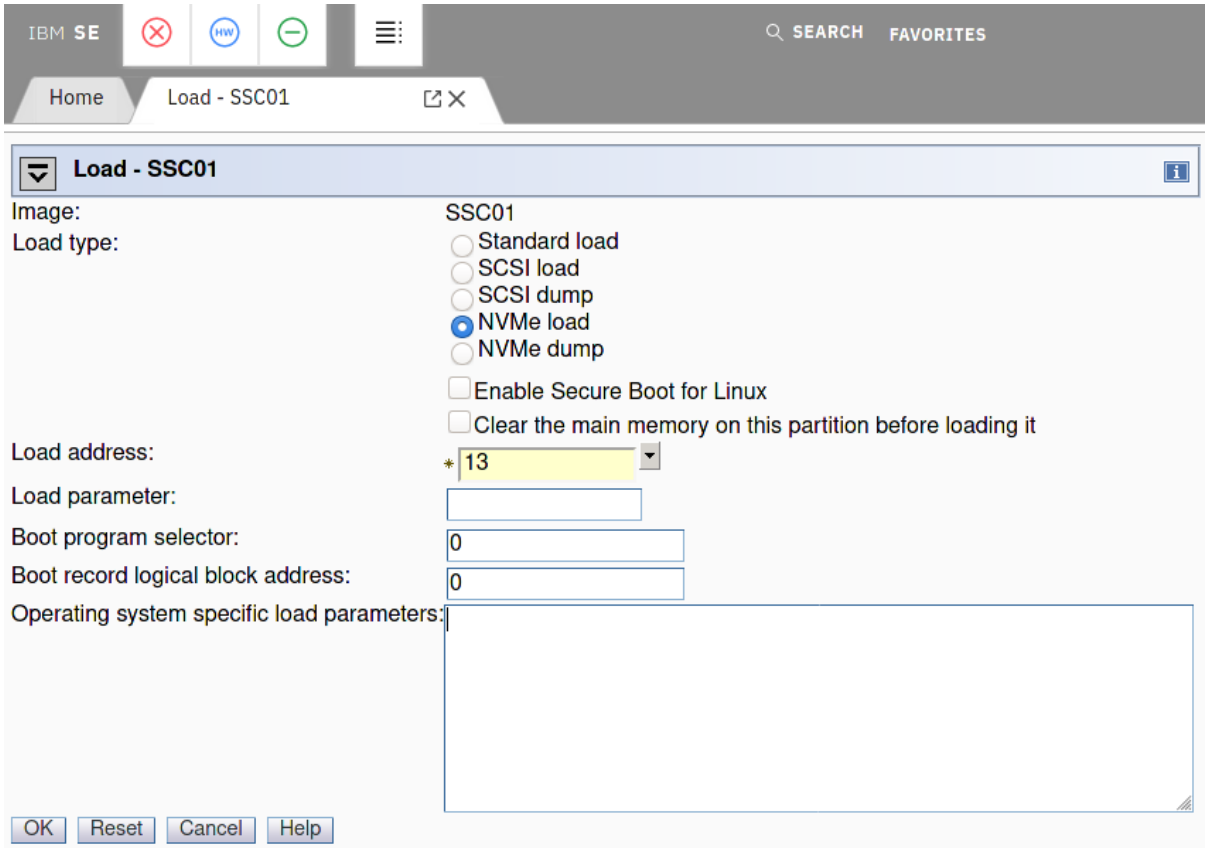
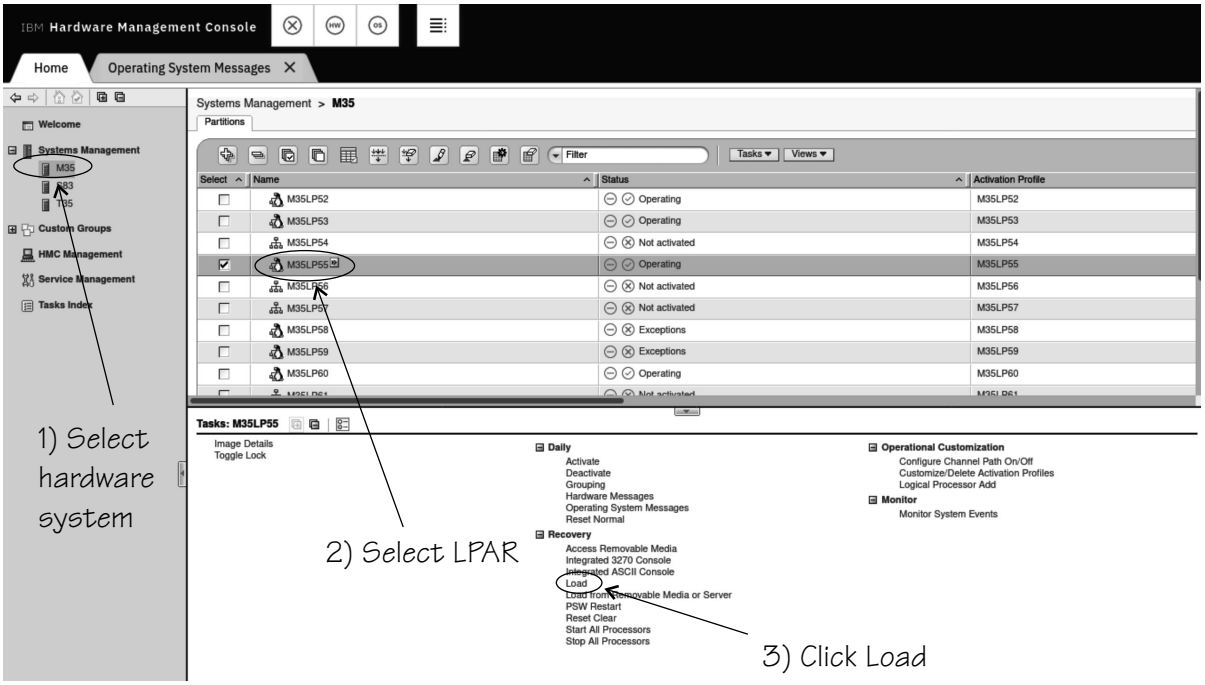
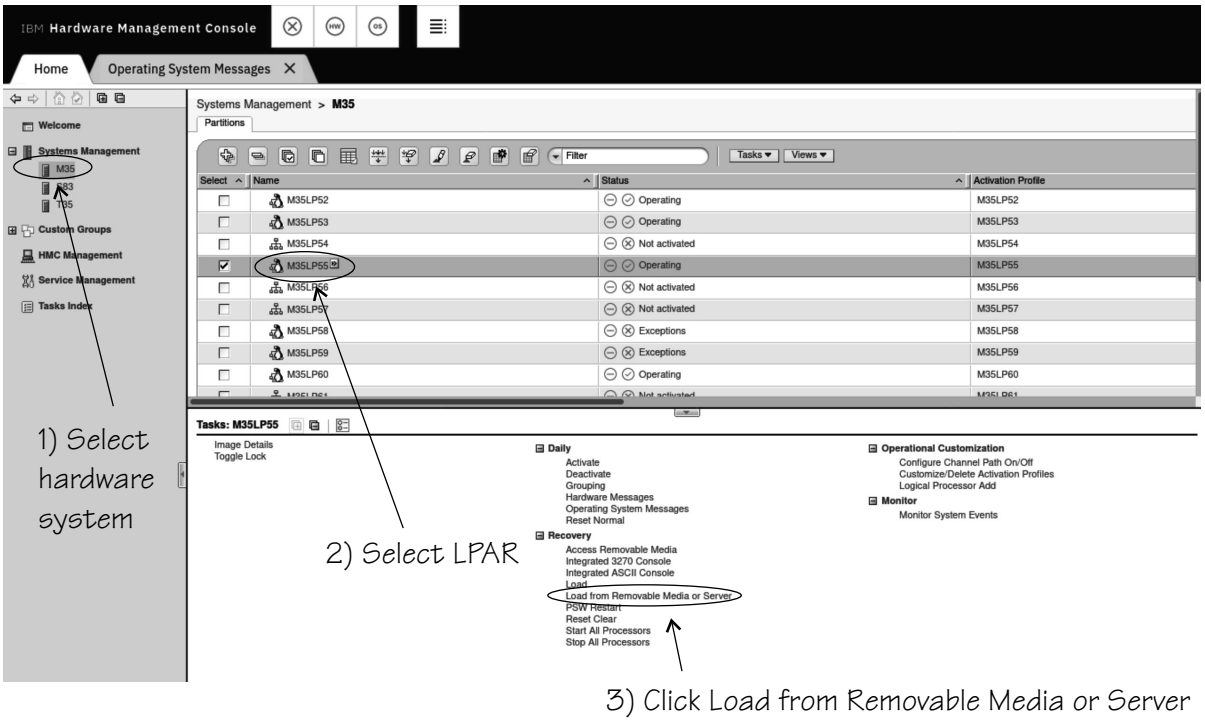
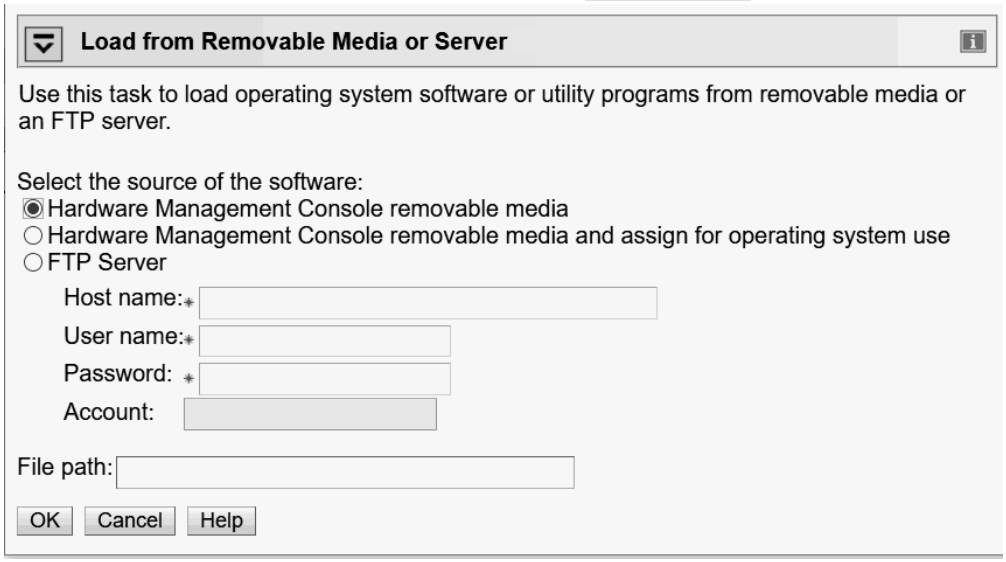
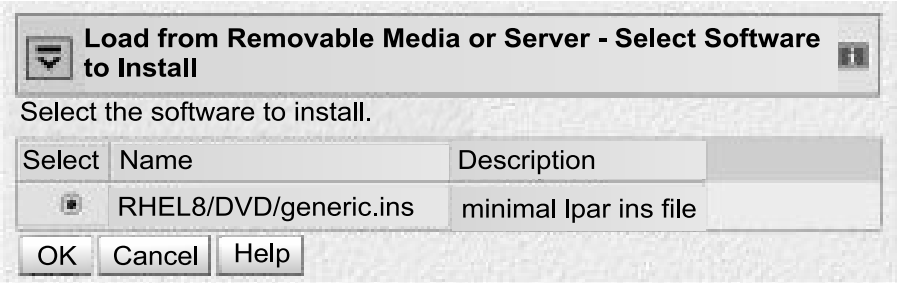
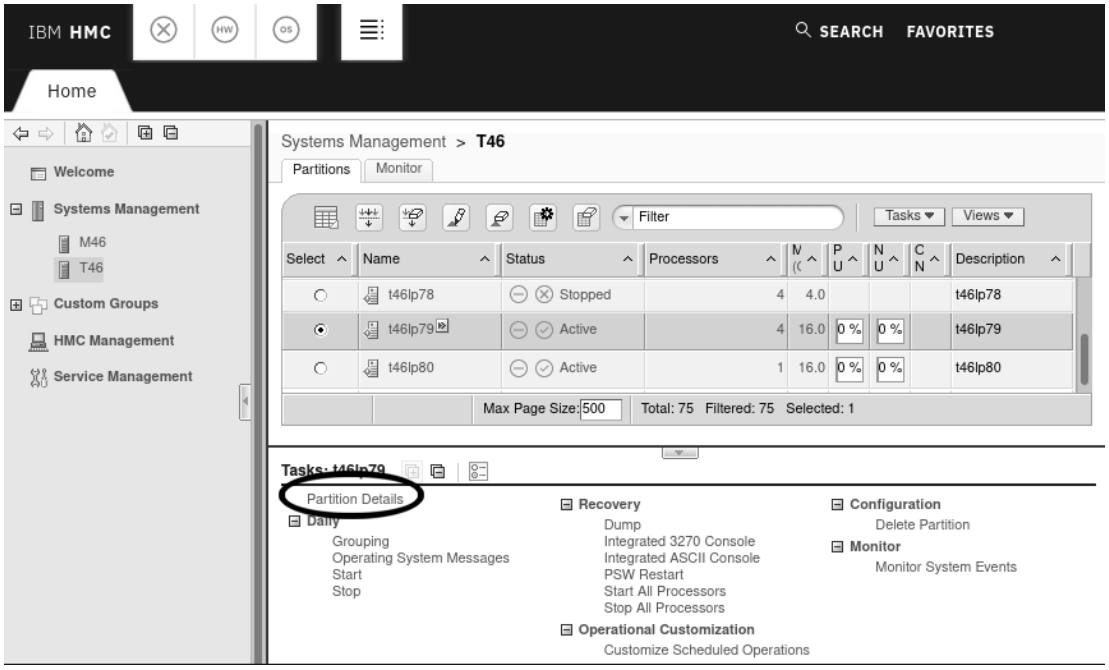
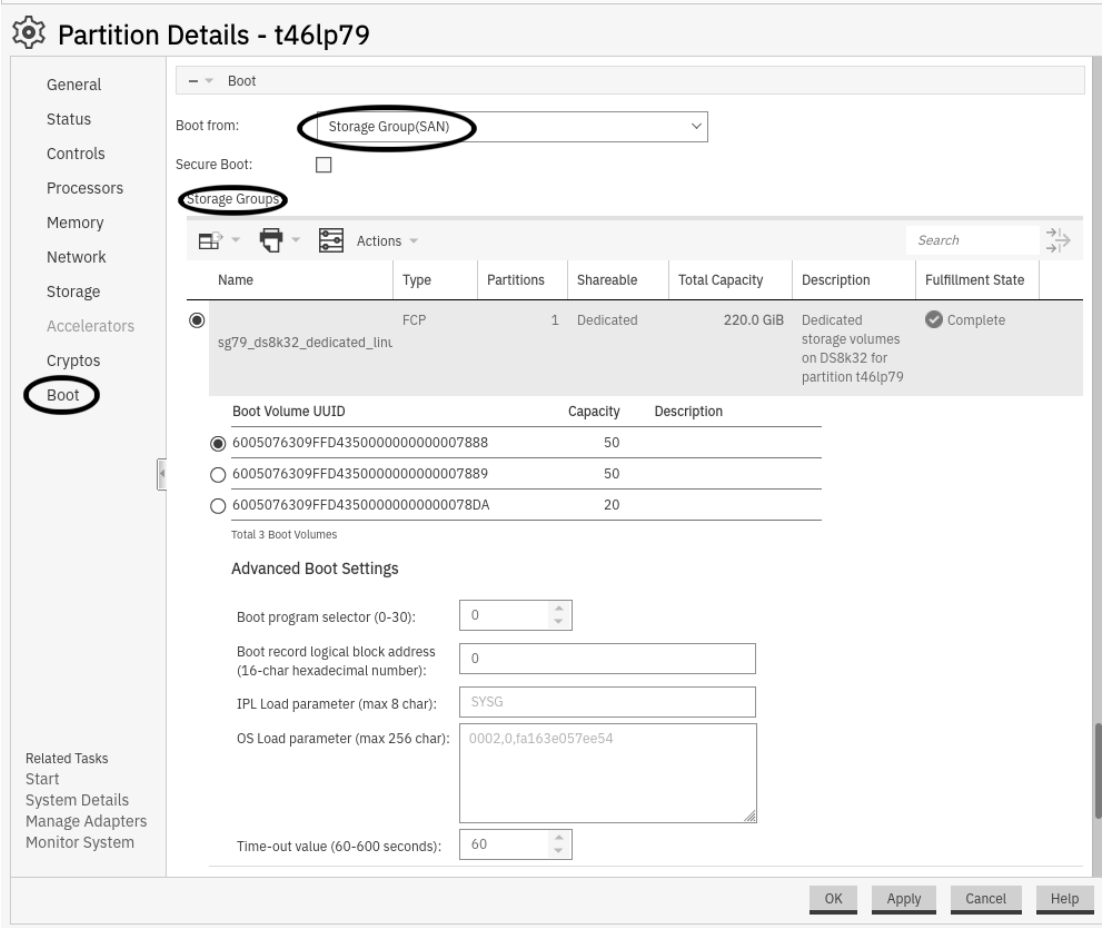
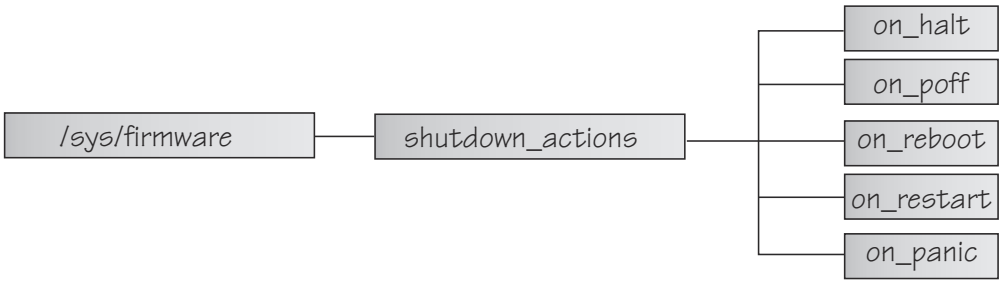
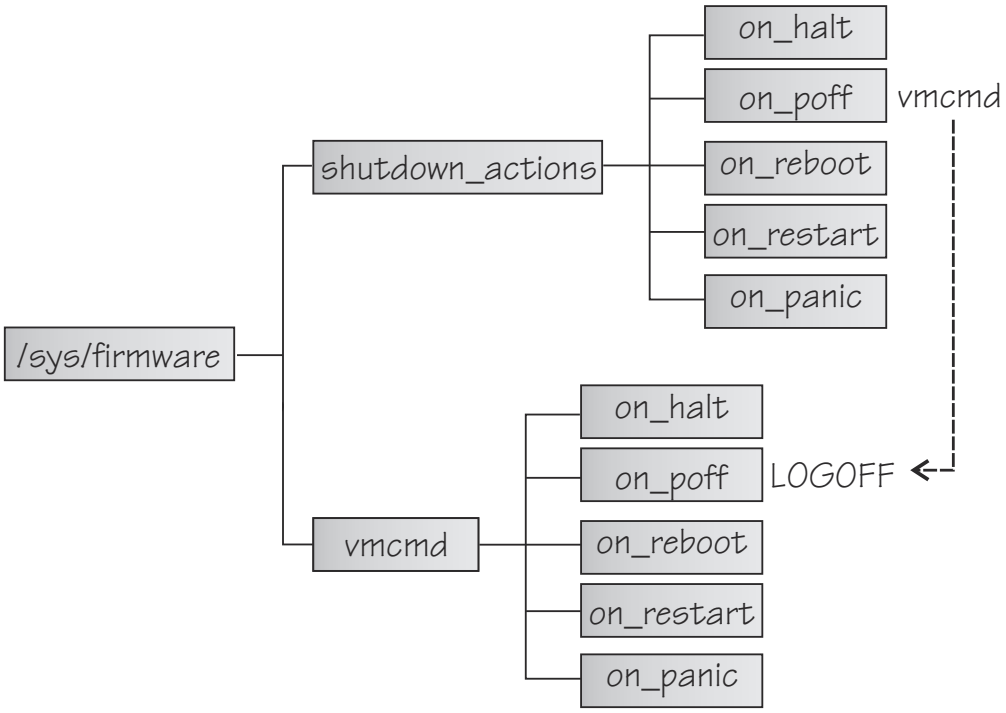
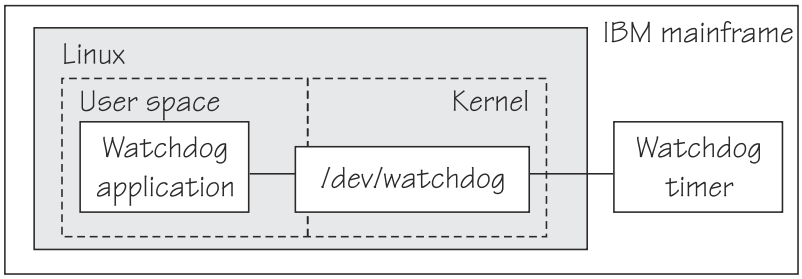
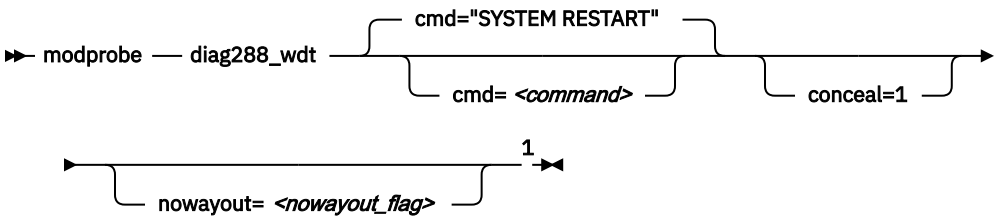
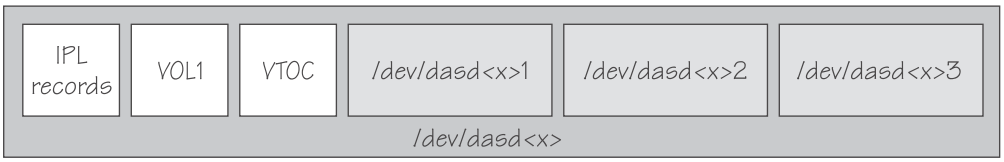
# First boot configuration (DASD)
[boot1]
ramdisk=/boot/initrd
parameters='root=/dev/ram0 ro ${COMMON}'
image=/boot/image-1
target=/boot

# Second boot configuration (DASD)
[boot2]
image=/boot/mnt/image-3
ramdisk=/boot/mnt/initrd
parmfile=/boot/mnt/parmf-3
target=/boot

...
```

For the boot2 configuration section, the `parmfile` option specifies a kernel parameter file that contains kernel parameters, including the `${COMMON}` variable, for example:

```
# cat /boot/mnt/parmf-3
....
CRASH=256M
PANIC_TIMEOUT=panic=8
${COMMON}
```

3. Call **zipl** to install multiple boot configurations that include the ${COMMON} variable.
   For example, to make zipl install both boot configuration boot1 and boot2:

```
# zipl
```

   At boot time, the ${COMMON} variable is replaced by `nosmt`, as specified in the installed zipl environment block by the keyword `COMMON`.

Results

The zipl environment block now defines the `COMMON` keyword:

```
# zipl-editenv --list
...
COMMON=nosmt
```

IPL any boot configuration that uses a command line with the ${COMMON} variable, and display the actual command line:

```
# cat /proc/cmdline
... nosmt ...
```

To not have nosmt set, first ensure that the environment file does not define the COMMON variable, or defines it as an empty string:

```
# cat /etc/ziplenv
COMMON=
```

Then install the boot record.

However, if you do not want to re-install the boot record, you can either set the COMMON variable to the empty string in the already installed zipl environment block:

```
# zipl-editenv -s COMMON=
# zipl-editenv -l
COMMON=
```

Or remove it from the zipl environment block:

```
# zipl-editenv -u COMMON
# zipl-editenv -l
```

Specifying variables for future use

You can extend a zipl environment file to include keywords that are not used yet, but can be in the future.

About this task

You can add keywords for future use by setting them to empty strings. Using such reserved keywords helps you avoid boot record re-installation in the future, when you want to add more parameters to the kernel command line.

Procedure

1. Assume the same command line as before:

```
root=/dev/dasda1 panic=9
```

To add a variable that can be used in the future, add one or more variables to the command line, for example `${MYVARIABLE_1} ${MYVARIABLE_2}`:

```
root=/dev/dasda1 panic=9 ${MYVARIABLE_1} ${MYVARIABLE_2}
```

Ensure that all variables are separated by blanks from other variables.

2. Use a zipl environment file to set values for the keywords.

   You now need to define keyword-value pairs for these variables in a zipl environment file. Keyword-value pairs set to the empty string resolve to the empty string until you define values for them:

```
# cat /etc/ziplenv
root=/dev/dasda1
panic=9
MYVARIABLE_1=
MYVARIABLE_2=
```

3. Run **zipl** to install a boot configuration.
4. Reboot with the prepared boot configuration, and log in.

   Display the command line that was used for the currently running Linux instance. You notice that the original command line is unchanged:

```
# cat /proc/cmdline
root=/dev/dasda1 panic=9 ...
```

Results

The original installation works as before, but you can now use **zipl-editenv** to assign a value in the zipl environment block for a specific IPL device. For example:

```
# zipl-editenv --set MYVARIABLE_1=console=ttyS1
# zipl-editenv --list
root=/dev/dasda1
panic=9
MYVARIABLE_1=console=ttyS1
MYVARIABLE_2=
```

The value for a reserved keyword must be the complete kernel parameter specification. For many kernel parameters this is a *<parameter>=<parameter_value>* pair, for example `panic=9`.

After rebooting, you can see that the new value was applied to the installation as it shows up in the command line:

```
# cat /proc/cmdline
root=/dev/dasda1 panic=9 console=ttyS1 ...
```

Site-aware zipl environment

A site-aware zipl environment supports failover setups accross multiple sites. You can define a sections in a zipl environment file for each site of your failover environment, thus providing flexible boot configurations across sites, see Figure 27 on page 88.

Each section can have site-specific keyword-value pairs defined that replace kernel parameters at boot time. For details on how to create sections, see “Defining sections in the zipl environment file” on page 88.

You use the load parameter specification to select a site-specifc section when setting up re-IPL with **chreipl** or booting with the IPL command. On the HMC, the field is **Load parameter**. For details, see “Activating site-specific kernel parameters” on page 89.

A scenario using a Linux instance running as a z/VM guest that is booted from a DASD is given in "Example: Using a site-aware zipl environment" on page 89.



Figure 27. Site-aware zipl environment

Defining sections in the zipl environment file

You can define a section for each of your failover sites in a zipl environment file.

About this task

To create sections for use with a failover solution, edit the environment file, by default located at `/etc/ziplenv`, to contain a section for each failover site. Then install the boot record by running the zipl command. The installation procedure maps the sections to name spaces in the environment block.

Sections must start with a title of the form `[site <x>]`, where

- *<x>* is a numeral in the range 0 - 9.
- The "site" keyword is not case sensitive.
- The title must be enclosed in square brackets.

All lines between consecutive titles form one section. Sections with identical titles are considered one, compound section. The final section extends to the end of the file.

Kernel parameters that are defined before the first section are considered common to all sections.

Tip: Specify any common kernel parameters first, followed by one or more sections.

Procedure

As an example, consider adding two sites, site 1 and site 2, to the following zipl environment file:

```
ROOT=/dev/dasda1
CRASH=256M
PANIC_TIMEOUT=panic=8
PANIC_TIMEOUT=panic=9
RESERVED=
...
```

Assume you want to use the different PANIC_TIMEOUT values for different sites. Add section titles for the sites:

```
ROOT=/dev/dasda1
CRASH=256M
[site 1]
PANIC_TIMEOUT=panic=8
[site 2]
PANIC_TIMEOUT=panic=9
RESERVED=
...
```

Now the ROOT= and the CRASH= keyword-value pairs are common to all sites. PANIC_TIMEOUT=panic=8 applies to site 1 only, and PANIC_TIMEOUT=panic=9 and RESERVED= apply to site 2.

What to do next

You can now:

- Modify the installed zipl environment block, see "Modifying a zipl environment block with zipl-editenv" on page 84.
- Activate the site-specific kernel parameters, see "Activating site-specific kernel parameters" on page 89.

Activating site-specific kernel parameters

Use loadparm to activate site-specific kernel parameters.

About this task

You can specify loadparm either with the **chreipl** command, or using the z/VM IPL command.

Procedure

- Specify the loadparm parameter `-L` when using **chreipl**.
  For example, to set up a reboot to use DASD 0.0.1000, and kernel parameters for site 2, issue:

```
# chreipl ccw 0.0.1000 -L "S2"
Re-IPL type: ccw
Device: 0.0.1000
Loadparm: "S2"
Bootparms: ""
clear: 0
```

  Specify the loadparm value with a capital S. For more information about the **chreipl** command, see "chreipl - Modify the re-IPL configuration" on page 580.
- Specify the loadparm parameter `-L` when using the z/VM IPL command.
  For example, to boot your Linux image under z/VM using DASD 0.0.384c, boot menu 1, and kernel parameters from site 2, on the z/VM console issue:

```
#cp i 384c loadparm 1S2
```

  For KVM guests, you can specify the loadparm parameter through the disks `<boot>` element, for example, `<boot order="1" loadparm="1S2"/>`.

Example: Using a site-aware zipl environment

This example demonstrates how to set up a flexible boot configuration for the next boot, using variables in a site-aware zipl environment.

About this task

This example, illustrated in Figure 28 on page 90, demonstrates how to use a site-aware zipl environment with a Linux instance running as a guest of z/VM and that is booted from a DASD.



Figure 28. zipl site-aware scenario

Procedure

1. Choose a kernel image, an initial RAM disk, and a kernel command line, that contains zipl environment variables.
2. In the environment file, define sections for each failover site and values for the variables.
3. Run zipl to create a boot configuration.
4. As usual, you can use **zipl-editenv** to modify the environment on the boot volume. You can also target a specific section with `--S`.
5. Activate the site-specific values with loadparm, either by preparing a reboot with **chreipl**, or by specifying loadparm with a site specification at when initiating the boot process.
6. Boot the Linux instance.

Example

Assume that you are on z/VM and your boot device is a DASD with a bus ID 0.0.1000. You have prepared the boot components kernel image, initial RAM disk, and kernel command line, for the next boot.

1. Assume that you want to create three variables in the kernel command line and use them to define different values for different sites. For example, define the following variables in your kernel command line:

   ```
   ${VAR_A} ${VAR_B} ${VAR_C}
   ```

2. Create a zipl environment file, `/etc/ziplenv`, with two sites, site 1, and site 2, as follows:

   ```
   VAR_A=A
   [site 1]
   VAR_A=A1
   [site 2]
   VAR_B=B2
   ```

   For details about the syntax for sites, see "Defining sections in the zipl environment file" on page 88.

   This environment file assigns multiple values of keyword VAR_A. First, the value A is assigned in the common space, before any site definition. Second, the value A1 is assigned for site 1.

3. Run **zipl** to create a boot configuration.
4. Modify the installed zipl environment block.

Check the installed environment with the **zipl-editenv** command:

```
# zipl-editenv --list
Common variables:
  VAR_A=A
Site 1:
  VAR_A=A1
Site 2:
  VAR_B=B2
```

Define, for example, one more value for the keyword VAR_A for site 2, and a value C for the new keyword VAR_C in the common space:

```
# zipl-editenv --set VAR_C=C
# zipl-editenv --set VAR_A=A2 --site 2
# zipl-editenv --list
Common variables:
  VAR_A=A
  VAR_C=C
Site 1:
  VAR_A=A1
Site 2:
  VAR_A=A2
  VAR_B=B2
```

For details about the **zipl-editenv** command, see "zipl-editenv - Edit the zipl environment block" on page 764.

5. Use loadparm to activate the keywords for the desired site, for example site 2. To do this, use **chreipl** with the loadparm option:

```
# chreipl ccw 0.0.1000 -L "S2"
Re-IPL type: ccw
Device:      0.0.1000
Loadparm:    "S2"
Bootparms:   ""
clear: 0
```

For details about loadparm and how to specify menu configurations and sites, see "Booting as a z/VM guest from a DASD" on page 110.

6. Boot the Linux instance. If you used **chreipl** to set up the boot, in z/VM, issue:

```
# cp ipl 1000
```

Alternatively, if you did not use **chreipl**, you can specify the loadparm option with the **ipl** command:

```
# cp ipl 1000 loadparm S2
```

What to do next

Log in to the Linux instance and issue `cat /proc/cmdline` to read the command line.

You should see the following values:

```
C A2 B2
```

Value C was obtained from the common area. Values A2 and B2 were obtained from the site 2 section in the environment file.

Chapter 7. Booting Linux

The options and requirements you have for booting Linux depend on your platform, LPAR, z/VM, or KVM, and on your boot medium.

For details about defining a Linux virtual machine, see *z/VM: Getting Started with Linux on System z®*, SC24-6287, the chapter about creating your first Linux virtual machine.

For details about setting up a KVM virtual server, see *KVM Virtual Server Management*, SC34-2752.

IPL and booting

On IBM Z, you usually start booting Linux by performing an Initial Program Load (IPL) from an IPL device.

A traditional IPL device contains all data that is required to start an IBM Z operating system or a stand-alone program. For Linux this includes a kernel image, possibly an initial RAM disk and kernel parameters, and a boot loader.

For SCSI IPL disks , NVMe devices, and generally for IPL of a KVM guest, the boot loader code is supplied by the hypervisor and not required on the IPL device.

Figure 29 on page 93 summarizes the main steps of the boot process for a traditional IPL device.



Figure 29. IPL and boot process

Use the **zipl** tool to prepare DASD, SCSI, NVMe, and tape devices as IPL devices for booting Linux or for dumping. For more information about **zipl**, see Chapter 6, "Initial program loader for IBM Z - zipl," on page 57.

LPAR

If your Linux instance is to run in an LPAR, you can initiate the IPL from the Support Element (SE) through a load action against an IPL device. Alternatively, you can initiate an IPL by copying the Linux kernel to the mainframe memory (see "Loading Linux from removable media or from an FTP server" on page 104).

You can use secure boot if you IPL from a SCSI or NVMe device. For more information, see "Secure boot" on page 102.

KVM

For Linux on IBM Z as a KVM guest, an IPL is initiated by starting a virtual server on the KVM hypervisor.

The hypervisor first assigns resources to the virtual hardware, then it loads `s390-ccw.img` into the memory of the new virtual hardware. For KVM guests, `s390-ccw.img` takes the role of the boot loader. If needed, `s390-ccw.img` loads `s390-netboot.img` to retrieve boot data over the network.

LPAR and z/VM

An IPL can also start a dump process. See *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751 for more information about dumps.

You can find the newest version of this book at ibm.com/docs/en/linux-on-systems?topic=linuxone-distributions

Control point and boot medium

The control point from where you can start the IPL depends on your hypervisor environment.

LPAR
: For Linux in LPAR mode, the control point is the mainframe's Support Element (SE) or an attached Hardware Management Console (HMC).

z/VM
: For Linux on z/VM, the control program (CP) of the hosting z/VM system.

KVM
: For Linux on KVM, the control point is the KVM host.

The media that can be used as boot devices also depend on where Linux is to run. Table 11 on page 94 provides an overview of the possibilities:

Table 11. Boot media

Medium	LPAR	z/VM guest	KVM guest
DASD	✓	✓	✓
tape	✓	✓	
SCSI	✓	✓	
NVMe	✓		
CD-ROM/DVD/FTP	✓		
z/VM reader		✓	
virtio block device			✓
virtio SCSI device			✓

In the table:

- As of z14, a SCSI boot device is an FC-attached disk. Support for an FC-attached CD-ROM or DVD drive as a boot device is available on IBM Z hardware prior to z14.
- CD-ROM/DVD/FTP can be the CD-ROM or DVD drive of the SE or HMC, or it can be a remote FTP server.
- A virtio block device can be backed by an ISO image in the KVM host file system or by any IPL device that was prepared with **`zipl`**.

Typically, booting from removable media applies to initial installation of Linux. Booting from DASD or SCSI disk devices usually applies to previously installed Linux instances.

Menu configurations

In Red Hat Enterprise Linux 9.1, you can use **zipl** to prepare a DASD or SCSI boot disk with a menu configuration.

A boot device with a menu configuration can hold the code for multiple boot configurations. For SCSI and NVMe disks, the menu can also include one or more system dumpers.

Each boot and dump configuration in a menu is associated with a configuration number. At IPL time, you can specify a configuration number to select the configuration to be used.

For menu configurations on DASD, you can display a menu with the configuration numbers (see "Example for a DASD menu configuration on z/VM" on page 111 and "Example for a DASD menu configuration (LPAR)" on page 98). For menu configurations on SCSI disks, you need to know the configuration numbers without being able to display the menus.

See "Menu configurations" on page 76 for information about defining menu configurations.

Boot data

To boot Linux, you generally need a kernel image, boot loader code, kernel parameters, and an initial RAM disk image.

For sequential I/O boot devices, z/VM reader and tape, the order in which this data is provided is significant. For random access devices, there is no required order.

On Red Hat Enterprise Linux 9.1, kernel images are installed into the `/boot` directory and are named `vmlinuz-<version>.s390x`. For information about where to find the images and how to start an installation, see the Red Hat documentation website https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

Boot loader code

Red Hat Enterprise Linux 9.1 kernel images are compiled to contain boot loader code for IPL from z/VM reader devices.

If you want to boot a kernel image from a device that does not correspond to the included boot loader code, you can provide alternate boot loader code separate from the kernel image.

Use **zipl** to prepare boot devices with separate DASD, SCSI, or tape boot loader code. You can then boot from DASD, SCSI, or tape regardless of the boot loader code in the kernel image.

Kernel parameters

The kernel parameters are in form of an ASCII text string of up to 895 characters. If the boot device is tape or the z/VM reader, the string can also be encoded in EBCDIC.

Individual kernel parameters are single keywords or keyword/value pairs of the form keyword=*<value>* with no blank. Blanks are used to separate consecutive parameters.

If you use the **zipl** command to prepare your boot device, you can provide kernel parameters on the command line, in a parameter file, and in a **zipl** configuration file.

See Chapter 4, "Kernel and module parameters," on page 25, Chapter 6, "Initial program loader for IBM Z - zipl," on page 57, or the **zipl** and `zipl.conf` man pages for more details.

Initial RAM disk image

An initial RAM disk holds files, programs, or modules that are not included in the kernel image but are required for booting.

Red Hat Enterprise Linux 9.1 provides a RAM disk in `/boot` and named `initramfs-<kernel version>.s390x.img`. When a RAM disk is installed or modified, you must call **zipl** to update the boot record.

Rebuilding the initial RAM disk image

Configuration changes might apply to components that are required in the boot process before the root file system is mounted. For Red Hat Enterprise Linux, such components and their configuration are provided through an initial RAM disk.

Procedure

Perform these steps to make configuration changes for components in the initrd take effect:

1. Issue **`dracut -f`** to update the initial RAM disk of your target kernel.
2. Issue **`zipl`** to rewrite the zipl boot record.

Booting Linux in LPAR mode

You can boot Linux in LPAR mode from a Hardware Management Console (HMC) or Support Element (SE).

About this task

The following description refers to an HMC, but the same steps also apply to an SE.

Booting from DASD

Use the SE or HMC to boot Linux in LPAR mode from a DASD boot device.

Before you begin

You need a boot device that is prepared with **`zipl`** (see “Preparing a boot device” on page 60).

Procedure

Perform these steps to boot from a DASD boot device:

1. In the navigation pane of the HMC, expand **Systems Management** and select the hardware system that you want to work with. A table of LPARs is displayed on the **Partitions** tab in the content area.
2. Select the LPAR where you want to boot Linux.
3. In the **Tasks** area, expand **Recovery** and click **Load** (see Figure 30 on page 97).



Figure 30. Load task on the HMC

4. Select the load type **Standard load** (see Figure 31 on page 97).

   Select the **Clear the main memory on this partition before loading it** check box only if you must clear memory. Memory clearing can considerably prolong the IPL procedure.



Figure 31. Load panel for booting from DASD

5. Enter the device number of the DASD boot device in the **Load address** field.

   To IPL from a subchannel set other than 0, specify five digits: The subchannel set ID followed by the device number, for example 1E711.

6. If the boot configuration is part of a **`zipl`** created menu configuration, enter the configuration number that identifies your DASD boot configuration within the menu in the **Load parameter** field.

   Configuration number `0` specifies the default configuration. Depending on the menu configuration, omitting this option might display the menu or select the default configuration. Specifying "prompt" instead of a configuration number forces the menu to be displayed.

   When the menu is displayed, you can specify additional kernel parameters (see “Example for a DASD menu configuration (LPAR)” on page 98). These additional kernel parameters are appended to the

parameters you might have provided in a parameter file. The combined parameter string must not exceed 895 bytes.

See “Menu configurations” on page 76 for more details about menu configurations.

7. Click **OK** to start the boot process.

Results

Check the output on the preferred console (see “Console kernel parameter syntax” on page 42) to monitor the boot progress.

Example for a DASD menu configuration (LPAR)

This example illustrates how menu2 in the sample configuration file in Figure 23 on page 78 is displayed on the HMC or SE:

```
zIPL interactive boot menu

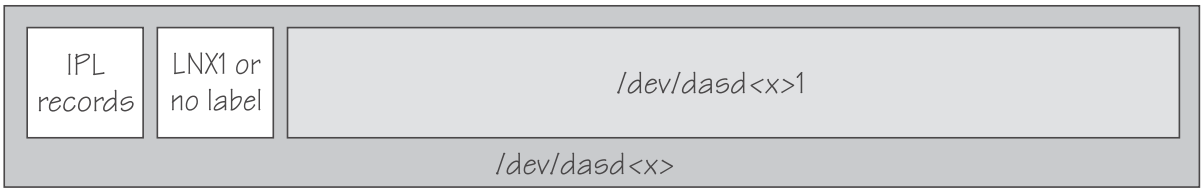
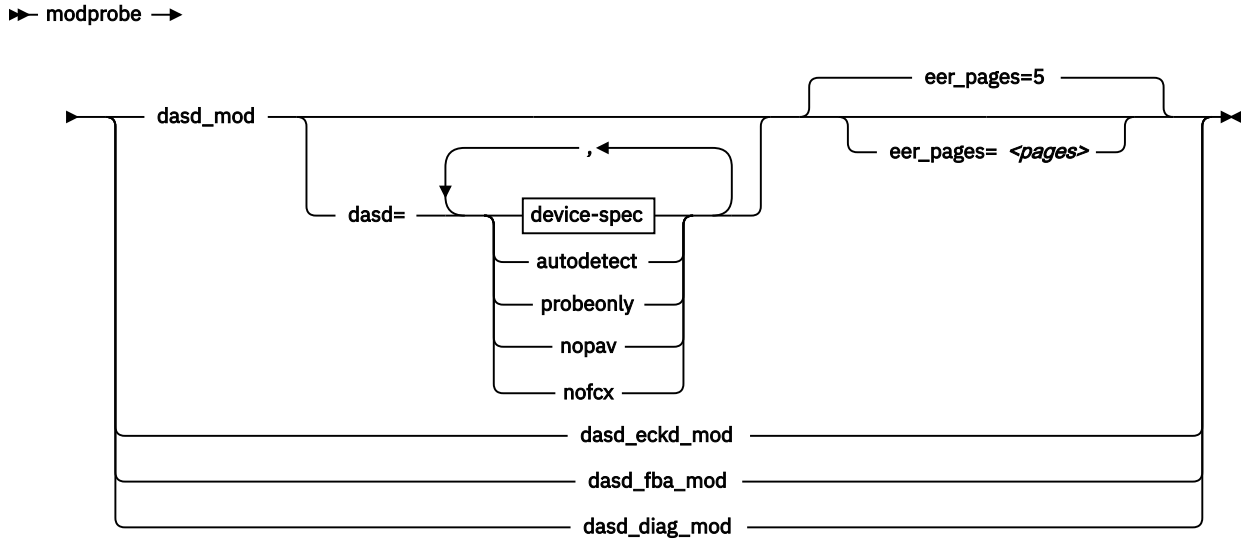
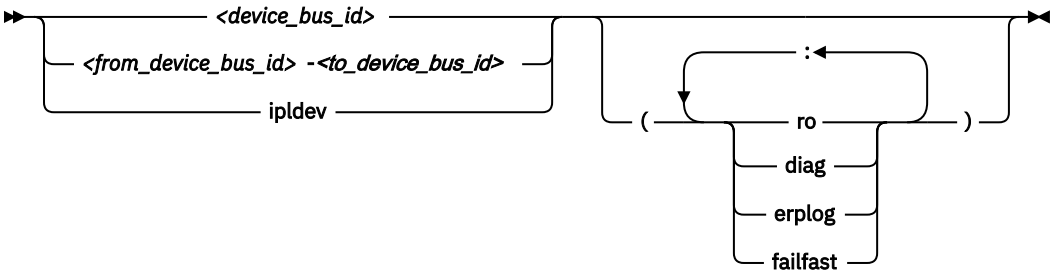
0. default (boot1)

1. boot1
2. boot3

Please choose (default will boot in 30 seconds):
```

You choose a configuration by specifying the configuration number. For example, to boot configuration boot3, issue:

```
# 2
```

You can also specify additional kernel parameters by appending them to this command. For example:

```
# 2 maxcpus=1
```

Booting from SCSI

Use the SE or HMC to boot Linux in LPAR from a SCSI boot device.

Before you begin

You need a boot device that is prepared with **`zipl`** (see “Preparing a boot device” on page 60).

For information about boot devices, see Table 11 on page 94.

Procedure

Perform these steps to boot from a SCSI boot device:

1. In the navigation pane of the HMC, expand **Systems Management** and select the hardware system that you want to work with. A table of LPARs is displayed on the **Partitions** tab in the content area.
2. Select the LPAR where you want to boot Linux.
3. In the **Tasks** area, expand **Recovery** and click **Load** (see Figure 32 on page 99).



Figure 32. Load task on the HMC

4. Select load type **SCSI load** (see Figure 33 on page 99).



Figure 33. Load panel with SCSI feature enabled - for booting from a SCSI device

Select the **Clear the main memory on this partition before loading it** check box only if you must clear memory. Memory clearing can considerably prolong the IPL procedure.

5. Enter the device number of the FCP channel through which the SCSI device is accessed in the **Load address** field.
6. Enter the WWPN of the SCSI device in the **World wide port name** field.
7. Enter the LUN of the SCSI device in the **Logical unit number** field.
8. If the boot configuration is part of a **`zipl`** created menu configuration, enter the configuration number that identifies your SCSI boot configuration within the menu in the **Boot program selector** field.

Configuration number `0` specifies the default configuration.

See "Menu configurations" on page 76 for more details about menu configurations.

9. Type kernel parameters in the **Operating system specific load parameters** field.

   These parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration when booting Linux. The combined parameter string must not exceed 4096 bytes.

   Use ASCII characters only. If you enter characters other than ASCII characters, the boot process ignores the data in the **Operating system specific load parameters** field.
10. Accept the defaults for the remaining fields.
11. Click **OK** to start the boot process.

Results

Check the output on the preferred console (see "Console kernel parameter syntax" on page 42) to monitor the boot progress.

For information about IPL progress messages that are issued before the Linux kernel gets control, see *Small Computer Systems Interface (SCSI) IPL Machine Loader Messages*, SC28-7006.

Booting from an NVMe device

Use the SE or HMC to boot Linux in LPAR mode from a Non-Volatile Memory Express (NVMe) device.

Before you begin

- NVMe IPL devices are supported for LinuxONE III as of the firmware upgrade of November 2020.
- You need an NVMe device that is prepared with **`zipl`** (see "Preparing a boot device" on page 60).

Procedure

Perform these steps to boot from an NVMe boot device:

1. In the navigation pane of the HMC, expand **Systems Management** and select the hardware system that you want to work with. A table of LPARs is displayed on the **Partitions** tab in the content area.
2. Select the LPAR where you want to boot Linux.
3. In the **Tasks** area, expand **Recovery** and click **Load** as shown in the following graphic:



Figure 34. Load task on the HMC

4. On the **Load** panel, select load type **NVMe load**.

   Select the **Clear the main memory on this partition before loading it** check box only if you must clear memory. Memory clearing can considerably prolong the IPL procedure.



Figure 35. Load panel for NVMe load

5. Enter the PCIe function ID of the NVMe device in the **Load address** field. You can omit leading zeroes.
6. If the boot configuration is part of a **`zipl`** created menu configuration, type the configuration number that identifies your SCSI boot configuration within the menu in the **Boot program selector** field.

   Configuration number `0` specifies the default configuration.

   See "Menu configurations" on page 76 for more details about menu configurations.
7. For boot images in the secure-boot format, select the **Enable Secure Boot for Linux** option.
8. Type kernel parameters in the **Operating system specific load parameters** field.

   These parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration when booting Linux.

   Use ASCII characters only. If you enter characters other than ASCII characters, the boot process ignores the data in the **Operating system specific load parameters** field.
9. Accept the defaults for the remaining fields.
10. Click **OK** to start the boot process.

What to do next

Check the output on the preferred console (see "Console kernel parameter syntax" on page 42) to monitor the boot progress.

Secure boot

As of z15 and LinuxONE III, the operating system loader verifies that components that are loaded from SCSI disks or NVMe devices come from a trusted source. You can cancel loading for components that cannot be verified.

With secure boot enabled, an IPL fails if a component containing code is not signed or cannot be verified.

For details about how to prepare a device for secure boot, see “zipl modes and syntax overview” on page 58.

To check if a Linux instance was IPLed with secure boot see “Displaying current IPL parameters” on page 115.

Kernel interfaces are restricted in a kernel that is prepared for secure boot. In particular, in a kernel prepared for secure boot, all kernel modules must be signed by Red Hat. You cannot load modules that are not signed by Red Hat, like lin_tape.

KVM: You can IPL a KVM guest from a device with the secure boot format, but signatures are not verified.

Booting from tape

You can boot Linux in LPAR mode from tape.

Before you begin

You need a boot device that is prepared with **`zipl`** (see “Preparing a boot device” on page 60).

Procedure

Perform these steps to boot from a tape boot device:

1. In the navigation pane of the HMC, expand **Systems Management** and select the hardware system that you want to work with. A table of LPARs is displayed on the **Partitions** tab in the content area.
2. Select the LPAR where you want to boot Linux.
3. In the **Tasks** area, expand **Recovery** and click **Load** (see Figure 36 on page 103).



Figure 36. Load task on the HMC

4. Select load type **Standard load** (see Figure 31 on page 97).

   Select the **Clear the main memory on this partition before loading it** check box only if you must clear memory. Memory clearing can considerably prolong the IPL procedure.
5. Enter the device number of the tape boot device in the **Load address** field.
6. Click **OK** to start the boot process.

Results

Check the output on the preferred console (see “Console kernel parameter syntax” on page 42) to monitor the boot progress.

Loading Linux from removable media or from an FTP server

Instead of a boot loader, you can use SE functions to copy the Linux kernel image to your LPAR memory. After the Linux kernel is loaded, Linux is started using restart PSW.

Before you begin

You need installation data that includes a special file with installation information (with extension "ins"). This file can be in different locations:

- On a disk that is inserted in the CD-ROM or DVD drive of the system where the HMC runs
- In the file system of an FTP server that you can access through FTP from your HMC system

The `.ins` file contains a mapping of the location of installation data on the disk or FTP server and the memory locations where the data is to be copied.

For Red Hat Enterprise Linux 9.1, this file is called `generic.ins` and is in the root directory of the file system on the DVD.

Procedure

Perform these steps:

1. In the navigation pane of the HMC, expand **Systems Management** and select the hardware system that you want to work with. A table of LPARs is displayed on the **Partitions** tab in the content area.
2. Select the LPAR where you want to boot Linux.
3. In the **Tasks** area, expand **Recovery** and click **Load from Removable Media or Server** (see Figure 37 on page 104).



Figure 37. Load from Removable Media or Server task on the HMC

4. Specify the source of the code to be loaded.
   - For loading from a CD-ROM drive:

a. Select **Hardware Management Console removable media** (see Figure 38 on page 105).



Figure 38. Load from Removable Media or Server panel

b. Leave the **File location** field blank.

- For loading from an FTP server:
  a. Select **FTP Server**.
  b. Enter the IP address or host name of the FTP server with the installation code in the **Host name** entry field.
  c. Enter your user ID for the FTP server in the **User name** entry field.
  d. Enter your password for the FTP server in the **Password** entry field.
  e. If required by your FTP server, enter your account information in the **Account** entry field.
  f. Enter the path to the directory with the `generic.ins` in the file location entry field. You can leave this field blank if the file is in the FTP server's root directory.

5. Click **Continue** to display the **Select Software to Install** panel (Figure 39 on page 105).



Figure 39. Select Software to Install panel

6. Select the `generic.ins` file.
7. Click **OK** to start loading Linux.

Results

The kernel has started and the Red Hat Enterprise Linux 9.1 boot process continues.

Using the HMC Web Services API to boot in LPAR mode

You can boot Linux in LPAR mode remotely by using the HMC Web Services API. For information about the API, see *Hardware Management Console Web Services API* for your IBM Z or LinuxONE hardware.

You can find a client application that uses this API at https://github.com/zhmcclient/zhmccli. The examples that follow are based on this application, which provides the `zhmc` command as its user interface.

Hint: The `zhmc` command is case sensitive. For hardware and partition specifications, use the capitalization as shown in the HMC interface and the corresponding HMC API queries.

Booting from a SCSI boot device

The following example makes these assumptions about the hardware system, LPAR, and boot device:

- The name of the IBM Z or LinuxONE system is `M35`.
- The name of the LPAR is `m35lp55`.
- An FC-attached SCSI disk is prepared, with zipl, as a boot device.
- The LUN of the disk is `0x5241000000000000`
- The disk is accessed through WWPN `0x5005076300c20b8e`.
- The FCP device to access the disk has a bus ID `0.0.FC00`.

To start the IPL and boot process, issue:

```
# zhmc lpar scsi-load M35 m35lp55 FC00 5005076300c20b8e 5241000000000000
```

To view the operating system messages, issue:

```
# zhmc lpar console M35 m35lp55
```

For information about IPL progress messages that are issued before the Linux kernel gets control, see *Small Computer Systems Interface (SCSI) IPL Machine Loader Messages*, SC28-7006.

Booting Linux in a DPM partition

You can boot Linux in a DPM partition from a Hardware Management Console (HMC).

Booting in a DPM partition from a SCSI boot device

You can boot Linux in a Dynamic Partition Manager (DPM) partition from an FC-attached SCSI disk using the Hardware Management Console (HMC).

Before you begin

- You need a boot device that is prepared with **`zipl`** (see "Preparing a boot device" on page 60). For more information about SCSI boot devices, see Table 11 on page 94.
- You must have the SCSI IPL feature (FC9904) installed.
- SCSI boot devices are FC-attached disk volumes. In DPM mode, the HMC interface presents such disk volumes as part of SAN storage groups. To set up a SCSI disk as a boot device, you must know its storage group and the UUID that identifies it.

About this task

In Dynamic Partition Manager (DPM) mode, the boot process is initiated by the **Start** task for the partition. Before you can run the **Start** task, you must configure a boot volume for the partition. Subsequent boot processes for the partition use the configured boot volume configuration.

The steps that follow assume DPM version R3.1 or later. For more information about DPM, see *Dynamic Partition Manager (DPM) Guide* for your IBM Z or LinuxONE hardware.

Procedure

Perform these steps to set up and boot from a SCSI boot device for a DPM partition:

1. On the HMC, navigate to your partition.
   a) Expand **Systems Management** and select the hardware system that you want to work with.
   a) Select your partition on the **Partitions** tab in the content area.
2. Unless it is already configured, set up the boot device.
   a) In the **Tasks** area, click **Partition Details** (see Figure 40 on page 107).



Figure 40. Task area on the HMC

   b) In the left navigation pane of the **Partition Details** panel, select **Boot** to open the **Boot** tab.



Figure 41. Boot tab of the Partition Details panel

c) From the **Boot from** drop-down list, select "Storage Group(SAN)".

d) In the **Storage Group** section, select a storage group and a boot volume.

e) Boot configurations only: The **Boot program selector** field applies only to boot configurations that are part of a menu configuration that is created by **`zipl`**.

Enter the configuration number that identifies the boot configuration within the menu into this field. Configuration number `0` specifies the default configuration.

See “Menu configurations” on page 76 for more details about menu configurations.

f) Type kernel parameters in the **OS Load parameter** field.

These parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration when booting Linux. The combined parameter string must not exceed a length that is set when the kernel is compiled.

Use ASCII characters only. If you enter characters other than ASCII characters, the boot process ignores the data in this field.

g) Accept the defaults for the remaining fields.

With a configured boot device, you can boot according to step “3” on page 108. Alternatively, you can boot with the **`zhmc`** command, see “Using the HMC Web Services API to boot in DPM mode” on page 109.

3. Boot from the configured boot device.

a) Go to the HMC **Task** area for your partition.

b) Expand the **Daily** section, and click **Start**.

c) Optional: Check the output on the preferred console (see “Console kernel parameter syntax” on page 42) to monitor the boot progress.

For information about IPL progress messages that are issued before the Linux kernel gets control, see *Small Computer Systems Interface (SCSI) IPL Machine Loader Messages*, SC28-7006.

What to do next

You can repeatedly boot with the configured boot device settings. Proceed according to step "3" on page 108.

To change the boot device settings, proceed according to step "2" on page 107.

Using the HMC Web Services API to boot in DPM mode

You can boot Linux in a DPM partition remotely by using the HMC Web Services API. For information about the API, see *Hardware Management Console Web Services API* for your IBM Z or LinuxONE hardware.

You can find a client application that uses this API at https://github.com/zhmcclient/zhmccli. The examples that follow are based on this application, which provides the `zhmc` command as its user interface.

Hint: The `zhmc` command is case sensitive. For hardware and partition specifications, use the capitalization as shown in the HMC interface and the corresponding HMC API queries.

Booting from the configured boot device

The following example makes these assumptions about the hardware system, LPAR, and boot device:

- The name of the IBM Z or LinuxONE system is T46.
- The name of the DPM partition is `t46dp79`.
- A boot device has been configured for this DPM partition. For information about configuring boot devices for DPM partitions, see step "2" on page 107 in "Booting in a DPM partition from a SCSI boot device" on page 106.

To start the IPL and boot process, issue:

```
# zhmc partition start T46 t46dp79
```

To view the operating system messages, issue:

```
# zhmc partition console T46 t46dp79
```

For SCSI boot devices: For information about IPL progress messages that are issued before the Linux kernel gets control, see *Small Computer Systems Interface (SCSI) IPL Machine Loader Messages*, SC28-7006.

Booting Linux in a z/VM guest virtual machine

You boot Linux in a z/VM guest virtual machine by issuing CP commands from a CMS or CP session.

For more general information about z/VM guest environments for Linux, see *z/VM: Getting Started with Linux on System z*, SC24-6287.

Booting as a z/VM guest from a tape device

Boot Linux by issuing the IPL command with a tape boot device. The boot data on the tape must be arranged in a specific order.

Before you begin

You need a tape that is prepared as a boot device. A tape boot device must contain the following items in the specified order:

1. Tape boot loader code

   The tape boot loader code is included in the s390utils RPM.
2. Tape mark
3. Kernel image
4. Tape mark
5. Kernel parameters (optional)
6. Tape mark
7. Initial RAM disk (optional)
8. Tape mark
9. Tape mark

All tape marks are required even if an optional item is omitted. For example, if you do not provide an initial RAM disk image, the end of the boot information is marked with three consecutive tape marks. **zipl** prepared tapes conform to this layout. See "Preparing a boot device" on page 60 for information about preparing a tape with **zipl**.

Procedure

Perform these steps to start the boot process:

1. Establish a CMS or CP session with the z/VM guest virtual machine where you want to boot Linux.
2. Ensure that the boot device is accessible to your z/VM guest virtual machine.
3. Ensure that the correct tape is inserted and rewound.
4. Issue a command of this form:

```
#cp i <devno> clear parm <kernel_parameters>
```

   where

   <devno>
   : is the device number of the boot device as seen by the guest virtual machine.

   parm *<kernel_parameters>*
   : is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters that are used by your boot configuration (see "Preparing a boot device" on page 60 for information about the boot configuration).

     See also "Specifying kernel parameters when booting Linux" on page 27.

Booting as a z/VM guest from a DASD

Boot Linux by issuing the IPL command with a DASD boot device. You can specify additional parameters with the IPL command.

Before you begin

You need a DASD boot device that is prepared with **zipl** (see "Preparing a boot device" on page 60).

Procedure

Perform these steps to start the boot process:

1. Establish a CMS or CP session with the z/VM guest virtual machine where you want to boot Linux.
2. Ensure that the boot device is accessible to your z/VM guest virtual machine.
3. Issue a command of this form:

```
#cp i <devno> clear loadparm <n> parm <kernel_parameters>
```

where:

<devno>
specifies the device number of the boot device as seen by the guest.

loadparm *<n>S<m>*, where:

loadparm *<n>*
Applies only to menu configurations. Omit this specification if you are not addressing a menu configuration. *<n>* can be a positive integer, 0, or prompt.

Positive integer
Specifies the configuration number.

0
Specifies the default configuration.

prompt
Forces the menu to be displayed.

When the menu is displayed, you can specify additional kernel parameters (see "Example for a DASD menu configuration on z/VM" on page 111). These additional kernel parameters are appended to the parameters you might have provided in a parameter file. The combined parameter string must not exceed a length that is set when the kernel is compiled.

Depending on the menu configuration, omitting a value for *<n>* might display the menu or select the default configuration.

For more details about menu configurations, see "Menu configurations" on page 76. For examples of how a boot menu is displayed, see "Example for a DASD menu configuration (LPAR)" on page 98 and "Example for a DASD menu configuration on z/VM" on page 111.

loadparm S*<m>*
Applies only to zipl environments with site-specific sections. Omit this parameter if you are not working with site-specific sections.

S*<m>*
Specifies the section for which kernel parameters from a zipl environment should be used. If the section does not exist, an empty section is created. *<m>* can be a digit in the range 0 to 9.

S
Omitting *<m>* defaults to the lowest number of any enumerated sections. If no section is found, the common specification is used.

SS
Uses the subchannel set ID (SSID) as the section ID, for example, if the SSID is 0, section 0 is used.

For more information about sections, see "Defining sections in the zipl environment file" on page 88.

loadparm *<n>*S*<m>*
If you work with a combination of menu configurations and site-specific sections in a zipl environment file, specify both *<n>* and *<m>* as described in the preceding sections. Separate *<n>* and *<m>* with an S.

parm *<kernel_parameters>*
is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters that are used by your boot configuration (see "Preparing a boot device" on page 60 for information about the boot configuration).

See also "Specifying kernel parameters when booting Linux" on page 27.

Example for a DASD menu configuration on z/VM

Use the VI VMSG z/VM CP command to choose a boot configuration from a menu configuration.

This example illustrates how menu2 in the sample configuration file in Figure 23 on page 78 is displayed on the z/VM guest virtual machine console:

```
00: zIPL interactive boot menu
00:
00:  0. default (boot1)
00:
00:  1. boot1
00:  2. boot3
00:
00: Note: VM users please use '#cp vi vmsg <number> <kernel-parameters>'
00:
00: Please choose (default will boot in 30 seconds): #cp vi vmsg 2
```

You choose a configuration by specifying the configuration number. For example, to boot configuration boot3 specify

```
#cp vi vmsg 2
```

You can also specify additional kernel parameters by appending them to the configuration number. For example, you can specify:

```
#cp vi vmsg 2 maxcpus=1
```

These parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration when booting Linux.

Booting as a z/VM guest from a SCSI device

Boot Linux by issuing the IPL command with an FCP channel as the IPL device. You must specify the target port and LUN for the boot device in advance by setting the z/VM CP LOADDEV parameter.

Before you begin

You need a SCSI boot device that is prepared with **zipl** (see “Preparing a boot device” on page 60). For more information about SCSI boot devices, see Table 11 on page 94.

Procedure

Perform these steps to start the boot process:

1. Establish a CMS or CP session with the z/VM guest virtual machine where you want to boot Linux.
2. Ensure that the FCP channel that provides access to the SCSI boot disk is accessible to your z/VM guest virtual machine.
3. Specify the target port and LUN of the SCSI boot disk.

   Enter a command of this form:

   ```
   #cp set loaddev portname <wwpn> lun <lun>
   ```

   where:

   <wwpn>
   specifies the world wide port name (WWPN) of the target port in hexadecimal format. A blank separates the first eight digits from the final eight digits.

   <lun>
   specifies the LUN of the SCSI boot disk in hexadecimal format. A blank separating the first eight digits from the final eight digits.

   Example: To specify a WWPN 0x5005076300c20b8e and a LUN 0x5241000000000000:

   ```
   #cp set loaddev portname 50050763 00c20b8e lun 52410000 00000000
   ```

4. Optional for menu configurations: Specify the boot configuration (boot program in z/VM terminology) to be used. Enter a command of this form:

```
#cp set loaddev bootprog <n>
```

where *<n>* specifies the configuration number of the boot configuration. Omitting the bootprog parameter or specifying the value `0` selects the default configuration. For more information about menu configurations, see “Menu configurations” on page 76.

Example: To select a configuration with configuration number 2 from a menu configuration:

```
#cp set loaddev bootprog 2
```

5. Optional: Specify kernel parameters.

```
#cp set loaddev scpdata <APPEND|NEW> '<kernel_parameters>'
```

where:

<kernel_parameters>
specifies a set of kernel parameters to be stored as system control program data (SCPDATA). When booting Linux, these kernel parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration. The combined parameter string must not exceed 4096 bytes.

<kernel_parameters> must contain ASCII characters only. If characters other than ASCII characters are present, the boot process ignores the SCPDATA.

<kernel_parameters> as entered from a CMS or CP session is interpreted as lowercase on Linux. If you require uppercase characters in the kernel parameters, run the SET LOADDEV command from a REXX script instead. In the REXX script, use the "address command" statement. See *z/VM: REXX/VM Reference*, SC24-6314 and *z/VM: REXX/VM User's Guide*, SC24-6315 for details.

Optional: APPEND
appends kernel parameters to existing SCPDATA. This is the default.

Optional: NEW
replaces existing SCPDATA.

Examples:

- To append kernel parameter `novx` to the current SCPDATA:

```
#cp set loaddev scpdata 'novx'
```

- To replace the current SCPDATA with the kernel parameter `novx`:

```
#cp set loaddev scpdata NEW 'novx'
```

For a subsequent IPL command, this kernel parameter is concatenated to the end of the existing kernel parameters in your boot configuration.

6. Start the IPL and boot process by entering a command of this form:

```
#cp i <devno>
```

where *<devno>* is the device number of the FCP channel that provides access to the SCSI boot disk.

For information about IPL progress messages that are issued before the Linux kernel gets control, see *Small Computer Systems Interface (SCSI) IPL Machine Loader Messages*, SC28-7006.

Tip

You can specify the target port and LUN of the SCSI boot disk, a boot configuration, and SCPDATA all with a single SET LOADDEV command. See *z/VM: CP Commands and Utilities Reference*, SC24-6268 for more information about the SET LOADDEV command.

Booting from the z/VM reader

Boot Linux by issuing the IPL command with the z/VM reader as the IPL device. You first must transfer the boot data to the reader.

Before you begin

You need the following files, all in record format `fixed 80`:

- Linux kernel image with built-in z/VM reader boot loader code. This is the case for the default Red Hat Enterprise Linux 9.1 kernel.
- Kernel parameters (optional)
- Initial RAM disk image (optional)

About this task

This information is a summary of how to boot Linux from a z/VM reader. For more details, refer to Redpaper *Building Linux Systems under IBM VM*, REDP-0120.

Procedure

Proceed like this to boot Linux from a z/VM reader:

1. Establish a CMS session with the guest where you want to boot Linux.
2. Transfer the kernel image, kernel parameters, and the initial RAM disk image to your guest.

   You can obtain the files from a shared minidisk or use:

   - The z/VM sendfile facility.
   - An FTP file transfer in binary mode.

   Files that are sent to your reader contain a file header that you must remove before you can use them for booting. Receive files that you obtain through your z/VM reader to a minidisk.
3. Set up the reader as a boot device.
   a) Ensure that your reader is empty.
   b) Direct the output of the punch device to the reader. Issue:

   ```
   spool pun * rdr
   ```

   c) Use the CMS PUNCH command to transfer each of the required files to the reader.
   Be sure to use the "no header" option to omit the file headers.

   First transfer the kernel image.
   Second transfer the kernel parameters.
   Third transfer the initial RAM disk image, if present.

   For each file, issue a command of this form:

   ```
   pun <file_name> <file_type> <file_mode> (noh
   ```

   d) Optional: Ensure that the contents of the reader remain fixed.

   ```
   change rdr all keep nohold
   ```

If you omit this step, all files are deleted from the reader during the IPL that follows.

4. Issue the IPL command:

```
ipl 000c clear parm <kernel_parameters>
```

where:

0x000c
is the device number of the reader.

parm *<kernel_parameters>*
is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters that are used by your boot configuration (see “Preparing a boot device” on page 60 for information about the boot configuration).

See also “Specifying kernel parameters when booting Linux” on page 27.

Booting Linux on KVM

You boot Linux as a KVM guest on IBM Z from the KVM host, by starting a KVM virtual server.

About this task

For information about managing virtual servers, see *KVM Virtual Server Management*, SC34-2752.

Displaying current IPL parameters

To display the IPL parameters, use the **`lsreipl`** command with the **`-i`** option. Alternatively, a sysfs interface is available.

For more information about the **`lsreipl`** command, see “lsreipl - List IPL and re-IPL settings” on page 672. In sysfs, information about IPL parameters is available in subdirectories of `/sys/firmware/ipl`.

```
/sys/firmware/ipl/ipl_type
```

The `/sys/firmware/ipl/ipl_type` file contains the device type from which the kernel was booted. The following values are possible:

ccw
The IPL device is a CCW device, for example, a DASD, the z/VM reader, or a virtio block device.

fcp
The IPL device is an FCP device.

nvme
The IPL device is an NVMe device.

unknown
The IPL device is not known.

Depending on the IPL type, there might be more files in `/sys/firmware/ipl/`.

Further attributes for IPL type ccw

For IPL from a CCW device, the following attributes are present:

`device`
Contains the bus ID of the CCW device that is used for IPL, for example:

```
# cat /sys/firmware/ipl/device
0.0.1234
```

loadparm
Contains up to 8 characters for the loadparm that is used for selecting from a zipl boot menu during IPL of a CCW device, for example:

```
# cat /sys/firmware/ipl/loadparm
1
```

parm

Contains additional kernel parameters that are specified with the PARM parameter when booting with the z/VM CP IPL command, for example:

```
# cat /sys/firmware/ipl/parm
novx
```

See also "Specifying kernel parameters when booting Linux" on page 27.

A leading equal sign (=) indicates that the existing kernel parameters used by the boot configuration were ignored and the kernel parameters of the `parm` attribute where the only kernel parameters used for booting Linux. See "Replacing all kernel parameters in a boot configuration" on page 28.

Further attributes for IPL type fcp

For IPL from an FCP-attached SCSI device, the following attributes are present: (also see Chapter 12, "SCSI-over-Fibre Channel device driver," on page 177 for details):

binary_parameter
Contains the information of the preceding files in binary format.

bootprog
Contains the boot program number. Used for selecting from a zipl boot menu during IPL of a SCSI disk device.

br_lba
Contains the logical block address of the boot record on the boot device (usually 0).

device
Contains the bus ID of the FCP device that is used for IPL, for example:

```
# cat /sys/firmware/ipl/device
0.0.50dc
```

has_secure
Indicates whether the host environment supports secure boot. If the value is 1, secure boot is supported and the secure-boot enabled format can be used, see "Secure boot" on page 102.

lun
Contains the LUN used for IPL, for example:

```
# cat /sys/firmware/ipl/lun
0x5010000000000000
```

scp_data
Contains additional kernel parameters that are used when booting from a SCSI device (see "Booting as a z/VM guest from a SCSI device" on page 112 and "Booting from SCSI" on page 98). A leading equal sign (=) indicates that the existing kernel parameters used by the boot configuration were ignored and the kernel parameters of the `scp_data` attribute where the only kernel parameters used for booting Linux.

```
# cat /sys/firmware/ipl/scp_data
novx
```

secure
Read the sysfs attribute `/sys/firmware/ipl/secure` to check whether the Linux instance was IPLed with secure boot. Issue the following command:

```
# cat /sys/firmware/ipl/secure
1
```

If the value is 1, Linux was IPLed with secure boot.

wwpn
Contains the WWPN used for IPL, for example:

```
# cat /sys/firmware/ipl/wwpn
0x5005076300c20b8e
```

Further attributes for IPL type nvme

For IPL from an NVMe device, the following attributes are present:

binary_parameter
Contains the information of the other attributes a in binary format.

bootprog
Contains the boot program number that was used for selecting from a zipl boot menu during IPL of the NVMe device.

br_lba
Contains the logical block address of the boot record on the boot device (usually 0).

fid
PCIe function ID of the NVMe device.

has_secure
Indication of whether the host environment supports secure boot. If the value is 1, secure boot is supported and the secure-boot enabled format can be used, see “Secure boot” on page 102.

loadparm
Contains up to 8 characters for the loadparm.

nsid
NVMe name space ID of the NVMe device. Name space IDs are assigned by NVMe disk controllers to divide a physical NVMe device into multiple logical devices.

scp_data
Contains any additional kernel parameters that were used when booting from the NVMe device, for example:

```
# cat /sys/firmware/ipl/scp_data
novx
```

See “Booting from an NVMe device” on page 100.

A leading equal sign (=) indicates that the existing kernel parameters used by the boot configuration were ignored and the kernel parameters of the `scp_data` attribute where the only kernel parameters used for booting Linux. See “Replacing all kernel parameters in a boot configuration” on page 28.

secure
Indicates secure-boot mode. If the value is 1, the Linux instance was IPLed with secure boot.

Rebooting from an alternative source

When you reboot Linux, the system conventionally boots from the last used location. However, you can configure an alternative device to be used for re-IPL instead of the last used IPL device.

When the system is re-IPLed, the alternative device is used to boot the kernel.

To configure the re-IPL device, use the **chreipl** tool (see "chreipl - Modify the re-IPL configuration" on page 580).

Alternatively, you can use a sysfs interface. The virtual configuration files are located under `/sys/firmware/reipl`. To configure, write strings into the configuration files. The following re-IPL types can be set with the `/sys/firmware/reipl/reipl_type` attribute:

ccw
For ccw devices such as DASDs that are attached through ESCON or FICON®, and for virtio block devices on KVM guests.

fcp
For FCP SCSI devices. For information about boot devices, see Table 11 on page 94.

nvme
For PCIe-attached NVMe devices.

nss
For Named Saved Systems (z/VM only)

For each supported re-IPL type a sysfs directory is created under `/sys/firmware/reipl` that contains the configuration attributes for the device. The directory name is the same as the name of the re-IPL type.

When Linux is booted, the re-IPL attributes are set by default to the values of the boot device, which can be found under `/sys/firmware/ipl`.

Automatic path failover for re-IPL from an FC-attached SCSI disk

The **chreipl-fcp-mpath** tool set helps you to use multipath information for re-IPL path failover on a running Linux instance. When the configured re-IPL path becomes unavailable it automatically changes the configured re-IPL path to a different operational path to the same volume.

To use the tool set, install the `s390utils-chreipl-fcp-mpath` sub-package from s390-utils. Disable the feature by un-installing the sub-package. For more information, see the documentation installed with the sub-package: `/usr/share/doc/s390utils-chreipl-fcp-mpath/README.md`.

Attributes for ccw

You can find the attributes for re-IPL type ccw in the `/sys/firmware/reipl/ccw` sysfs directory.

device
Device number of the re-IPL device. For example 0.0.7412 or 0.1.5119.

loadparm
Up to eight characters for the loadparm used to select the boot configuration in the zipl menu (if available).

parm
A 64-byte string of kernel parameters that is concatenated to the boot command-line. The PARM parameter can be set only for Linux on z/VM. See also "Specifying kernel parameters when booting Linux" on page 27.

A leading equal sign (=) means that the existing kernel parameter line in the boot configuration is ignored and the boot process uses the kernel parameters in the `parm` attribute only. See also "Replacing all kernel parameters in a boot configuration" on page 28.

clear
A flag that controls memory clearing for a reboot from the device. Valid values are 1 to clear memory during the boot process, or 0 to omit clearing memory during the boot process. Booting without clearing memory is faster and it is the default.

With memory clearing enabled, all hotplug memory is offline after the reboot. Without memory clearing, the online status of hotplug memory is preserved. For more information, see "Memory state and reboot" on page 364.

Attributes for fcp

You can find the attributes for re-IPL type fcp in the `/sys/firmware/reipl/fcp` sysfs directory.

`device`
Device number of the FCP device that is used for re-IPL. For example, 0.0.7412.

Note: IPL is possible only from subchannel set 0.

`wwpn`
World wide port number of the FCP re-IPL device.

`lun`
Logical unit number of the FCP re-IPL device.

`bootprog`
Boot program selector. Used to select the boot configuration in the zipl menu (if available).

`br_lba`
Boot record logical block address. Master boot record. Is always 0 for Linux.

`scp_data`
Kernel parameters to be used for the next FCP re-IPL.

A leading equal sign (=) means that the existing kernel parameter line in the boot configuration is ignored and the boot process uses the kernel parameters in the `scp_data` attribute only. See also "Replacing all kernel parameters in a boot configuration" on page 28.

clear
A flag that controls memory clearing for a reboot from the device. Valid values are 1 to clear memory during the boot process, or 0 to omit clearing memory during the boot process. Booting without clearing memory is faster and it is the default.

Regardless of the setting for memory clearing, all hotplug memory is offline after the reboot. For more information, see "Memory state and reboot" on page 364.

This attribute is present only for Linux in LPAR mode on z14 or later hardware.

Attributes for nvme

You can find the attributes for re-IPL type nvme in the `/sys/firmware/reipl/nvme` sysfs directory.

`bootprog`
Boot program selector. Used to select the boot configuration in the zipl menu (if available).

`br_lba`
Boot record logical block address. Master boot record. Is always 0 for Linux.

clear
A flag that controls memory clearing for a reboot from the device. Valid values are 1 to clear memory during the boot process, or 0 to omit clearing memory during the boot process. Booting without clearing memory is faster and it is the default.

Regardless of the setting for memory clearing, all hotplug memory is offline after the reboot. For more information, see "Memory state and reboot" on page 364.

This attribute is present only for Linux in LPAR mode on LinuxONE hardware as of LinuxONE III.

`fid`
PCIe function ID of the NVMe device. This value specifies the slot at `/sys/bus/pci/slots`.

`loadparm`
Up to eight characters for the loadparm.

`nsid`
NVMe name space ID. Name space IDs are assigned by NVMe disk controllers to divide a physical NVMe device into multiple logical devices.

scp_data
Kernel parameters to be used for the next NVMe re-IPL. See also “Specifying kernel parameters when booting Linux” on page 27.

A leading equal sign (=) means that the existing kernel parameter line in the boot configuration is ignored and the boot process uses the kernel parameters in the `scp_data` attribute only. See also “Replacing all kernel parameters in a boot configuration” on page 28.

Attributes for nss

You can find the attributes for re-IPL type nss in the `/sys/firmware/reipl/nss` sysfs directory.

name
Name of the NSS. The NSS name can be 1-8 characters long and must consist of alphabetic or numeric characters. The following examples are all valid NSS names: 73248734, NSSCSITE, or NSS1234.

parm
A 56-byte string of parameters that is passed to the NSS to be booted.

Kernel panic settings

Set the attribute `/sys/firmware/shutdown_actions/on_panic` to `reipl` to make the system re-IPL with the current re-IPL settings if a kernel panic occurs.

For Linux in LPAR mode and Linux on z/VM, you might want to trigger a system dump in response to a kernel panic. See also the description of the **dumpconf** tool in *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751 on the IBM Documentation website at ibm.com/docs/en/linux-on-systems?topic=linuxone-service-support-troubleshooting

Examples for configuring re-IPL

Typical examples include configuring re-IPL from an FCP device and specifying parameters for re-IPL.

- To configure a CCW device with bus ID `0.0.7e78` as the re-IPL device:

```
# chreipl 0.0.7e78
```

  Alternatively, you can write directly to sysfs:

```
# echo 0.0.7e78 > /sys/firmware/reipl/ccw/device
```

- To ensure that memory is cleared during the re-IPL from the CCW device:

  Ensure that the `clear` attribute exists:

```
# ls /sys/firmware/reipl/ccw/clear
```

  If the `clear` attribute does not exist, memory is always cleared for re-IPL in your environment, and no further action is needed.

  If the `clear` attribute exists, write 1 to the attribute to configure memory clearing:

```
# echo 1 > /sys/firmware/reipl/ccw/clear
```

  Hint: If supported in your environment, re-IPL without clearing memory is the default. For large memory sizes, clearing memory can considerably slow down the re-IPL process.

- To configure an FCP re-IPL device 0.0.5711 with a LUN 0x1711000000000000 and a WWPN 0x5005076303004715 with an additional kernel parameter `novx`:

```
# chreipl 0.0.5711 0x5005076303004715 0x1711000000000000 -p "novx"
```

Alternatively, you can write directly to sysfs. For an FCP re-IPL device, additional boot parameters, as specified with the `-p` option, are stored as scp data. Therefore, the `novx` kernel parameter of the example is written to `/sys/firmware/reipl/fcp/scp_data`.

```
# echo 0.0.5711 > /sys/firmware/reipl/fcp/device
# echo 0x5005076303004715 > /sys/firmware/reipl/fcp/wwpn
# echo 0x1711000000000000 > /sys/firmware/reipl/fcp/lun
# echo "novx" > /sys/firmware/reipl/fcp/scp_data
# echo fcp > /sys/firmware/reipl/reipl_type
```

- To specify additional kernel parameters for re-IPL of an instance of Linux on z/VM:

  Write the parameters to the `parm` attribute:

```
# echo "novx" > /sys/firmware/reipl/ccw/parm
```

Chapter 8. Shutdown actions

Several triggers can cause Linux to shut down. For each shutdown trigger, you can configure a specific shutdown action to be taken as a response.

Table 12. Shutdown triggers and default action overview

Trigger	Command or condition	Default shutdown action
halt	Linux **shutdown -H** command	stop
poff	Linux **poweroff** or **shutdown -P** command	stop
reboot	Linux **reboot** or **shutdown -r** command	reipl
restart	Depending on the hyperisor environment: **LPAR** A **PSW restart** on the HMC **z/VM** A CP **system restart** command **KVM** A virsh command on the KVM host	stop
panic	Linux kernel panic	stop

The available shutdown actions are summarized in Table 13 on page 123.

Table 13. Shutdown actions

Action	Explanation	See also
stop	For `panic` or `restart`, enters a disabled wait state. For all other shutdown triggers, stops all CPUs. For Linux on KVM, frees the resources that were used by the Linux instance. Depending on your virtual server configuration, the memory might be preserved.	n/a
ipl	Performs an IPL according to the specifications in `/sys/firmware/ipl`.	"Displaying current IPL parameters" on page 115
reipl	Performs an IPL according to the specifications in `/sys/firmware/reipl`.	"Rebooting from an alternative source" on page 117
dump	For Linux in LPAR mode and Linux on z/VM, creates a dump according to the specifications in `/sys/firmware/dump`.	*Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751
dump_reipl	For Linux in LPAR mode and Linux on z/VM, performs the dump action followed by the `reipl` action.	*Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751

Table 13. Shutdown actions (continued)		
Action	**Explanation**	**See also**
vmcmd	For Linux on z/VM, issues one or more z/VM CP commands according to the specifications in `/sys/firmware/vmcmd`.	"Configuring z/VM CP commands as a shutdown action" on page 126

Use **lsshut** to find out which shutdown action is configured for each shutdown trigger, see "lsshut - List the current system shutdown actions" on page 675.

Use the applicable command to configure the shutdown action for a shutdown trigger:

- For `halt`, `power off`, and `reboot` use **chshut**, see "chshut - Control the system shutdown actions" on page 585.
- For `restart` and `panic` on Linux in LPAR mode or Linux on z/VM, use **dumpconf**, see *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751

kdump for restart and panic

If kdump is set up for a Linux instance, kdump is started to create a dump, regardless of the shutdown actions that are specified for `restart` and `panic`. With kdump, these settings act as a backup that is used only if kdump fails.

Note: kdump is not a shutdown action that you can set as a sysfs attribute value for a shutdown trigger. See *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751 about how to set up kdump.

The shutdown configuration in sysfs

The configured shutdown action for each shutdown trigger is stored in a sysfs attribute `/sys/firmware/shutdown_actions/on_<trigger>`.



Figure 42. sysfs branch with shutdown action settings

The preferred way to read or change these settings is using the **lsshut**, **chshut**. For dump actions on Linux in LPAR mode or Linux on z/VM, the preferred way is the **dumpconf** command. Alternatively, you can read and write to the `/sys/firmware/shutdown_actions/on_<trigger>` attributes.

Examples

- This command reads the shutdown configuration:

```
# lsshut
Trigger          Action
=======================
Halt             stop
Power off        stop
Reboot           reipl
Restart          kdump,stop
Panic            kdump,stop
```

- This command reads the shutdown setting for the `poff` shutdown trigger.

```
# cat /sys/firmware/shutdown_actions/on_poff
stop
```

- This command changes the setting for the `reboot` shutdown trigger to `ipl`:

```
# chshut reboot ipl
```

  Alternatively, you can directly write the new setting to sysfs:

```
# echo ipl > /sys/firmware/shutdown_actions/on_reboot
```

- This command changes the setting for the `restart` shutdown trigger to `ipl`:

```
# echo ipl > /sys/firmware/shutdown_actions/on_restart
```

Details for the `ipl`, `reipl`, `dump`, and `vmcmd` shutdown actions are contained in the corresponding subdirectories in `/sys/firmware`. For example, `/sys/firmware/ipl` contains specifications for an IPL device and other IPL parameters.

Configuring z/VM CP commands as a shutdown action

Use **chshut** and **dumpconf** to configure a CP command as a shutdown action, or directly write to the relevant sysfs attributes.

Before you start: This information applies to Linux on z/VM only.

In sysfs, two attributes are required to set a z/VM CP command as a shutdown action for a trigger *<trigger>*:

- `/sys/firmware/shutdown_actions/on_<trigger>` must be set to `vmcmd`.
- `/sys/firmware/vmcmd/on_<trigger>` specifies the z/VM CP command.

The values of the attributes in the `/sys/firmware/vmcmd` directory must conform to these rules:

- The value must be a valid z/VM CP command.
- The commands, including any z/VM user IDs or device numbers, must be specified with uppercase characters.
- Commands that include blanks must be delimited by double quotation marks (").
- The value must not exceed 127 characters.

You can specify multiple z/VM CP commands that are separated by the newline character "\n". Each newline is counted as one character. When writing values with multiple commands, use this syntax to ensure that the newline character is processed correctly:

```
# echo -e <cmd1>\n<cmd2>... | cat > /sys/firmware/vmcmd/on_<trigger>
```

where *<cmd1>*\n*<cmd2>*`...` are two or more z/VM CP commands and `on_<trigger>` is one of the attributes in the `vmcmd` directory.

The **-e echo** option and redirect through **cat** are required because of the newline character.

Example for a single z/VM CP command

Issue the following command to configure the z/VM CP LOGOFF command as the shutdown action for the `poff` shutdown trigger.

```
# chshut poff vmcmd "LOGOFF"
```

Alternatively, you can issue the following commands to directly write the shutdown configuration to sysfs:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_poff
# echo LOGOFF > /sys/firmware/vmcmd/on_poff
```

Figure 43 on page 127 illustrates the relationship of the sysfs attributes for this example.



Figure 43. sysfs branch with shutdown action settings

Example for multiple z/VM CP commands

Issue the following command to configure two z/VM CP commands as the shutdown action for the `poff` shutdown trigger. First a message is sent to user OPERATOR, and then the LOGOFF command is issued.

```
# chshut poff vmcmd "MSG OPERATOR Going down" vmcmd "LOGOFF"
```

Alternatively, you can issue the following commands to directly write the shutdown configuration to sysfs:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_poff
# echo -e "MSG OPERATOR Going down\nLOGOFF" | cat > /sys/firmware/vmcmd/on_poff
```

Chapter 9. The diag288 watchdog device driver

The watchdog device driver provides Linux watchdog applications with access to the watchdog timer.

You can use the diag288 watchdog in these environments:

- Linux on z/VM
- Linux in LPAR mode as of z13s® and z13® with the enhancements of February 2016.
- Linux as a KVM guest

The diag288 watchdog device driver provides the following features:

- Access to the watchdog timer on IBM Z.
- An API for watchdog applications (see “External programming interfaces ” on page 131).

Watchdog applications can be used to set up automated restart mechanisms. Watchdog-based restart mechanisms are an alternative to a networked heartbeat with STONITH.

Watchdog applications that communicate directly with the IBM Z firmware, the z/VM control program (CP), or the KVM host do not require a third operating system to monitor a heartbeat.

What you should know about the diag288 watchdog device driver

The watchdog function comprises two components: a watchdog application that runs on the Linux instance being controlled and a watchdog timer outside the Linux instance.For Linux in LPAR mode, the timer runs in the IBM Z firmware. For Linux on z/VM the timer is provided by z/VM CP. For Linux on KVM, the timer runs on the KVM host.

While the Linux instance operates satisfactorily, the watchdog application reports a positive status to the watchdog timer at regular intervals. The watchdog application uses a device node to pass these status reports to the timer (Figure 44 on page 129).



Figure 44. Watchdog application and timer

The watchdog application typically derives its status by monitoring critical network connections, file systems, and processes on the Linux instance. If a specified time elapses without a positive report being received by the watchdog timer, the watchdog timer assumes that the Linux instance is in an error state. The watchdog timer then triggers a predefined action against the Linux instance. For example, Linux might be shut down or rebooted, or a system dump might be initiated. For information about setting the default timer and performing other actions, see “External programming interfaces ” on page 131.

Linux on z/VM only: Loading or saving a DCSS can take a long time during which the virtual machine does not respond, depending on the size of the DCSS. As a result, a watchdog might time out and restart the guest. You are advised not to use the watchdog in combination with loading or saving DCSSs.

See also the generic watchdog documentation in the Linux kernel source tree under `Documentation/watchdog`.

Loading and configuring the diag288 watchdog device driver

You configure the diag288 watchdog device driver when you load the module.

watchdog module parameter syntax



Notes:

[1] `cmd=` and `conceal=` apply only to Linux on z/VM and are ignored for Linux in LPAR mode and Linux on KVM.

where:

<command>
configures the shutdown action to be taken if Linux on z/VM fails. The default, "SYSTEM RESTART", configures the shutdown action that is specified for the `restart` shutdown trigger (see Chapter 8, “Shutdown actions,” on page 123).

Any other specification dissociates the timeout action from the `restart` shutdown trigger. Instead, the specification is issued by CP and must adhere to these rules:

- It must be a single valid CP command
- It must not exceed 230 characters
- It must be enclosed by quotation marks if it contains any blanks or newline characters

The specification is converted from ASCII to uppercase EBCDIC.

For details about CP commands see *z/VM: CP Commands and Utilities Reference*, SC24-6268.

On an running instance of Linux on z/VM, you can write to `/sys/module/diag288_wdt/parameters/cmd` to replace the command you specify when loading the module. Through this sysfs interface, you can also specify multiple commands to be issued, see Examples for more details.

The preferred method for configuring a timeout action other than a system restart is to configure a different shutdown action for the `restart` shutdown trigger.

conceal=1
enables the protected application environment where the guest is protected from unexpectedly entering CP READ. Do not enable the protected environment for guests with multiprocessor configurations. The protected application facility supports only virtual uniprocessor systems.

For details, see the "SET CONCEAL" section of *z/VM: CP Commands and Utilities Reference*, SC24-6268.

<nowayout_flag>
determines what happens when the watchdog device node is closed by the watchdog application.

If the flag is set to 1 (default), the watchdog timer keeps running and triggers an action if no positive status report is received within the specified time interval. If the character "V" is written to the device and the flag is set to `0`, the z/VM watchdog timer is stopped and the Linux instance continues without the watchdog support.

Examples for Linux on z/VM

The following command loads the watchdog module and determines that, on failure, the Linux instance is to be IPLed from a device with devno 0xb1a0. The protected application environment is not enabled. The watchdog application can close the watchdog device node after writing "V" to it. As a result the watchdog timer becomes ineffective and does not IPL the guest.

```
# modprobe diag288_wdt cmd="ipl b1a0" nowayout=0
```

The following example shows how to specify multiple commands to be issued.

```
# /usr/bin/printf "<cmd1>\n<cmd2>\n<cmd3>" > /sys/module/diag288_wdt/parameters/cmd
```

where *<cmd1>*, *<cmd2>*, and *<cmd3>*are z/VM commands.

Use the **printf** version at `/usr/bin/printf`. The built-in **printf** command from bash might not process the newline characters as intended.

To verify that your commands have been accepted, issue: To verify that your commands have been accepted, issue:

```
# cat /sys/module/diag288_wdt/parameters/cmd
<cmd1>
<cmd2>
<cmd3>
```

Note: You cannot specify multiple commands as module parameters while loading the module.

Setting the timeout action

How to configure the timeout action for the diag288 watchdog device driver depends on your hypervisor environment.

LPAR
For Linux in LPAR mode, the shutdown action is defined through the `restart` shutdown trigger (see Chapter 8, “Shutdown actions,” on page 123).

z/VM
For Linux on z/VM, the shutdown action is defined through the `restart` shutdown trigger.

You can also use the `diag288_wdt.cmd=` kernel parameter or the `cmd=` module parameter to bypass the `restart` shutdown trigger and directly specify a z/VM CP command to be issued.

KVM
For Linux on KVM, the shutdown action is defined in the virtual server configuration on the KVM hypervisor.

External programming interfaces

There is an API for applications that work with the watchdog device driver.

Application programmers: This information is intended for programmers who want to write watchdog applications that work with the watchdog device driver.

For information about the API and the supported IOCTLs, see the `Documentation/watchdog/watchdog-api.txt` file in the Linux source tree.

The default watchdog timeout is 30 seconds, the minimum timeout that can be set through the IOCTL WDIOC_SETTIMEOUT is 15 seconds.

Chapter 10. KASLR support

With kernel address space layout randomization (KASLR), the kernel is loaded to a random location in memory.

Loading the kernel to a random location can protect against attacks that rely on knowledge of the kernel addresses.

The KASLR feature is enabled by default. You can use the nokaslr kernel parameter to disable it, see “nokaslr - Disable kernel randomization” on page 787.

With KASLR enabled, the kernel is loaded to a random address, but kernel messages can reveal kernel internal addresses. Prevent access to the kernel messages for unprivileged users by setting the `dmesg_restrict` sysctl to 1. This setting restricts **dmesg** access to users with CAP_SYSLOG privilege.

Kernel addresses can also be compromised through `/proc` and other interfaces. To prevent this, set the `kptr_restrict` sysctl to 1.

For more information about the `dmesg_restrict` and `kptr_restrict` sysctls, see the `Documentation/sysctl/kernel.txt` in the kernel source tree.

Part 3. Storage

LPAR and z/VM: This part applies to Linux in LPAR mode and to Linux on z/VM.

Red Hat Enterprise Linux 9.1 includes several storage device drivers that are specific to z/Architecture.

For information about storage networks and I/O to storage devices, see
www.ibm.com/it-infrastructure/z/capabilities/networking

Newest version

You can find the newest version of this book at
ibm.com/docs/en/linux-on-systems?topic=linuxone-distributions

Restrictions

For prerequisites and restrictions see the IBM Z architecture specific information in the Red Hat Enterprise Linux 9.1 release notes at
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

Chapter 11. DASD device driver

The DASD device driver provides access to all real or emulated direct access storage devices (DASD) that can be attached to the channel subsystem of an IBM mainframe.

DASD devices include various physical media on which data is organized in blocks or records or both. The blocks or records in a DASD can be accessed for read or write in random order.

Traditional DASD devices are attached to a control unit that is connected to a mainframe I/O channel. Today, these real DASDs have been largely replaced by emulated DASDs. For example, emulated DASDs can be the volumes of the IBM System Storage® DS8000® Turbo, or the volumes of the IBM System Storage DS6000™. These emulated DASD are completely virtual and the identity of the physical device is hidden.

SCSI disks that are attached through an FCP channel are not classified as DASD. They are handled by the zfcp driver (see Chapter 12, "SCSI-over-Fibre Channel device driver," on page 177).

Features

The DASD device driver supports a wide range of disk devices and disk functions.

- The DASD device driver has no dependencies on the adapter hardware that is used to physically connect the DASDs to the IBM Z hardware. You can use any adapter that is supported by the IBM Z hardware (see www.ibm.com/systems/support/storage/ssic/interoperability.wss for more information).
- The DASD device driver supports ESS virtual ECKD type disks
- The DASD device driver supports the control unit attached physical ECKD (Extended Count Key Data) and FBA (Fixed Block Access) devices as summarized in Table 14 on page 137:

Table 14. Supported control unit attached DASD

Device format	Control unit type	Device type
ECKD	1750	3380 and 3390
ECKD	2107	3380 and 3390
ECKD	2105	3380 and 3390
ECKD	3990	3380 and 3390
ECKD	9343	9345
ECKD	3880	3390
FBA	6310	9336
FBA	3880	3370

  All models of the specified control units and device types can be used with the DASD device driver. This includes large devices with more than 65520 cylinders, for example, 3390 Model A. Check the storage support statement to find out what works for Red Hat Enterprise Linux.
- The DASD device driver provides a disk format with up to three partitions per disk. See "IBM Z compatible disk layout" on page 139 for details.
- The DASD device driver provides an option for extended error reporting for ECKD devices. Extended error reporting can support high availability setups.
- The DASD device driver supports parallel access volume (PAV) and HyperPAV on storage devices that provide this feature. For more information about PAV and HyperPAV, see *How to Improve Performance with PAV*, SC33-8414. Use the **dasdstat** command to check whether a DASD uses PAV, see "Scenario: Verifying that PAV and HPF are used" on page 162.

- The DASD device driver supports High Performance FICON, including multitrack requests, on storage devices that provide this feature. Use the **dasdstat** command to check whether a DASD uses High Performance FICON, see "Scenario: Verifying that PAV and HPF are used" on page 162.
- The DASD device driver supports large volumes (devices with more than 65520 cylinders, for example, 3390 Model A), solid state devices, and encrypted devices.
- The DASD device driver supports extent space efficient (ESE) DASDs, see "Managing extent space efficient DASDs" on page 169.

What you should know about DASD

The DASD device driver supports various disk layouts with different partitioning capabilities. The DASD device naming scheme helps you to keep track of your DASDs and DASD device nodes.

The IBM label partitioning scheme

Linux on IBM Z supports the same standard DASD format that is also used by traditional mainframe operating systems, but it also supports any other Linux partition table.

The DASD device driver is embedded into the Linux generic support for partitioned disks. As a result, you can use any partition table format that is supported by Linux for your DASDs.

Traditional mainframe operating systems (such as, z/OS, z/VM, and z/VSE®) expect a standard DASD format. In particular, the format of the first two tracks of a DASD is defined by this standard. These tracks include the IBM Z IPL, label, and for some layouts VTOC records. Partitioning schemes for platforms other than IBM Z generally do not preserve these mainframe specific records.

Red Hat Enterprise Linux 9.1 for IBM Z includes the IBM label partitioning scheme that preserves the IBM Z IPL, label, and VTOC records. With this partitioning scheme, Linux can share a disk with other mainframe operating systems. For example, a traditional mainframe operating system can handle backup and restore for a partition that is used by Linux.

The following sections describe the layouts that are supported by the IBM label partitioning scheme:

- "IBM Z compatible disk layout" on page 139
- "Linux disk layout" on page 141
- "CMS disk layout" on page 141

DASD partitions

Partitioning DASD has the same advantages as for other disk types, but there are some prerequisites and a special tool, **fdasd**.

A DASD partition is a contiguous set of DASD blocks that is treated by Linux as an independent disk and by the traditional mainframe operating systems as a data set.

With the Linux disk layout (LDL) and the CMS disk layout, you always have a single partition only. This partition is defined by the LDL or CMS formatted area of the disk. With the compatible disk layout, you can have up to three partitions.

There are several reasons why you might want to have multiple partitions on a DASD, for example:

Limit data growth
: Runaway processes or undisciplined users can consume disk space to an extend that the operating system runs short of space for essential operations. Partitions can help to isolate the space that is available to particular processes.

Encapsulate your data
: If a file system gets damaged, this damage is likely to be restricted to a single partition. Partitioning can reduce the scope of data damage.

Recommendations:

- Use **fdasd** to create or alter partitions on ECKD type DASD that are formatted with the compatible disk layout. If you use another partition editor, it is your responsibility to ensure that partitions do not overlap. If they do, data damage occurs.
- Leave no gaps between adjacent partitions to avoid wasting space. Gaps are not reported as errors, and can be reclaimed only by deleting and re-creating one or more of the surrounding partitions and rebuilding the file system on them.

A disk need not be partitioned completely. You can begin by creating only one or two partitions at the start of your disk and convert the remaining space to a partition later.

There is no facility for moving, enlarging, or reducing partitions, because **fdasd** has no control over the file system on the partition. You can only delete and re-create them. Changing the partition table results in loss of data in all altered partitions. It is up to you to preserve the data by copying it to another medium.

IBM Z compatible disk layout

With the compatible disk layout a DASD can have up to three partitions that can be accessed by traditional mainframe operating systems.

You can only format ECKD type DASD with the compatible disk layout.

Figure 45 on page 139 illustrates a DASD with the compatible disk layout.



Figure 45. Compatible disk layout

The IPL records, volume label (VOL1), and VTOC of disks with the compatible disk layout are on the first two tracks of the disks. These tracks are not intended for use by Linux applications. Using the tracks can result in data loss.

Linux can address the device as a whole as /dev/dasd<*x*>, where <*x*> can be one to four letters that identify the individual DASD (see "DASD naming scheme" on page 142). See "DASD device nodes" on page 143 for alternative addressing possibilities.

Disks with the compatible disk layout can have one to three partitions. Linux addresses the first partition as /dev/dasd<*x*>1, the second as /dev/dasd<*x*>2, and the third as /dev/dasd<*x*>3.

You use the **dasdfmt** command (see "dasdfmt - Format a DASD" on page 615) to format a disk with the compatible disk layout. You use the **fdasd** command (see "fdasd – Partition a DASD" on page 632) to create and modify partitions.

Volume label

The volume label includes information about the disk layout, the VOLSER, and a pointer to the VTOC.

The DASD volume label is located in the third block of the first track of the device (cylinder 0, track 0, block 2). This block has a 4-byte key, and an 80-byte data area with the following content:

key
for disks with the compatible disk layout, contains the four EBCDIC characters "VOL1" to identify the block as a volume label.

label identifier
is identical to the key field.

VOLSER
is a name that you can use to identify the DASD device. A volume serial number (VOLSER) can be one to six EBCDIC characters. If you want to use VOLSERs as identifiers for your DASD, be sure to assign unique VOLSERs.

You can assign VOLSERs from Linux by using the **dasdfmt** or **fdasd** command. These commands enforce that VOLSERs:

- Are alphanumeric
- Are uppercase (by uppercase conversion)
- Contain no embedded blanks
- Contain no special characters other than $, #, @, and %

  Tip: Avoid special characters altogether.

Note: The VOLSER values SCRTCH, PRIVAT, MIGRAT, or L*nnnnn* (An "L" followed by five digits) are reserved for special purposes by other mainframe operating systems and should not be used by Linux.

These rules are more restrictive than the VOLSERs that are allowed by the traditional mainframe operating systems. For compatibility, Linux tolerates existing VOLSERs with lowercase letters and special characters other than $, #, @, and %. Enclose VOLSERs with special characters in single quotation marks if you must specify it, for example, as a command parameter.

VTOC address
contains the address of a standard IBM format 4 data set control block (DSCB). The format is: *cylinder* (2 bytes) *track* (2 bytes) *block* (1 byte).

All other fields of the volume label contain EBCDIC space characters (code 0x40).

VTOC

Instead of a regular Linux partition table, Red Hat Enterprise Linux 9.1 for IBM Z, like other mainframe operating systems, uses a Volume Table Of Contents (VTOC).

The VTOC contains pointers to the location of every data set on the volume. These data sets form the Linux partitions.

The VTOC is on the second track (cylinder 0, track 1). It contains a number of records, each written in a separate data set control block (DSCB). The number of records depends on the size of the volume:

- One DSCB that describes the VTOC itself (format 4)
- One DSCB that is required by other operating systems but is not used by Linux. **fdasd** sets it to zeros (format 5).
- For volumes with more than 65534 cylinders, 1 DSCB (format 7)
- For each partition:
  - On volumes with 65534 or less cylinders, 1 DSCB (format 1)
  - On volumes with more than 65534 cylinders, 1 format 8 and one format 9 DSCB

  The key of the format 1 or format 8 DSCB contains the data set name, which identifies the partition to z/OS, z/VM, and z/VSE.

The VTOC can be displayed with standard IBM Z tools such as VM/DITTO. A Linux DASD with physical device number 0x0193, volume label "LNX001", and three partitions might be displayed like this example:

```
                          VM/DITTO DISPLAY VTOC                LINE 1 OF 5
===>                                                         SCROLL ===> PAGE

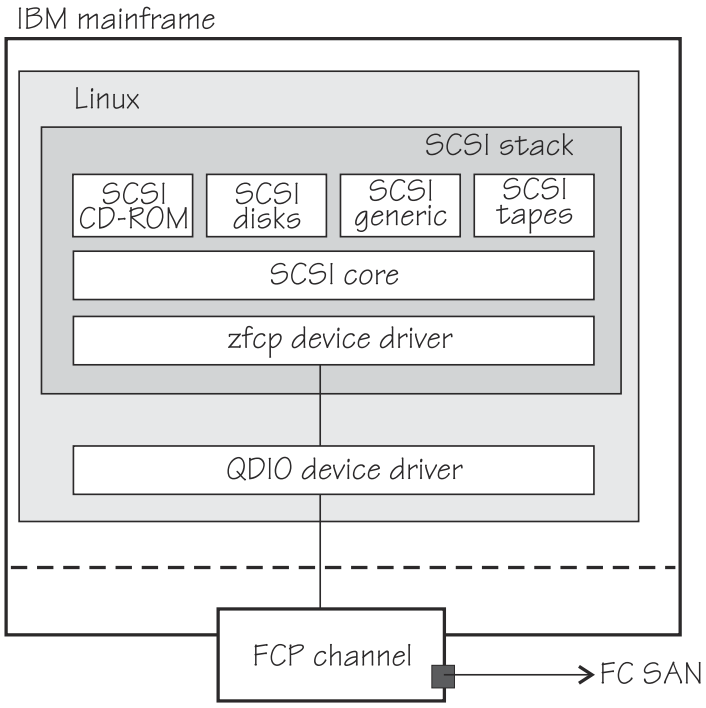
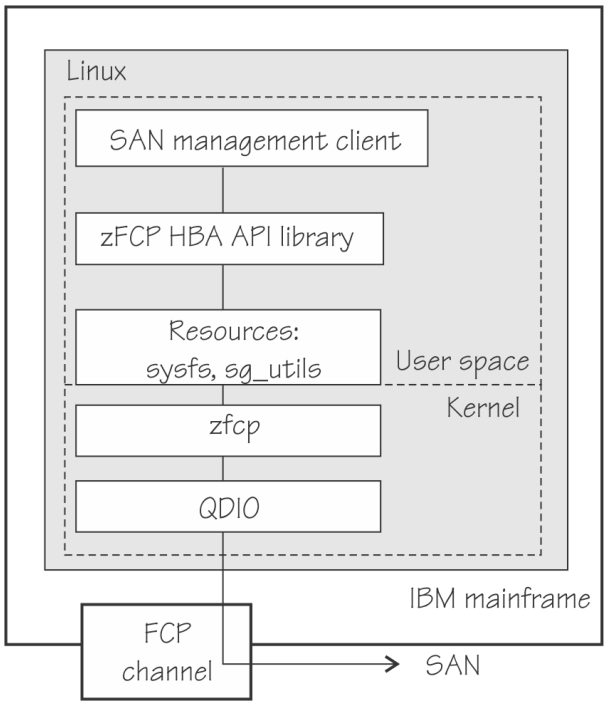
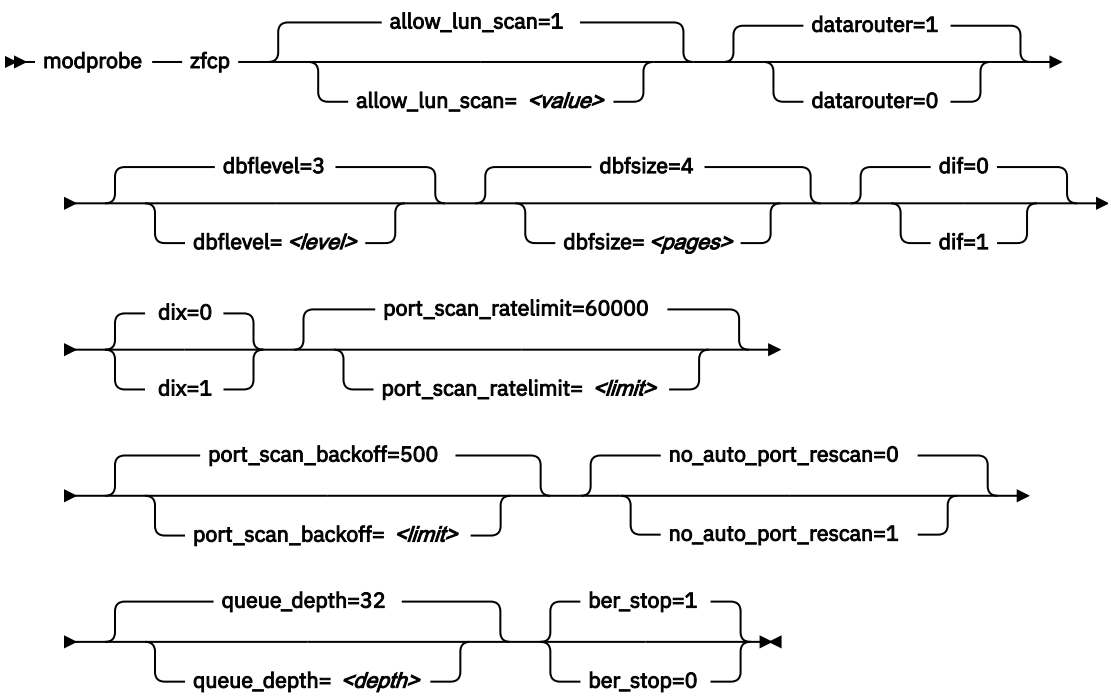
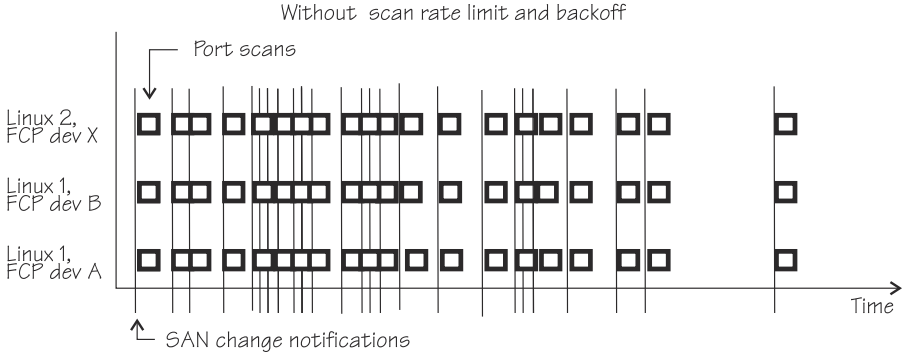
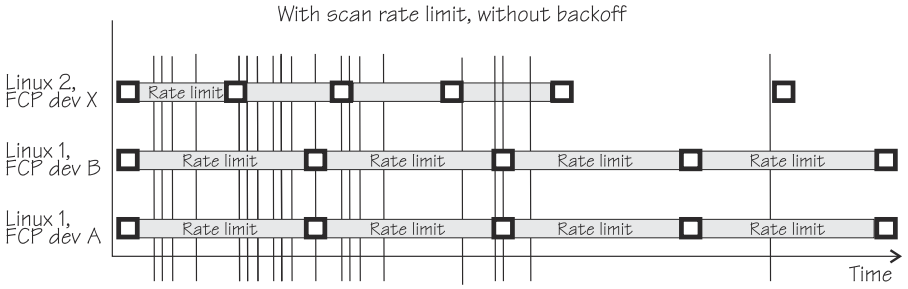
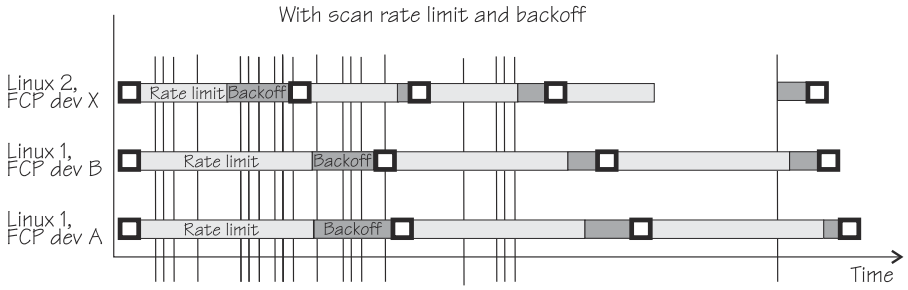
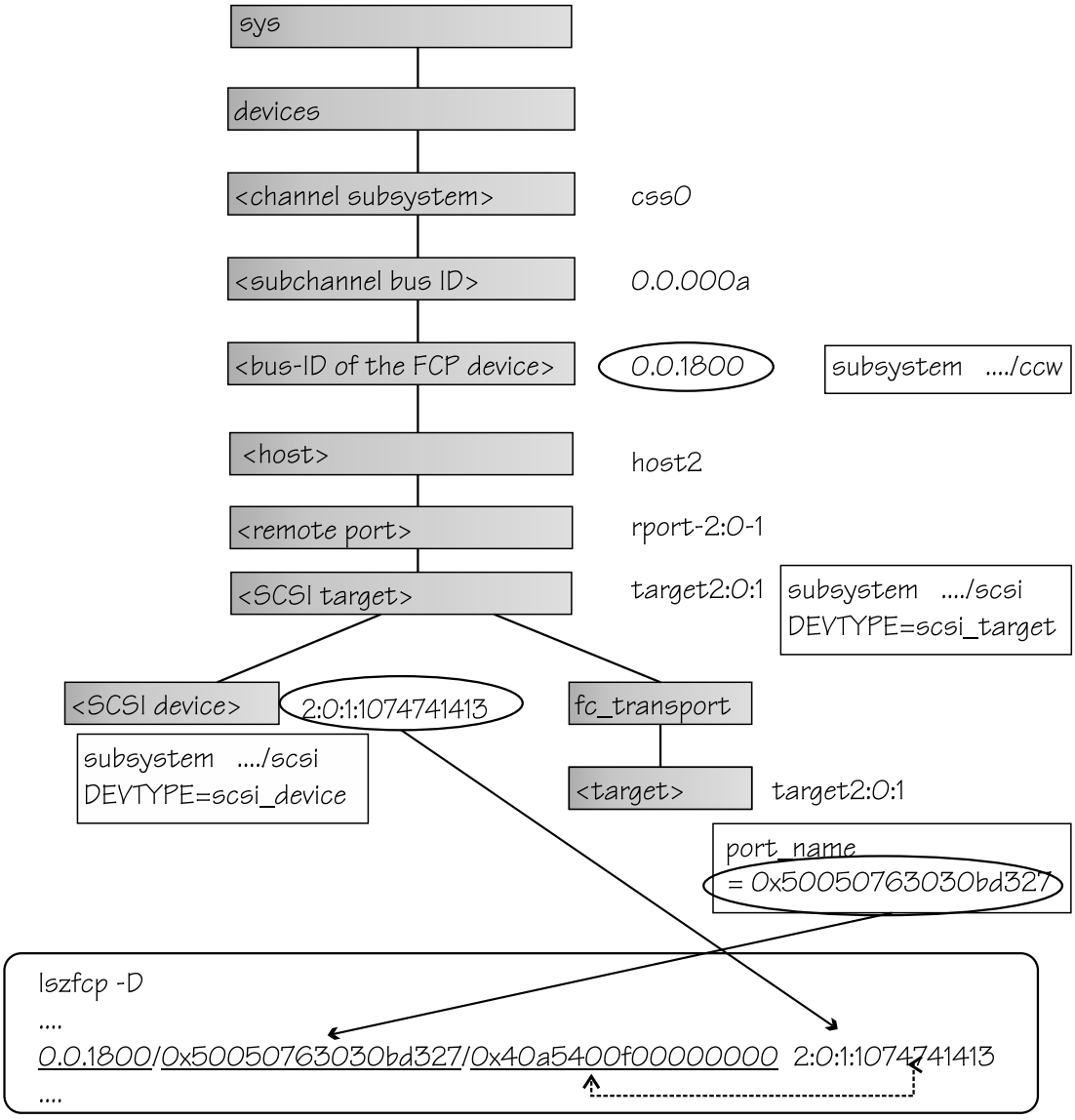
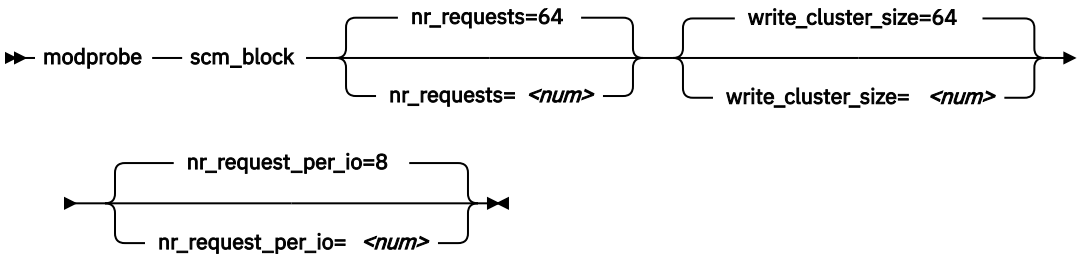
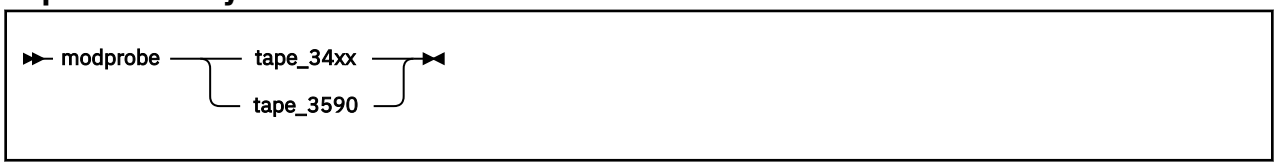
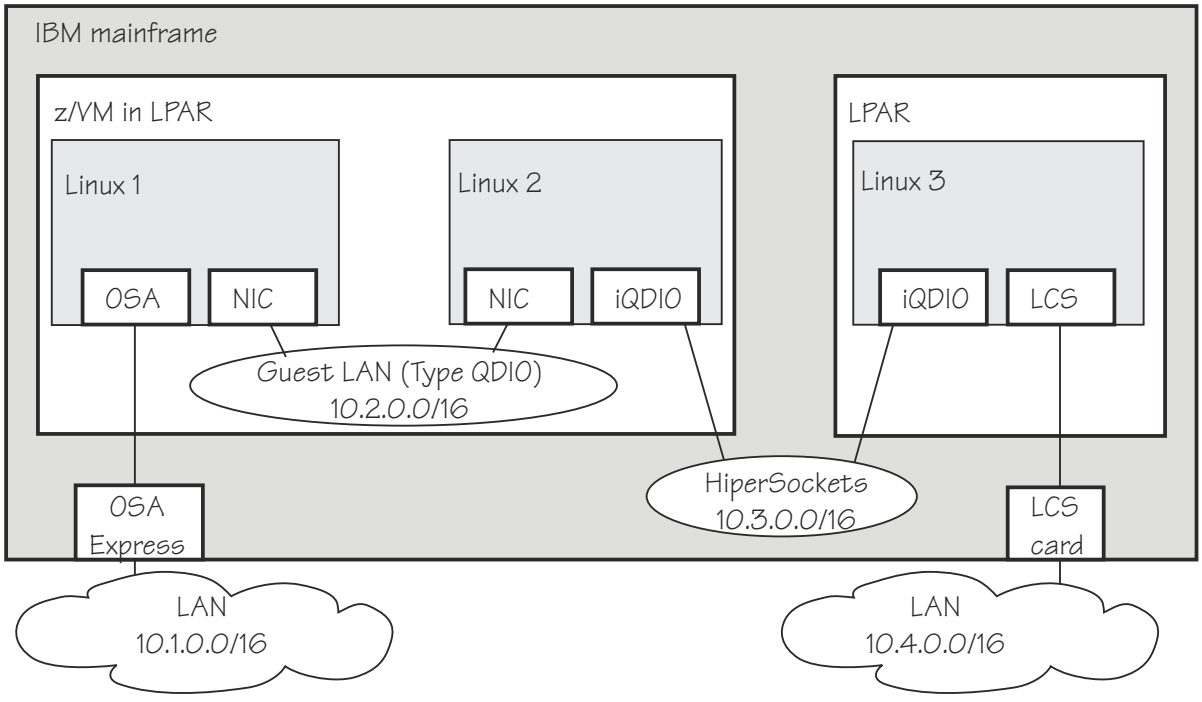
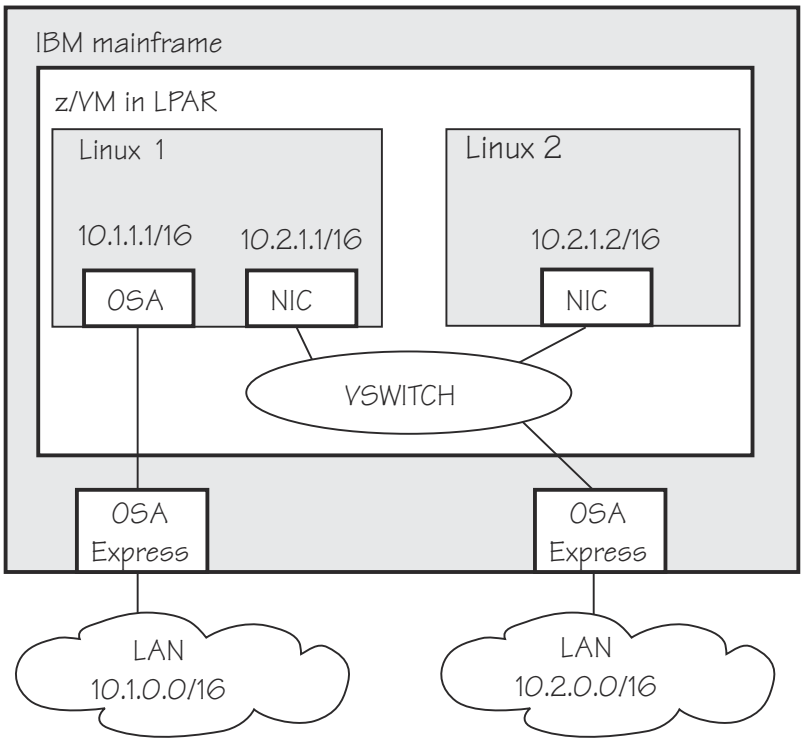
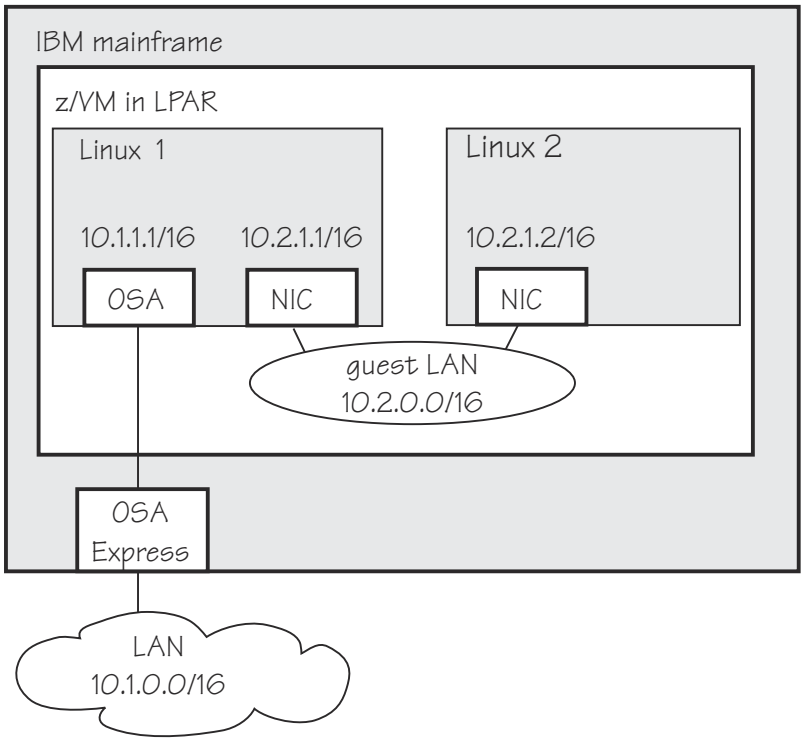
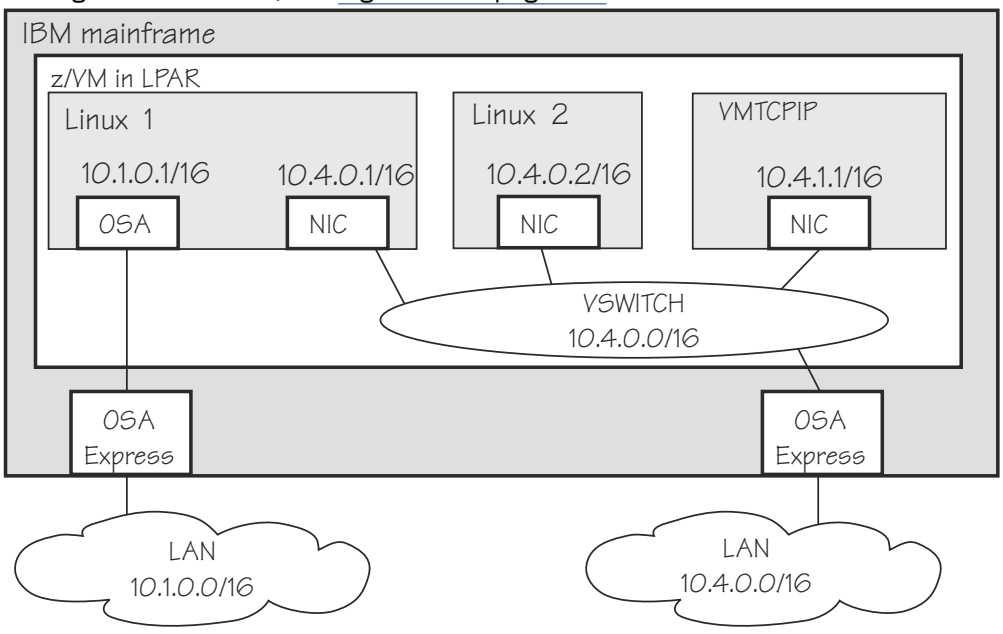
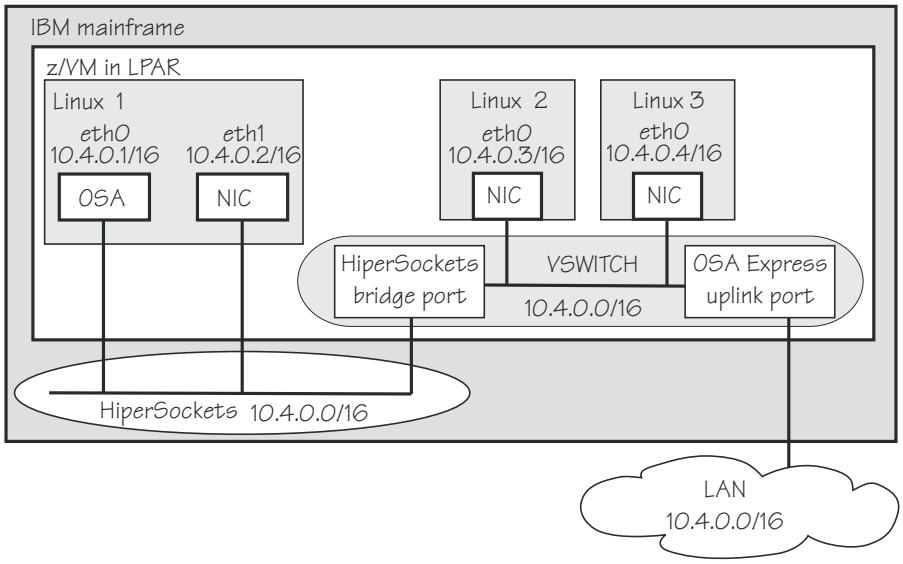
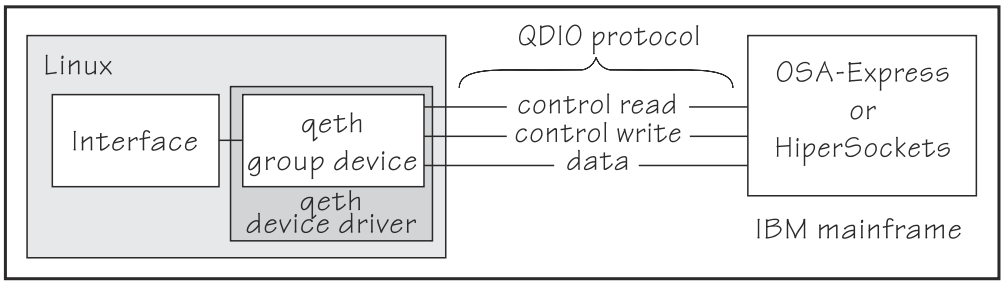
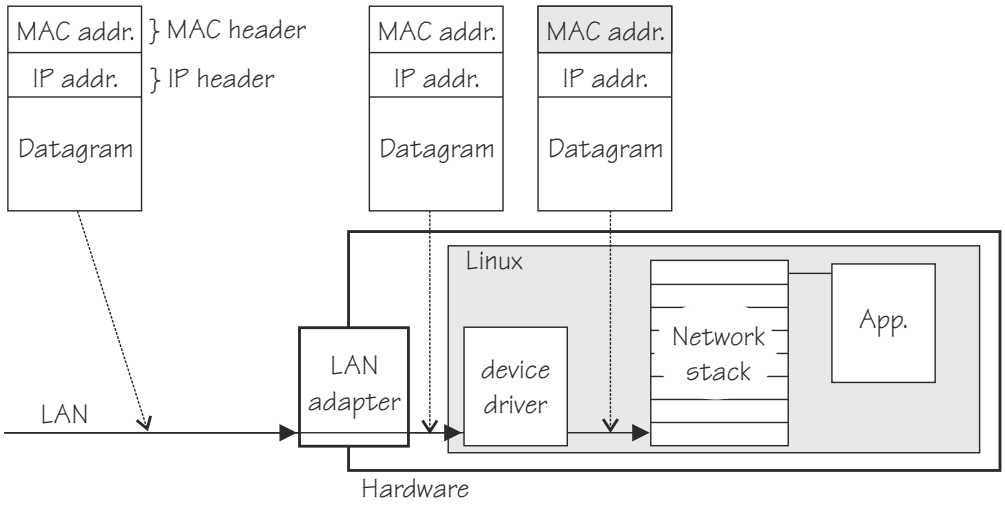
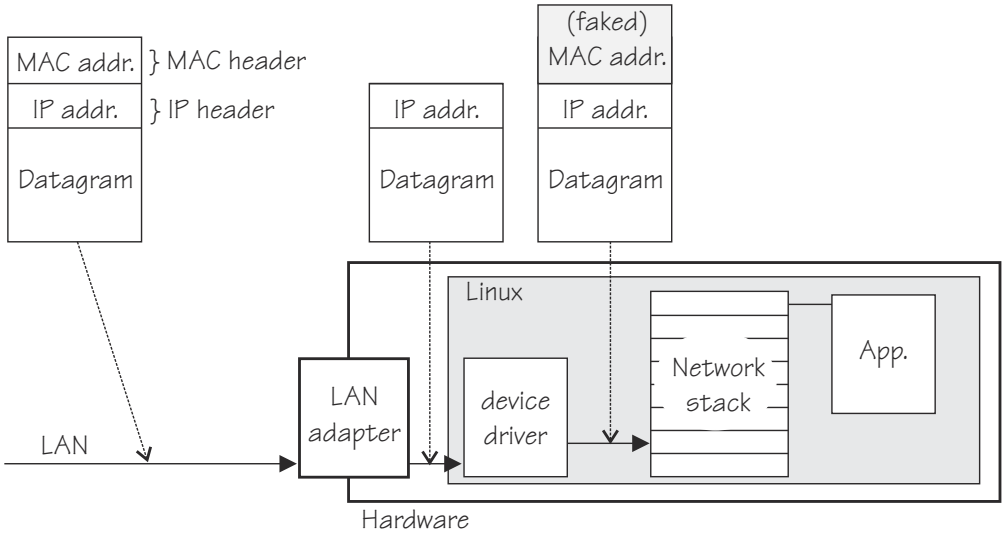
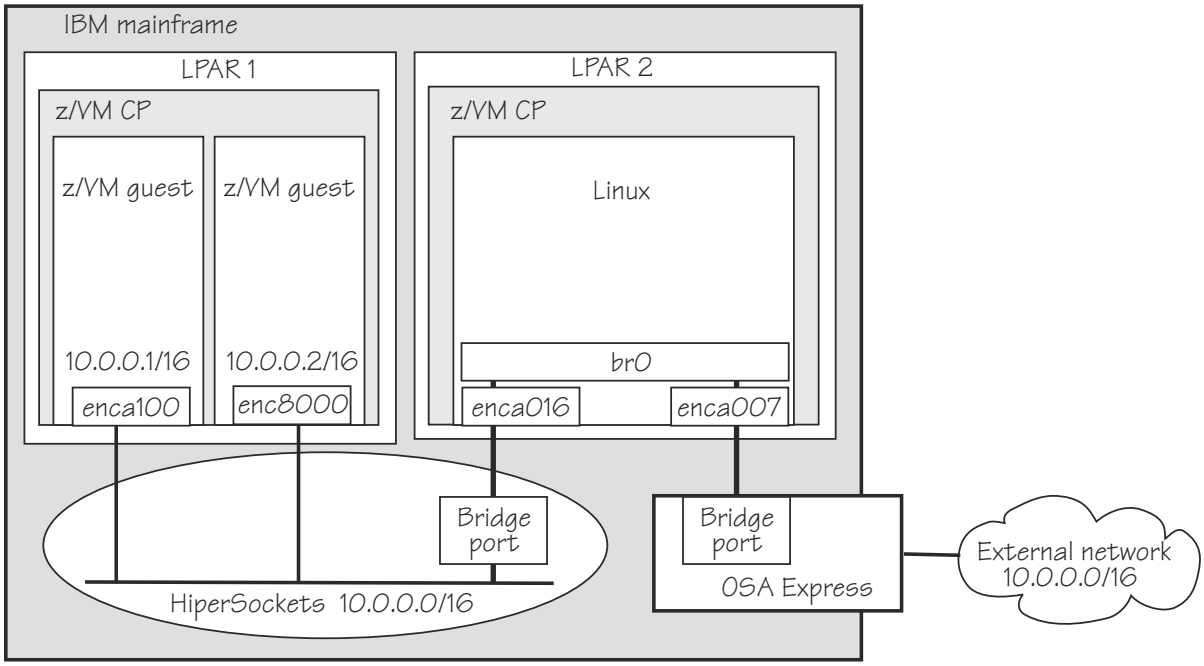
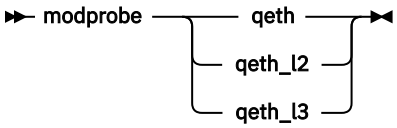
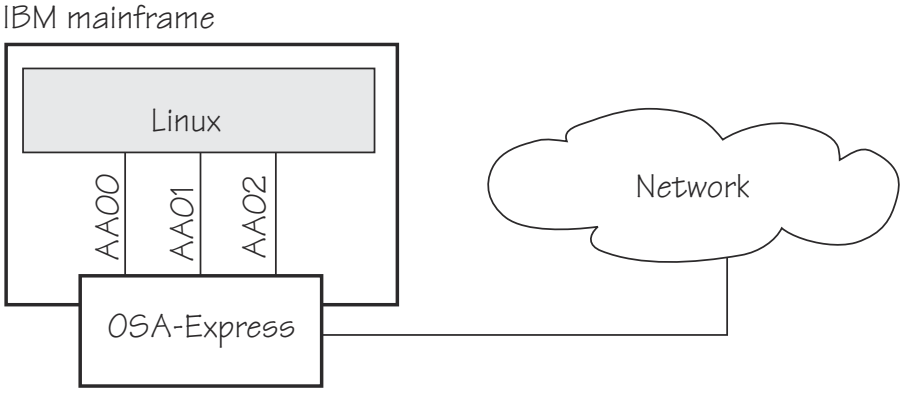
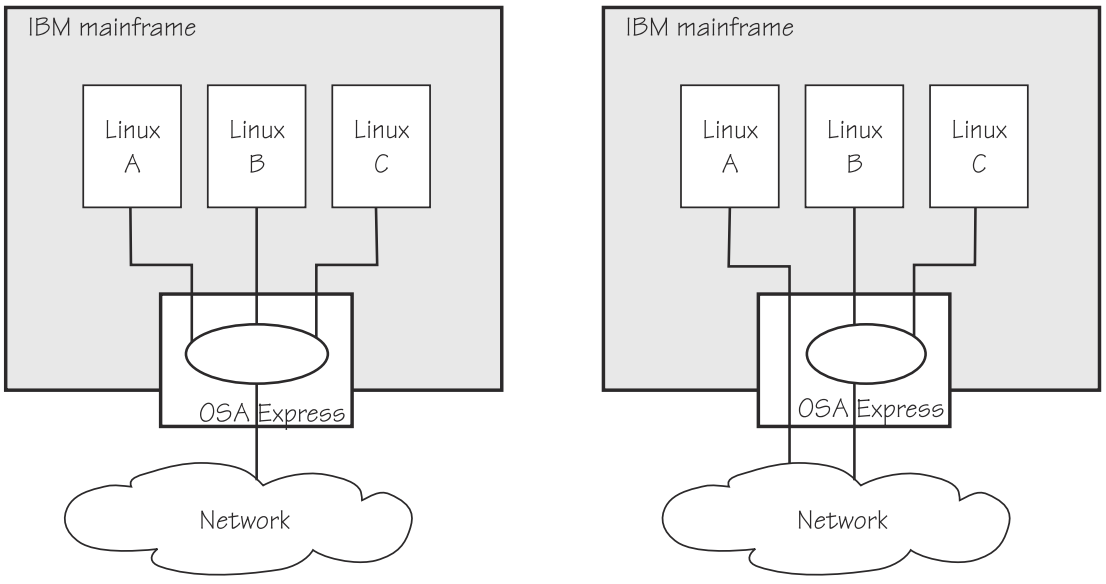
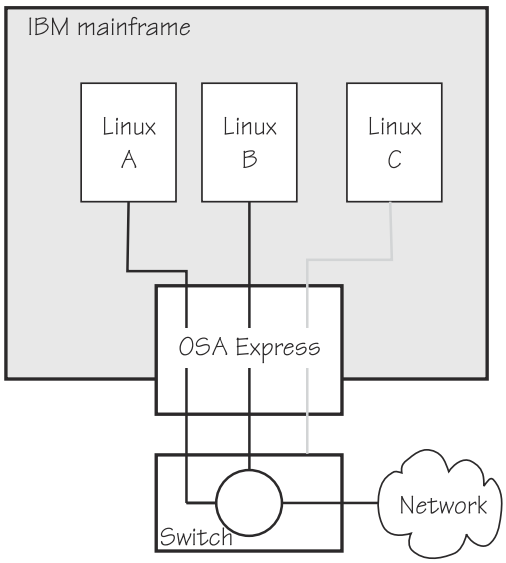
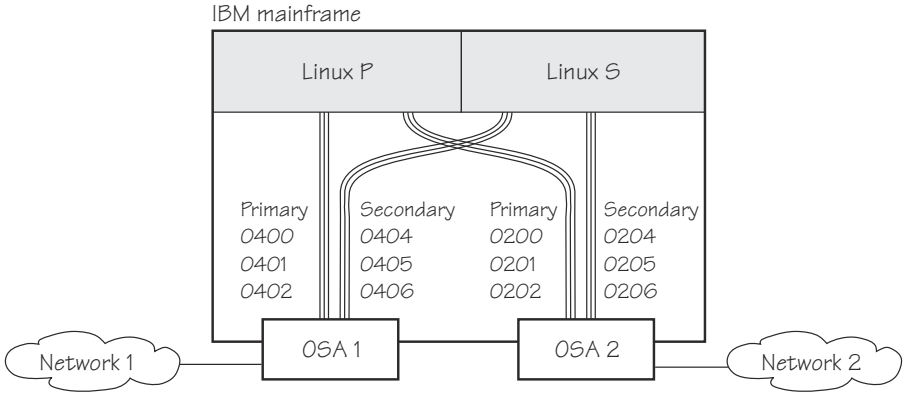
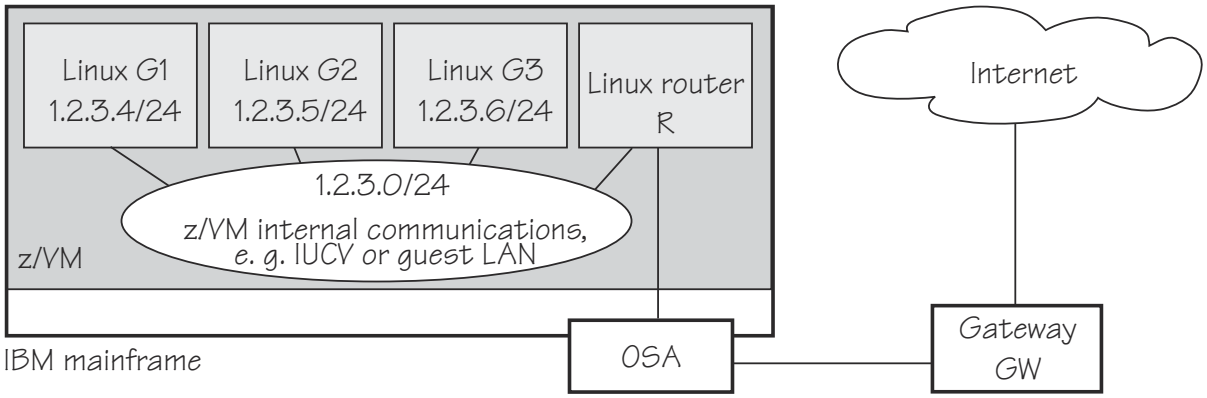
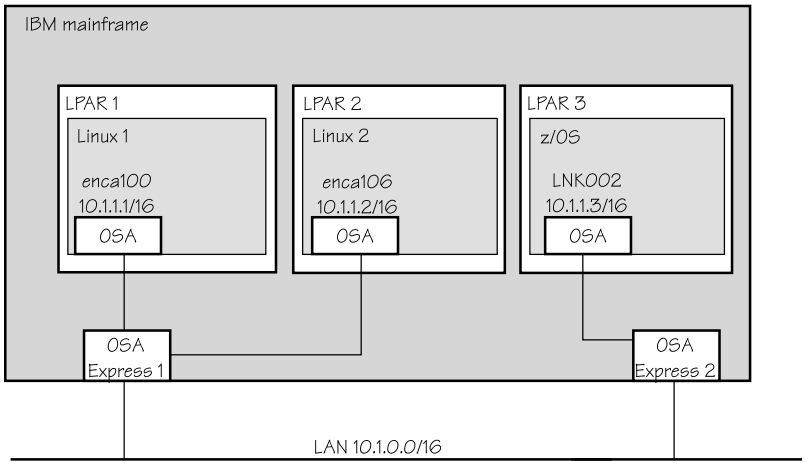
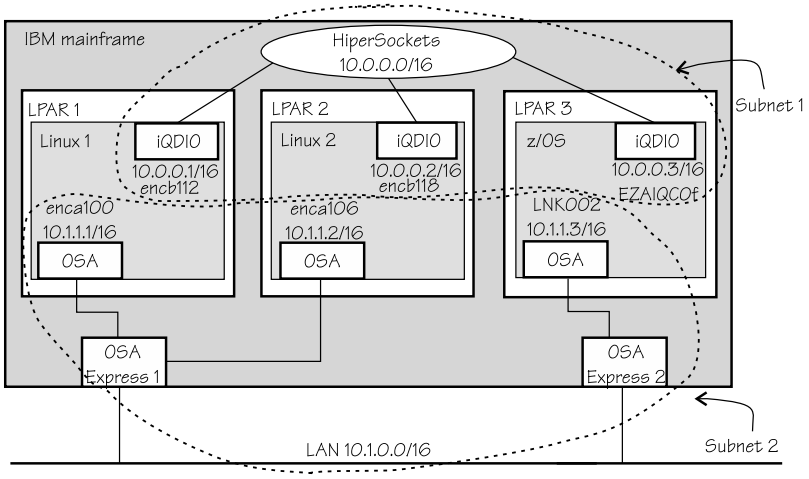
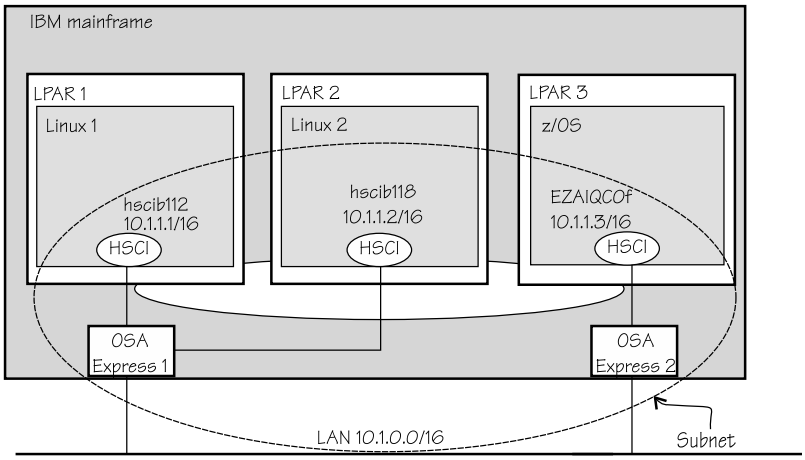
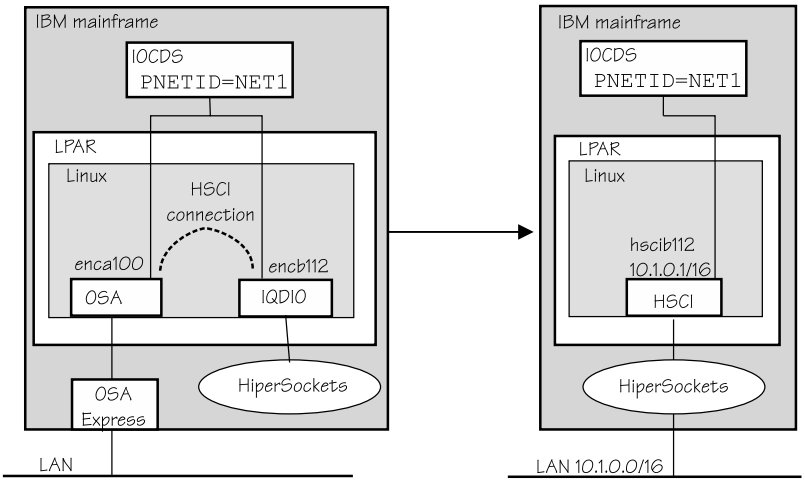
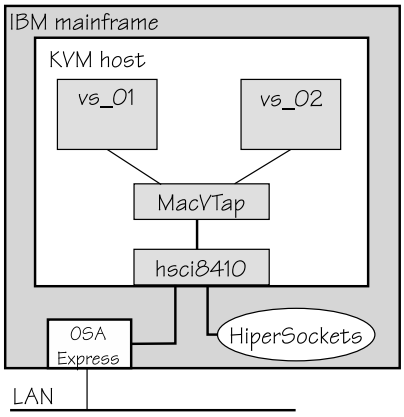
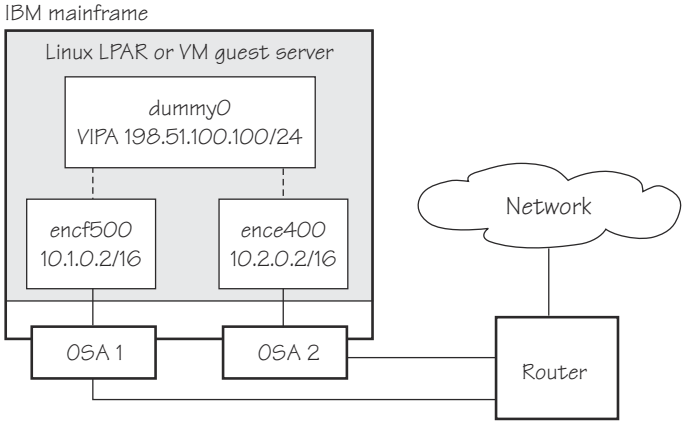
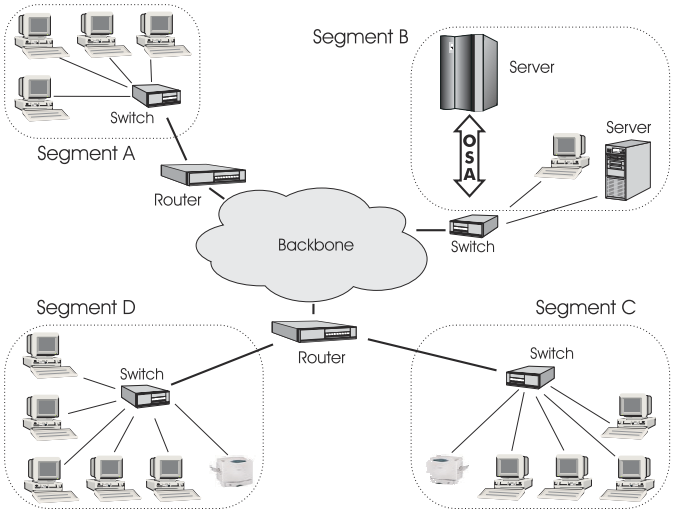
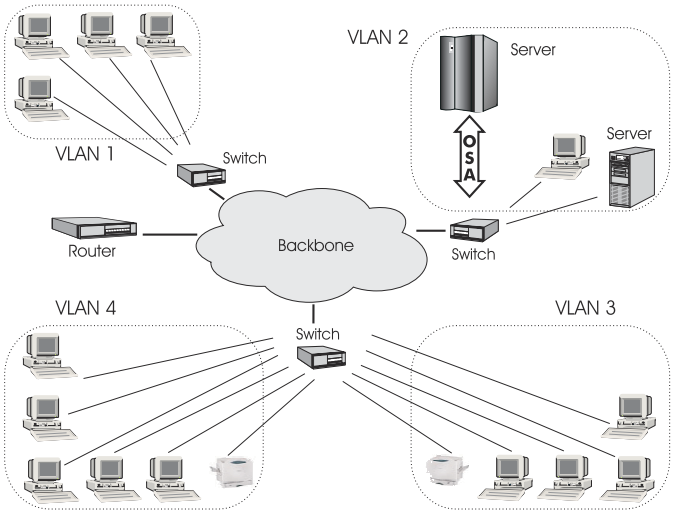
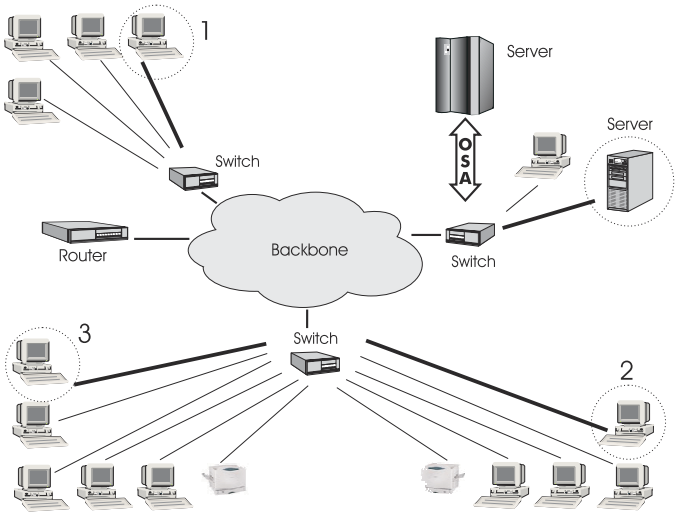
CUU,193 ,VOLSER,LNX001   3390, WITH   100 CYLS, 15 TRKS/CYL, 58786 BYTES/TRK

--- FILE NAME --- (SORTED BY =,NAME  ,) ---- EXT    BEGIN-END     RELTRK,
1...5...10...15...20...25...30...35...40.... SQ   CYL-HD   CYL-HD      NUMTRKS
 *** VTOC EXTENT ***                          0     0  1     0  1      1,1
LINUX.VLNX001.PART0001.NATIVE                 0     0  2    46 11      2,700
LINUX.VLNX001.PART0002.NATIVE                 0    46 12    66 11    702,300
LINUX.VLNX001.PART0003.NATIVE                 0    66 12    99 14   1002,498
 *** THIS VOLUME IS CURRENTLY 100 PER CENT FULL WITH      0 TRACKS AVAILABLE

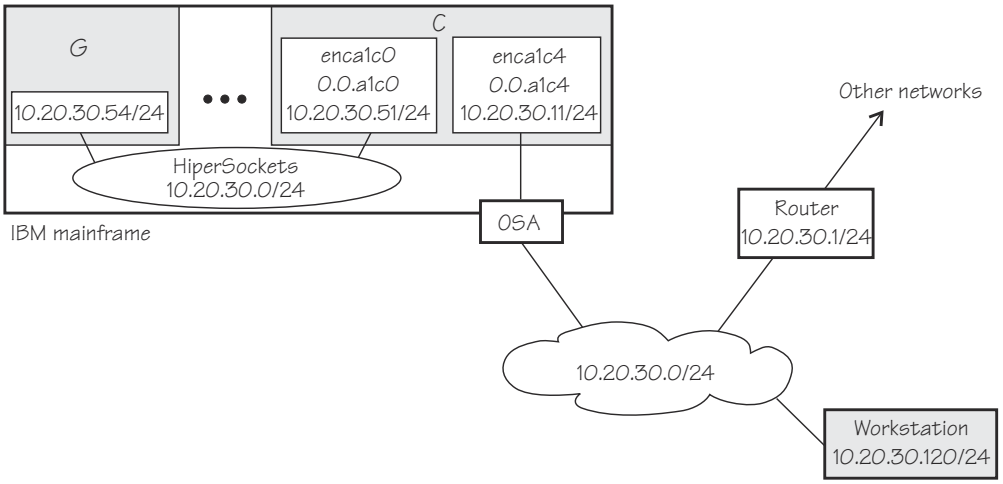
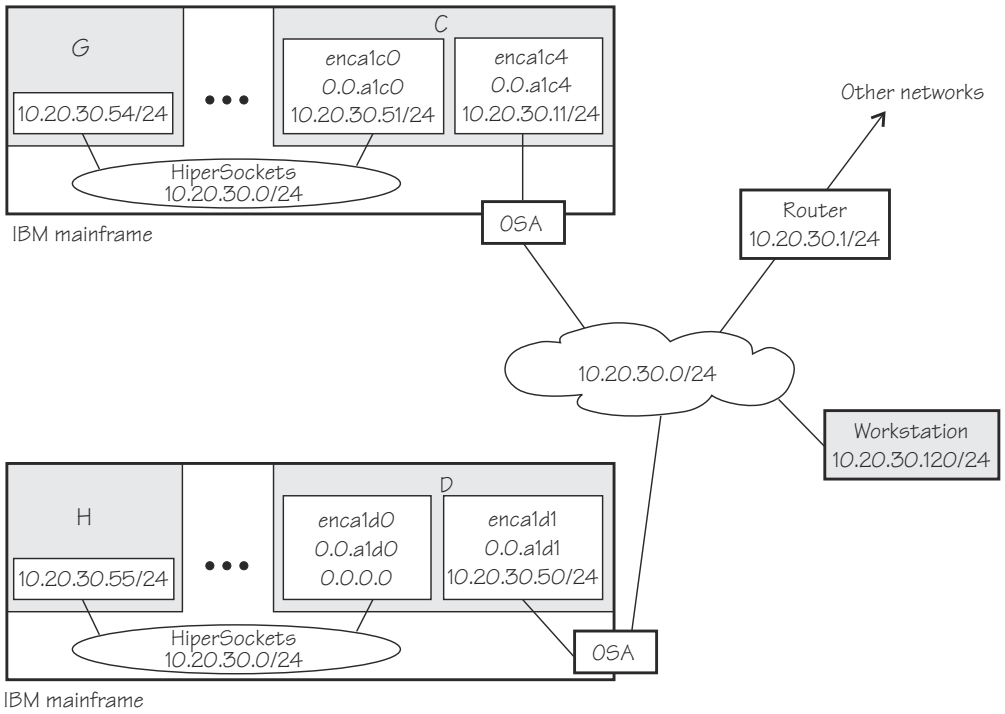
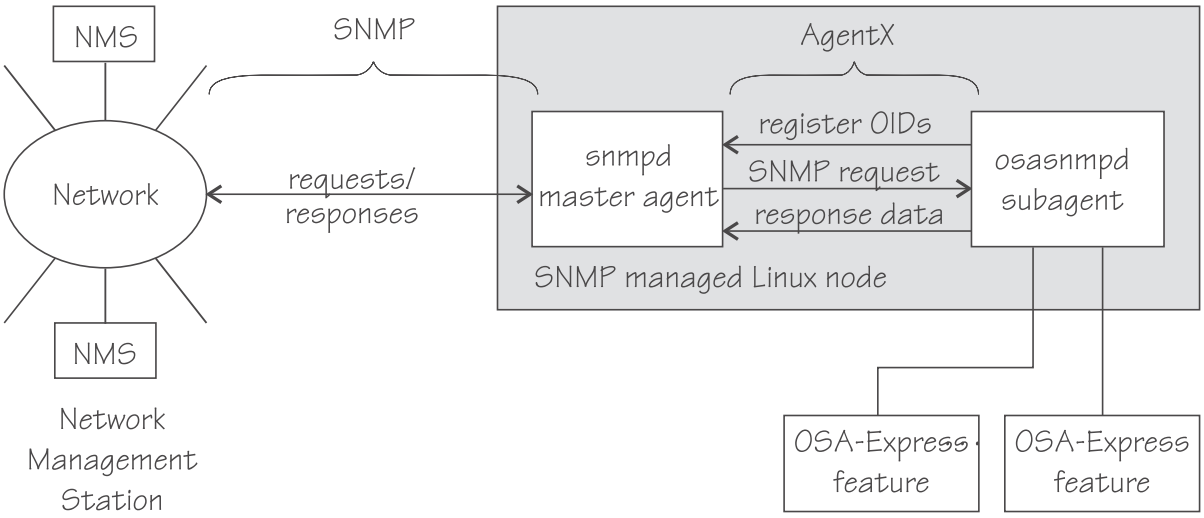
PF  1=HELP      2=TOP       3=END       4=BROWSE    5=BOTTOM    6=LOCATE
PF  7=UP        8=DOWN      9=PRINT    10=RGT/LEFT 11=UPDATE   12=RETRIEVE
```

The **ls** command on Linux might list this DASD and its partitions like this example:

```
# ls -l /dev/dasda*
brw-rw---- 1 root disk 94, 0 Jan 27 09:04 /dev/dasda
brw-rw---- 1 root disk 94, 1 Jan 27 09:04 /dev/dasda1
brw-rw---- 1 root disk 94, 2 Jan 27 09:04 /dev/dasda2
brw-rw---- 1 root disk 94, 3 Jan 27 09:04 /dev/dasda3
```

where dasda represent the whole DASD and dasda1, dasda2, and dasda3 represent the individual partitions.

Linux disk layout

The Linux disk layout does not have a VTOC, and DASD partitions that are formatted with this layout cannot be accessed by traditional mainframe operating systems.

You can format only ECKD type DASD with the Linux disk layout. Apart from accessing the disks as ECKD devices, you can also access them using the DASD DIAG access method. See "Enabling the DASD device driver to use the DIAG access method" on page 152 for how to enable DIAG.

Figure 46 on page 141 illustrates a disk with the Linux disk layout.



Figure 46. Linux disk layout

DASDs with the Linux disk layout either have an LNX1 label or are not labeled. The first records of the device are reserved for IPL records and the volume label, and are not intended for use by Linux applications. All remaining records are grouped into a single partition. You cannot have more than a single partition on a DASD that is formatted in the Linux disk layout.

Linux can address the device as a whole as /dev/dasd<*x*>, where <*x*> can be one to four letters that identify the individual DASD (see "DASD naming scheme" on page 142). Linux can access the partition as /dev/dasd<*x*>1.

You use the **dasdfmt** command (see "dasdfmt - Format a DASD" on page 615) to format a disk with the Linux disk layout.

CMS disk layout

The CMS disk layout applies only to Linux on z/VM. The disks are formatted with z/VM tools.

Both ECKD or FBA type DASD can have the CMS disk layout. DASD partitions that are formatted with this layout cannot be accessed by traditional mainframe operating systems. Apart from accessing the disks as ECKD or FBA devices, you can also access them using the DASD DIAG access method.

Figure 47 on page 142 illustrates two variants of the CMS disk layout.

Figure 47. CMS disk layout

The first variant contains IPL records, a volume label (CMS1), and a CMS data area. Linux treats DASD like this equivalent to a DASD with the Linux disk layout, where the CMS data area serves as the Linux partition.

The second variant is a CMS reserved volume. In this variant, the DASD was reserved by a `CMS RESERVE fn ft fm` command. In addition to the IPL records and the volume label, DASD with the CMS disk layout also have CMS metadata. The CMS reserved file serves as the Linux partition.

For both variants of the CMS disk layout, you can have only a single Linux partition. The IPL record, volume label and (where applicable) the CMS metadata, are not intended for use by Linux applications.

Addressing the device and partition is the same for both variants. Linux can address the device as a whole as `/dev/dasd<x>`, where *<x>* can be one to four letters that identify the individual DASD (see "DASD naming scheme" on page 142). Linux can access the partition as `/dev/dasd<x>1`.

"Enabling the DASD device driver to use the DIAG access method" on page 152 describes how you can enable DIAG.

Disk layout summary

The available disk layouts differ in their support of device formats, the DASD DIAG access method, and the maximum number of partitions.

Table 15. Disk layout summary

Disk layout	ECKD device format	FBA device format	DIAG access method support (z/VM only)	Maximum number of partitions	Formatting tool
Compatible disk layout	Yes	No	No	3	**dasdfmt**
Linux disk layout	Yes	No	Yes	1	**dasdfmt**
CMS (z/VM only)	Yes	Yes	Yes	1	z/VM tools

DASD naming scheme

The DASD naming scheme maps device names and minor numbers to whole DASDs and to partitions.

The DASD device driver uses the major number 94. For each configured device it uses four minor numbers:

- The first minor number always represents the device as a whole, including IPL, VTOC, and label records.
- The remaining three minor numbers represent the up to three partitions.

With 1,048,576 (20-bit) available minor numbers, the DASD device driver can address 262,144 devices.

The DASD device driver uses a device name of the form dasd*<x>* for each DASD. In the name, *<x>* is one to four lowercase letters. Table 16 on page 143 shows how the device names map to the available minor numbers.

Table 16. Mapping of DASD names to minor numbers

Name for device as a whole		**Minor number for device as a whole**		**Number of devices**
From	**To**	**From**	**To**	
dasda	dasdz	0	100	26
dasdaa	dasdzz	104	2804	676
dasdaaa	dasdzzz	2808	73108	17,576
dasdaaaa	dasdnwtl	73112	1048572	243,866
Total number of devices:				262,144

The DASD device driver also uses a device name for each partition. The name of the partition is the name of the device as a whole with a 1, 2, or 3 appended to identify the first, second, or third partition. The three minor numbers that follow the minor number of the device as a whole are the minor number for the first, second, and third partition.

Examples

- "dasda" refers to the whole of the first disk in the system and "dasda1", "dasda2", and "dasda3" to the three partitions. The minor number for the whole device is 0. The minor numbers of the partitions are 1, 2, and 3.
- "dasdz" refers to the whole of the 101st disk in the system and "dasdz1", "dasdz2", and "dasdz3" to the three partitions. The minor number for the whole device is 100. The minor numbers of the partitions are 101, 102, and 103.
- "dasdaa" refers to the whole of the 102nd disk in the system and "dasdaa1", "dasdaa2", and "dasdaa3" to the three partitions. The minor number for the whole device is 104. The minor numbers of the partitions are 105, 106, and 107.

DASD device nodes

Red Hat Enterprise Linux 9.1 uses udev to create multiple device nodes for each DASD that is online.

Device nodes based on device names
udev creates device nodes that match the device names used by the kernel. These standard device nodes have the form `/dev/<name>`.

The mapping between standard device nodes and the associated physical disk space can change, for example, when you reboot Linux. To ensure that you access the intended physical disk space, you need device nodes that are based on properties that identify a particular DASD.

udev creates additional devices nodes that are based on the following information:

- The bus ID of the disk
- The disk label (VOLSER)
- The universally unique identifier (UUID) of the file system on the disk
- If available: The label of the file system on the disk

Device nodes based on bus IDs
udev creates device nodes of the form

```
/dev/disk/by-path/ccw-<device_bus_id>
```

for whole DASD and

```
/dev/disk/by-path/ccw-<device_bus_id>-part<n>
```

for the <*n*>th partition.

Device nodes that are based on VOLSERs

udev creates device nodes of the form

```
/dev/disk/by-id/ccw-<volser>
```

for whole DASD and

```
/dev/disk/by-id/ccw-<volser>-part<n>
```

for the <*n*>th partition.

If you want to use device nodes based on VOLSER, be sure that the VOLSERs in your environment are unique (see “Volume label” on page 139).

If you assign the same VOLSER to multiple devices, Linux can still access each device through its standard device node. However, only one of the devices can be accessed through the VOLSER-based device node. Thus, the node is ambiguous and might lead to unintentional data access.

Furthermore, if the VOLSER on the device that is addressed by the node is changed, the previously hidden device is not automatically addressed instead. To reassign the node, you must reboot Linux or force the kernel to reread the partition tables from disks, for example, by issuing:

```
# blockdev --rereadpt /dev/dasdzzz
```

You can assign VOLSERs to ECKD type devices with **dasdfmt** when formatting or later with **fdasd** when creating partitions.

Device nodes based on file system information

udev creates device nodes of the form

```
/dev/disk/by-uuid/<uuid>
```

where <*uuid*> is the UUID for the file system in a partition.

If a file system label exists, udev also creates a node of the form

```
/dev/disk/by-label/<label>
```

There are no device nodes for the whole DASD that are based on file system information.

If you want to use device nodes that are based on file system labels, be sure that the labels in your environment are unique.

Additional device nodes

`/dev/disk/by-id` contains additional device nodes for the DASD and partitions, that are all based on a device identifier as contained in the `uid` attribute of the DASD.

Note: If you want to use device nodes that are based on file system information and VOLSER, be sure that they are unique for the scope of your Linux instance. This information can be changed by a user or it can be copied, for example when backup disks are created. If two disks with the same VOLSER or UUID are online to the same Linux instance, the matching device node can point to either of these disks.

Example

For a DASD that is assigned the device name dasdzzz, has two partitions, a device bus-ID 0.0.b100 (device number 0xb100), VOLSER LNX001, and a UUID 6dd6c43d-a792-412f-a651-0031e631caed for the first and f45e955d-741a-4cf3-86b1-380ee5177ac3 for the second partition, udev creates the following device nodes:

For the whole DASD:

- `/dev/dasdzzz` (standard device node according to the DASD naming scheme)
- `/dev/disk/by-path/ccw-0.0.b100`
- `/dev/disk/by-id/ccw-LNX001`

For the first partition:

- `/dev/dasdzzz1` (standard device node according to the DASD naming scheme)
- `/dev/disk/by-path/ccw-0.0.b100-part1`
- `/dev/disk/by-id/ccw-LNX001-part1`
- `/dev/disk/by-uuid/6dd6c43d-a792-412f-a651-0031e631caed`

For the second partition:

- `/dev/dasdzzz2` (standard device node according to the DASD naming scheme)
- `/dev/disk/by-path/ccw-0.0.b100-part2`
- `/dev/disk/by-id/ccw-LNX001-part2`
- `/dev/disk/by-uuid/f45e955d-741a-4cf3-86b1-380ee5177ac3`

Accessing DASD by udev-created device nodes

Use udev-created device nodes to access a particular physical disk space, regardless of the device name that is assigned to it.

Example

The following example is based on these assumptions:

- A DASD with bus ID 0.0.b100 has two partitions.
- The standard device node of the DASD is `dasdzzz`.
- udev creates the following device nodes for a DASD and its partitions:

```
/dev/disk/by-path/ccw-0.0.b100
/dev/disk/by-path/ccw-0.0.b100-part1
/dev/disk/by-path/ccw-0.0.b100-part2
```

Instead of issuing:

```
# fdasd /dev/dasdzzz
```

issue:

```
# fdasd /dev/disk/by-path/ccw-0.0.b100
```

In the file system information in `/etc/fstab` replace the following specifications:

```
/dev/dasdzzz1 /temp1 ext4 defaults 0 0
/dev/dasdzzz2 /temp2 ext4 defaults 0 0
```

with these specifications:

```
/dev/disk/by-path/ccw-0.0.b100-part1 /temp1 ext4 defaults 0 0
/dev/disk/by-path/ccw-0.0.b100-part2 /temp2 ext4 defaults 0 0
```

You can make similar substitutions with other device nodes that udev provides for you (see “DASD device nodes” on page 143).

Setting up the DASD device driver

Unless the DASD device driver modules are loaded for you during the boot process, load and configure them with the **modprobe** command.

DASD module parameter syntax



device-spec



Where:

dasd_mod
loads the device driver base module.

When you are loading the base module you can specify the `dasd=` parameter.

You can use the `eer_pages` parameter to determine the number of pages that are used for internal buffering of error records.

autodetect
causes the DASD device driver to allocate device names and the corresponding minor numbers to all DASD devices and set them online during the boot process. See “DASD naming scheme” on page 142 for the naming scheme.

The device names are assigned in order of ascending subchannel numbers. Auto-detection can yield confusing results if you change your I/O configuration and reboot, or if your Linux instance runs as a z/VM guest because the devices might appear with different names and minor numbers after rebooting.

probeonly
causes the DASD device driver to reject any "open" syscall with EPERM.

autodetect,probeonly
causes the DASD device driver to assign device names and minor numbers as for auto-detect. All devices regardless of whether they are accessible as DASD return EPERM to any "open" requests.

nopav
suppresses parallel access volume (PAV and HyperPAV) enablement for Linux instances that run in LPAR mode. The **nopav** keyword has no effect for Linux on z/VM.

nofcx
suppresses accessing the storage server with the I/O subsystem in transport mode (also known as High Performance FICON).

<device_bus_id>
specifies a single DASD.

<from_device_bus_id>-<to_device_bus_id>
specifies the first and last DASD in a range. All DASD devices with bus IDs in the range are selected. The device bus-IDs *<from_device_bus_id>* and *<to_device_bus_id>* need not correspond to actual DASD.

ipldev
for IPL from a DASD, specifies the IPL device. If the IPL device is not a DASD, this parameter is ignored.

(ro)
accesses the specified device or device range in read-only mode.

(diag)
forces the device driver to access the device (range) with the DIAG access method.

(erplog)
enables enhanced error recovery processing (ERP) related logging through syslogd. If **erplog** is specified for a range of devices, the logging is switched on during device initialization.

(failfast)
immediately returns "failed" for an I/O operation when the last path to a DASD is lost.

Attention: Enable immediate failure of I/O requests only in setups where a failed I/O request can be recovered outside the scope of a single DASD (see “Enabling and disabling immediate failure of I/O requests” on page 156).

dasd_eckd_mod
loads the ECKD module.

dasd_fba_mod
loads the FBA module.

dasd_diag_mod
loads the DIAG module.

If you supply a DASD module parameter with device specifications dasd=*<device-list1>,<device-list2> ...*, the device names and minor numbers are assigned in the order in which the devices are specified. The names and corresponding minor numbers are always assigned, even if the device is not present, or not accessible. For information about including device specifications in a boot configuration, see “Including parameters for modules in a boot configuration” on page 30.

If you use **autodetect** in addition to explicit device specifications, device names are assigned to the specified devices first and device-specific parameters, like **ro**, are honored. The remaining devices are handled as described for **autodetect**.

The DASD base component is required by the other modules. **modprobe** takes care of this dependency for you and ensures that the base module is loaded automatically, if necessary.

Hint: modprobe might return before udev has created all device nodes for the specified DASDs. If you need to assure that all nodes are present, for example in scripts, follow the **modprobe** command with:

```
# udevadm settle
```

For command details see the **modprobe** man page.

Example

The following example specifies a range of DASD devices and two individual DASD devices:

```
modprobe dasd_mod dasd=0.0.7000-0.0.7002,0.0.7005(ro),0.0.7006
```

Table 17 on page 148 shows the resulting allocation of device names:

Table 17. Example mapping of device names to devices

Name	To access
dasda	device 0.0.7000 as a whole
dasda1	the first partition on 0.0.7000
dasda2	the second partition on 0.0.7000
dasda3	the third partition on 0.0.7000
dasdb	device 0.0.7001 as a whole
dasdb1	the first partition on 0.0.7001
dasdb2	the second partition on 0.0.7001
dasdb3	the third partition on 0.0.7001
dasdc	device 0.0.7002 as a whole
dasdc1	the first partition on 0.0.7002
dasdc2	the second partition on 0.0.7002
dasdc3	the third partition on 0.0.7002
dasdd	device 0.0.7005 as a whole
dasdd1	the first partition on 0.0.7005 (read-only)
dasdd2	the second partition on 0.0.7005 (read-only)
dasdd3	the third partition on 0.0.7005 (read-only)
dasde	device 0.0.7006 as a whole
dasde1	the first partition on 0.0.7006
dasde2	the second partition on 0.0.7006
dasde3	the third partition on 0.0.7006

The following example specifies that High Performance FICON are to be suppressed for all DASDs:

```
modprobe dasd_mod dasd=nofcx,4711-4713
```

Working with DASDs

You might have to prepare DASD for use, configure troubleshooting functions, or configure special device features for your DASDs.

Most of the following tasks involve writing to and reading from device attributes in sysfs. This method is useful on a running system where you want to make dynamic changes. For changes that persist across IPLs, use the configuration file `/etc/zipl.conf` for DASDs that are part of the root file system and `/etc/dasd.conf` for data disks. An example of how to define a DASD device persistently is

included in the installation documentation on the Red Hat documentation website. A general discussion of configuration files can also be found there, see https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux.

See “Working with newly available devices” on page 10 to avoid errors when you are working with devices that have become available to a running Linux instance.

Preparing an ECKD type DASD for use

Before you can use an ECKD type DASD as a disk for Linux on IBM Z, you must format it with a suitable disk layout. You must then create a file system or define a swap space.

Before you begin

- The modules for the base component and the ECKD component of the DASD device driver must have been loaded.
- The DASD device driver must have recognized the device as an ECKD type device.
- You must know the device bus-ID for your DASD.

About this task

If you format the DASD with the compatible disk layout, you must create one, two, or three partitions. You can then use your partitions as swap areas or to create a Linux file system.

Procedure

Perform these steps to prepare the DASD:

1. Issue **lsdasd** (see “lsdasd - List DASD devices” on page 661) to find out if the device is online.

   If necessary, set the device online using **chccwdev** (see “chccwdev - Set CCW device attributes” on page 575).

   Example:

```
# chccwdev -e 0.0.b100
```

2. Format the device with the **dasdfmt** command (see “dasdfmt - Format a DASD” on page 615 for details). The formatting process can take hours for large DASDs.

   If you want to use the CMS disk layout, and your DASD is already formatted with the CMS disk layout, skip this step.

   Tips:

   - Use the largest possible block size, ideally 4096; the net capacity of an ECKD DASD decreases for smaller block sizes. For example, a DASD formatted with a block size of 512 byte has only half of the net capacity of the same DASD formatted with a block size of 4096 byte.
   - For DASDs that have previously been formatted with the `cdl` or `ldl` disk layout, use the **dasdfmt** quick format mode.
   - Use the **-p** option to display a progress bar.

   Example: Assuming that `/dev/dasdzzz` is a valid device node for 0.0.b100:

```
# dasdfmt -b 4096 -p  /dev/dasdzzz
```

3. Proceed according to your chosen disk layout:

   - If you have formatted your DASD with the Linux disk layout or the CMS disk layout, skip this step and continue with step “4” on page 150. You already have one partition and cannot add further partitions on your DASD.
   - If you have formatted your DASD with the compatible disk layout use the **fdasd** command to create up to three partitions (see “fdasd – Partition a DASD” on page 632 for details).

     Example: To start the partitioning tool in interactive mode for partitioning a device `/dev/dasdzzz` issue:

```
# fdasd /dev/dasdzzz
```

     If you create three partitions for a DASD `/dev/dasdzzz`, the device nodes for the partitions are `/dev/dasdzzz1`, `/dev/dasdzzz2`, and `/dev/dasdzzz3`.

     Result: **fdasd** creates the partitions and updates the partition table (see “VTOC” on page 140).

4. Depending on the intended use of each partition, create a file system on the partition or define it as a swap space.

   - Either, create a file system. For example, use the Linux **mkfs.ext4** command to create an ext4 file system (see the man page for details).

     Note: Do not make the block size of the file system smaller than the block size that was used for formatting the disk with the **dasdfmt** command.

     Example:

```
# mkfs.ext4 -b 4096 /dev/dasdzzz1
```

   - Or define the partition as a swap space with the **mkswap** command (see the man page for details).

5. Mount each file system to the mount point of your choice in Linux and enable your swap partitions.

   Example: To mount a file system in a partition `/dev/dasdzzz1` to a mount point `/mnt` and to enable a swap partition `/dev/dasdzzz2` issue:

```
# mount /dev/dasdzzz1 /mnt
# swapon /dev/dasdzzz2
```

Preparing an FBA-type DASD for use

Before you can use an FBA-type DASD as a disk for Linux on IBM Z, you must create a file system or define a swap space.

Before you begin

- The modules for the base component and the FBA component of the DASD device driver must have been loaded.
- The DASD device driver must have recognized the device as an FBA device.
- You must know the device bus-ID or the device node through which the DASD can be addressed.

Procedure

Perform these steps to prepare the DASD:

1. Depending on the intended use of the partition, create a file system on it or define it as a swap space.
   - Either create a file system, for example, with the Linux **mke2fs** command (see the man page for details).

     Example:

     ```
     # mke2fs -b 4096 /dev/dasdzzy1
     ```

   - Or define the partition as a swap space with the **mkswap** command (see the man page for details).
2. Mount the file system to the mount point of your choice in Linux or enable your swap partition.

   Tip: Mount file systems on FBA devices that are backed by z/VM VDISKs with the `discard` mount option. This option frees memory when data is deleted from the device.

   Examples:

   - To mount a file system in a partition `/dev/dasdzzy1`, issue:

     ```
     # mount /dev/dasdzzy1 /mnt
     ```

   - To mount a VDISK-backed file system in a partition `/dev/dasdzzx1`, and use the `discard` option to free memory when data is deleted, issue:

     ```
     # mount -o discard /dev/dasdzzx1 /mnt
     ```

What to do next

To access FBA devices, use the DIAG access method (see “Enabling the DASD device driver to use the DIAG access method” on page 152 for more information).

Accessing DASD by force

A Linux instance can encounter DASDs that are locked by another system. Such a DASD is referred to as "externally locked" or "boxed". The Linux instance cannot analyze a DASD while it is externally locked.

About this task

To check whether a DASD has been externally locked, read its availability attribute. This attribute should be "good". If it is "boxed", the DASD has been externally locked. Because a boxed DASD might not be recognized as DASD, it might not show up in the device driver view in sysfs. If necessary, use the device category view instead (see “Device views in sysfs” on page 11).

CAUTION: Breaking an external lock can have unpredictable effects on the system that holds the lock.

Procedure

1. Optional: To read the availability attribute of a DASD, issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/availability
```

   Example: This example shows that a DASD with device bus-ID 0.0.b110 (device number 0xb110) has been externally locked.

```
# cat /sys/bus/ccw/devices/0.0.b110/availability
boxed
```

   If the DASD is an ECKD type DASD and if you know the device bus-ID, you can break the external lock and set the device online. This means that the lock of the external system is broken with the "unconditional reserve" channel command.
2. To force a boxed DASD online, write `force` to the online device attribute. Issue a command of this form:

```
# echo force > /sys/bus/ccw/devices/<device_bus_id>/online
```

   Example: To force a DASD with device number 0xb110 online issue:

```
# echo force > /sys/bus/ccw/devices/0.0.b110/online
```

Results

If the external lock is successfully broken or if the lock has been surrendered by the time the command is processed, the device is analyzed and set online. If it is not possible to break the external lock (for example, because of a timeout, or because it is an FBA-type DASD), the device remains in the boxed state. This command might take some time to complete.

For information about breaking the lock of a DASD that has already been analyzed see “tunedasd - Adjust low-level DASD settings” on page 741.

Enabling the DASD device driver to use the DIAG access method

Linux on z/VM can use the DIAG access method to access DASDs with the help of z/VM functions.

Before you begin

This section applies only to Linux instances and DASD for which all of the following conditions are true:

- The Linux instance runs as a z/VM guest.
- The device can be of type ECKD with either LDL or CMS disk layout, or it can be a device of type FBA.
- The module for the DIAG component must be loaded.
- The module for the component that corresponds to the DASD type (dasd_eckd_mod or dasd_fba_mod) must be loaded.
- The DASD is offline.
- The DASD does not represent a parallel access volume alias device.

About this task

You can use the DIAG access method to access both ECKD and FBA-type DASD. You use the device's use_diag sysfs attribute to enable or switch off the DIAG access method in a system that is online. Set the `use_diag` attribute to `1` to enable the DIAG access method. Set the `use_diag` attribute to `0` to switch off the DIAG access method (this is the default).

Alternatively, you can specify `diag` on the command line, for example during IPL, to force the device driver to access the device (range) with the DIAG access method.

Procedure

Issue a command of this form:

```
# echo <flag> > /sys/bus/ccw/devices/<device_bus_id>/use_diag
```

where *<device_bus_id>* identifies the DASD.

If the DIAG access method is not available and you set the `use_diag` attribute to 1, you cannot set the device online (see "Setting a DASD online or offline" on page 154).

Note: When switching between an enabled and a disabled DIAG access method on FBA-type DASD, first reinitialize the DASD, for example, with CMS format or by overwriting any previous content. Switching without initialization might cause data-integrity problems.

For more details about DIAG, see *z/VM: CP Programming Services*, SC24-6272.

Example

In this example, the DIAG access method is enabled for a DASD with device number 0xb100.

1. Ensure that the driver is loaded:

   ```
   # modprobe dasd_diag_mod
   ```

2. Identify the sysfs CCW-device directory for the device in question and change to that directory:

   ```
   # cd /sys/bus/ccw/devices/0.0.b100/
   ```

3. Ensure that the device is offline:

   ```
   # echo 0 > online
   ```

4. Enable the DIAG access method for this device by writing '1' to the use_diag sysfs attribute:

   ```
   # echo 1 > use_diag
   ```

5. Use the online attribute to set the device online:

   ```
   # echo 1 > online
   ```

Using extended error reporting for ECKD type DASD

Control the extended error reporting feature for individual ECKD type DASD through the `eer_enabled` sysfs attribute. Use the character device of the extended error reporting module to obtain error records.

Before you begin

To use the extended error reporting feature, you need ECKD type DASD

About this task

The extended error reporting feature is turned off by default.

Procedure

To enable extended error reporting, issue a command of this form:

```
# echo 1 > /sys/bus/ccw/devices/<device_bus_id>/eer_enabled
```

where /sys/bus/ccw/devices/*<device_bus_id>* represents the device in sysfs.

When it is enabled on a device, a specific set of errors generates records and might have further side effects.

Disable extended error reporting, issue a command of this form:

```
# echo 0 > /sys/bus/ccw/devices/<device_bus_id>/eer_enabled
```

What to do next

You can obtain error records for all DASD for which extended error reporting is enabled from the character device of the extended error reporting module, /dev/dasd_eer. The device supports these file operations:

open
Multiple processes can open the node concurrently. Each process that opens the node has access to the records that are created from the time the node is opened. A process cannot access records that were created before the process opened the node.

close
You can close the node as usual.

read
Blocking read and non-blocking read are supported. When a record is partially read and then purged, the next read returns an I/O error -EIO.

poll
The poll operation is typically used with non-blocking read.

Setting a DASD online or offline

Use the **chzdev** command, the **chccwdev** command, or the online sysfs attribute of the device to set DASDs online or offline.

About this task

When Linux boots, it senses your DASD. Depending on your specification for the "dasd=" parameter, it automatically sets devices online.

Procedure

Use the **chzdev** command (“chzdev - Configure IBM Z devices” on page 590) to set a DASD online or offline.

Alternatively, use the **chccwdev** command, or write 1 to the device's sysfs online attribute to set it online or 0 to set it offline. In contrast to the sysfs attribute, the **chccwdev** command triggers a cio_settle for you and waits for the cio_settle to complete.

Outstanding I/O requests are canceled when you set a device offline. To wait indefinitely for outstanding I/O requests to complete before setting the device offline, use the **chccwdev** option --safeoffline or the sysfs attribute safe_offline.fcp_The **chzdev** command uses safe offline (if available), unless you specify the --force option.

When you set a DASD offline, the deregistration process is synchronous, unless the device is disconnected. For disconnected devices, the deregistration process is asynchronous.

Examples

- To set a DASD with device bus-ID 0.0.b100 online, issue:

```
# chzdev -e dasd 0.0.b100 --active
```

or

```
# chccwdev -e 0.0.b100
```

or

```
# echo 1 > /sys/bus/ccw/devices/0.0.b100/online
```

- To set a DASD with device bus-ID 0.0.b100 offline, issue:

```
# chzdev -d dasd 0.0.b100 --active
```

or

```
# chccwdev -d 0.0.b100
```

or

```
# echo 0 > /sys/bus/ccw/devices/0.0.b100/online
```

- To complete outstanding I/O requests and then set a DASD with device bus-ID 0.0.4711 offline, issue:

```
# chccwdev -s 0.0.4711
```

or

```
# echo 1 > /sys/bus/ccw/devices/0.0.4711/safe_offline
```

If an outstanding I/O request is blocked, the command might wait forever. Reasons for blocked I/O requests include reserved devices that can be released or disconnected devices that can be reconnected.

1. Try to resolve the problem that blocks the I/O request and wait for the command to complete.
2. If you cannot resolve the problem, issue `chccwdev -d` to cancel the outstanding I/O requests. The data is lost.

Dynamic attach and detach

You can dynamically attach devices to a running Red Hat Enterprise Linux 9.1 for IBM Z instance, for example, from z/VM.

When a DASD is attached, Linux attempts to initialize it according to the DASD device driver configuration. You can then set the device online. You can automate setting dynamically attached devices online by using CCW hotplug events (see “CCW hotplug events” on page 19).

Attention: Do not detach a device that is still being used by Linux. Detaching devices might cause the system to hang or crash. Ensure that you unmount a device and set it offline before you detach it.

See “Working with newly available devices” on page 10 to avoid errors when working with devices that have become available to a running Linux instance.

Be careful to avoid errors when working with devices that have become available to a running Linux instance.

Enabling and disabling logging

Use the dasd= kernel or module parameter or use the `erplog` sysfs attribute to enable or disable error recovery processing (ERP) logging.

Procedure

You can enable and disable error recovery processing (ERP) logging on a running system. There are two methods:

- Use the dasd= parameter when you load the base module of the DASD device driver.

  Example:

  To define a device range (0.0.7000-0.0.7005) and enable, change the parameter line to contain:

  ```
  dasd=0.0.7000-0.0.7005(erplog)
  ```

- Use the sysfs attribute `erplog` to turn ERP-related logging on or off.

  Logging can be enabled for a specific device by writing `1` to the `erplog` attribute.

  Example:

  ```
  echo 1 > /sys/bus/ccw/devices/<device_bus_id>/erplog
  ```

  To disable logging, write `0` to the `erplog` attribute.

  Example:

  ```
  echo 0 > /sys/bus/ccw/devices/<device_bus_id>/erplog
  ```

Enabling and disabling immediate failure of I/O requests

Prevent devices in mirror setups from being blocked while paths are unavailable by making I/O requests fail immediately.

About this task

By default, if all path have been lost for a DASD, the corresponding device in Linux waits for one of the paths to recover. I/O requests are blocked while the device is waiting.

If the DASD is part of a mirror setup, this blocking might cause the entire virtual device to be blocked. You can use the `failfast` attribute to immediately return I/O requests as failed while no path to the device is available.

Attention: Use this attribute with caution and only in setups where a failed I/O request can be recovered outside the scope of a single DASD.

Procedure

Use one of these methods:

- You can enable immediate failure of I/O requests when you load the base module of the DASD device driver.

  Example:

  To define a device range (0.0.7000-0.0.7005) and enable immediate failure of I/O requests specify:

  ```
  dasd=0.0.7000-0.0.7005(failfast)
  ```

- You can use the sysfs attribute `failfast` of a DASD to enable or disable immediate failure of I/O requests on or off.

To enable immediate failure of I/O requests, write `1` to the `failfast` attribute.

Example:

```
echo 1 > /sys/bus/ccw/devices/<device_bus_id>/failfast
```

To disable immediate failure of I/O requests, write `0` to the `failfast` attribute.

Example:

```
echo 0 > /sys/bus/ccw/devices/<device_bus_id>/failfast
```

Setting the timeout for I/O requests

DASD I/O requests can time out at two levels in the software stack.

About this task

When the DASD device driver receives an I/O request from an application, it issues one or more low-level I/O requests to the affected storage system. Both the initial I/O request from the application and the resulting low-level requests to the storage system can time out. You set the timeout values through two sysfs attributes of the DASD.

expires
specifies the maximum time, in seconds, that the DASD device driver waits for a response to a low-level I/O request from a storage server.

The default for the maximum response time depends on the type of DASD:

ECKD
uses the default that is provided by the storage server.

FBA
300 s

DIAG
50 s

If the maximum response time is exceeded, the DASD device driver cancels the request. Depending on your setup, the DASD device driver might then try the request again, possibly in combination with other recovery actions.

timeout
specifies the time interval, in seconds, within which the DASD device driver must respond to an I/O request from a software layer above it. If the specified time expires before the request is completed, the DASD device driver cancels all related low-level I/O requests to storage systems and reports the request as failed.

This setting is useful in setups where the software layer above the DASD device driver requires an absolute upper limit for I/O requests.

A value of `0` means that there is no time limit. This value is the default.

Procedure

You can use the `expires` and `timeout` attributes of a DASD to change the timeout values for that DASD.

1. To find out the current timeout values, issue commands of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/expires
# cat /sys/bus/ccw/devices/<device_bus_id>/timeout
```

Example:

```
# cat /sys/bus/ccw/devices/0.0.7008/expires
30
# cat /sys/bus/ccw/devices/0.0.7008/timeout
0
```

In the example, a maximum response time of 30 seconds applies to the storage server for a DASD with bus ID `0.0.7008`. No total time limit is set for I/O requests to this DASD.

2. To set different timeout values, issue commands of this form:

```
# echo <max_wait> > /sys/bus/ccw/devices/<device_bus_id>/expires
# echo <total_max> > /sys/bus/ccw/devices/<device_bus_id>/timeout
```

where:

<max_wait>
is the new maximum response time, in seconds, for the storage server. The value must be a positive integer.

<total_max>
is the new maximum total time in seconds. The value must be a positive integer or `0`. `0` disables this timeout setting.

<device_bus_id>
is the device bus-ID of the DASD.

Example:

```
# echo 60 > /sys/bus/ccw/devices/0.0.7008/expires
# echo 120 > /sys/bus/ccw/devices/0.0.7008/timeout
```

This example sets timeout values for a DASD with bus ID `0.0.7008`. The maximum response time for the storage server is set to 60 seconds and the overall time limit for I/O requests is set to 120 seconds.

Working with DASD statistics in debugfs

Gather DASD statistics and display the data with the **dasdstat** command.

Before you begin

Instead of accessing raw DASD performance data in debugfs, you can use the **dasdstat** command to obtain more structured data (see “dasdstat - Display DASD performance statistics” on page 620).

About this task

The DASD performance data is contained in the following subdirectories of *<mountpoint>*`/dasd`, where *<mountpoint>* is the mount point of debugfs:

- A directory `global` that represents all available DASDs taken together.
- For each DASD, one directory with the name of the DASD block device with which the DASD is known to the DASD device driver (for example, `dasda`, `dasdb`, and `dasdc`).
- For each CCW device that corresponds to a DASD, a directory with the bus ID as the name.

  Block devices that are not set up for PAV or HyperPAV map to exactly one CCW device and the corresponding directories contain the same statistics.

  With PAV or HyperPAV, a bus ID can represent a base device or an alias device. Each base device is associated with a particular block device. The alias devices are not permanently associated with the same block device. At any one time, a DASD block device is associated with one or more CCW devices. Statistics that are based on bus ID, therefore, show more detail for PAV and HyperPAV setups.

Each of these directories contains a file `statistics` that you can use to perform these tasks:

- Start and stop data gathering.
- Reset statistics counters.
- Read statistics.

To control data gathering at the scope of a directory in *<mountpoint>*/dasd, issue a command of this form:

```
# echo <keyword> > <mountpoint>/dasd/<directory>/statistics
```

Where:

<directory>
is one of the directories in *<mountpoint>*/dasd.

<keyword>
specifies the action to be taken:

on
to start data gathering.

off
to stop data gathering.

reset
to reset the statistics counters.

To read performance data, issue a command of this form:

```
# cat <mountpoint>/dasd/<directory>/statistics
```

Examples for gathering and reading DASD statistics in debugfs

Use the **echo** command to start and stop data gathering for individual devices or across all DASDs. Use the **cat** command to access the raw performance data.

The following examples assume that debugfs is mounted at /sys/kernel/debug, see “debugfs” on page ix.

- To start data gathering for summary data across all available DASDs:

```
# echo on > /sys/kernel/debug/dasd/global/statistics
```

- To stop data gathering for block device dasdb:

```
# echo off > /sys/kernel/debug/dasd/dasdb/statistics
```

- To reset the counters for CCW device 0.0.b301:

```
# echo reset > /sys/kernel/debug/dasd/0.0.b301/statistics
```

- To read performance data for dasda, assuming that the degbugfs mount point is /sys/kernel/debug, issue:

```
# cat  /sys/kernel/debug/dasd/dasda/statistics
start_time 1283518578.085869197
total_requests 0
total_sectors 0
total_pav 0
total_hpf 0
histogram_sectors 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_io_times 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_io_times_weighted 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_time_build_to_ssch 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_time_ssch_to_irq 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_time_ssch_to_irq_weighted 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_time_irq_to_end 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_ccw_queue_length 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
total_read_requests 0
total_read_sectors 0
total_read_pav 0
total_read_hpf 0
histogram_read_sectors 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_read_times 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_read_time_build_to_ssch 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_read_time_ssch_to_irq 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_read_time_irq_to_end 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_read_ccw_queue_length 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Interpreting the data rows

The raw DASD performance data in the `statistics` directories in debugfs is organized into labeled data rows.

This section explains the raw data in the individual data rows of the statistics. Use the **dasdstat** command to obtain more structured data.

start_time
is the UNIX epoch time stamp when data gathering was started or when the counters were last reset.

Tip: Use the **date** tool to convert the time stamp to a more readily human-readable format. See the **date** man page for details.

Single counters
have a single integer as the statistics data. All rows with labels that begin with `total_` are of this data type.

The following rows show data for the sum of all requests, read and write:

total_requests
is the number of requests that have been processed.

total_sectors
is the sum of the sizes of all requests, in units of 512-byte sectors.

total_pav
is the number of requests that were processed through a PAV alias device.

total_hpf
is the number of requests that used High Performance FICON.

The following rows show data for read requests only:

total_read_requests
is the number of read requests that have been processed.

total_read_sectors
is the sum of the sizes of all read requests, in units of 512-byte sectors.

total_read_pav
is the number of read requests that were processed through a PAV alias device.

total_read_hpf
is the number of read requests that used High Performance FICON.

Linear histograms
have a series of 32 integers as the statistics data. The integers represent a histogram, with a linear scale, of the number of requests in the request queue each time a request has been queued. The first integer shows how often the request queue contained zero requests, the second integer shows how

often the queue contained one request, and the n-th value shows how often the queue contained n-1 requests.

histogram_ccw_queue_length
is the histogram data for all requests, read and write.

histogram_read_ccw_queue_length
is the histogram data for read requests only.

Logarithmic histograms
have a series of 32 integers as the statistics data. The integers represent a histogram with a logarithmic scale:

- The first integer always represents all measures of fewer than 4 units
- The second integer represents measures of 4 or more but less than 8 units
- The third integer represents measures of 8 or more but less than 16 units
- The n-th integer ($1 < n < 32$) represents measures of 2^n or more but less than 2^{n+1} units
- The 32nd integer represents measures of 2^{32} (= 4G = 4,294,967,296) units or more.

The following rows show data for the sum of all requests, read and write:

histogram_sectors
is the histogram data for request sizes. A unit is a 512-byte sector.

histogram_io_times
is the histogram data for the total time that is needed from creating the cqr to its completion in the DASD device driver and return to the block layer. A unit is a microsecond.

histogram_io_times_weighted
is the histogram data of the total time, as measured for `histogram_io_times`, devided by the requests size in sectors. A unit is a microsecond per sector.

This metric is deprecated and there is no corresponding histogram data for read requests.

histogram_time_build_to_ssch
is the histogram data of the time that is needed from creating the cqr to submitting the request to the subchannel. A unit is a microsecond.

histogram_time_ssch_to_irq
is the histogram data of the time that is needed from submitting the request to the subchannel until an interrupt indicates that the request has been completed. A unit is a microsecond.

histogram_time_ssch_to_irq_weighted
is the histogram data of the time that is needed from submitting the request to the subchannel until an interrupt indicates that the request has been completed, divided by the request size in 512-byte sectors. A unit is a microsecond per sector.

This metric is deprecated and there is no corresponding histogram data for read requests.

histogram_time_irq_to_end
is the histogram data of the time that is needed from return of the request from the channel subsystem, until the request is returned to the block layer. A unit is a microsecond.

The following rows show data for read requests only:

histogram_read_sectors
is the histogram data for read request sizes. A unit is a 512-byte sector.

histogram_read_io_times
is the histogram data, for read requests, for the total time that is needed from creating the cqr to its completion in the DASD device driver and return to the block layer. A unit is a microsecond.

histogram_read_time_build_to_ssch
is the histogram data, for read requests, of the time that is needed from creating the cqr to submitting the request to the subchannel. A unit is a microsecond.

histogram_read_time_ssch_to_irq
is the histogram data, for read requests, of the time that is needed from submitting the request to the subchannel until an interrupt indicates that the request has been completed. A unit is a microsecond.

histogram_read_time_irq_to_end
is the histogram data, for read requests, of the time that is needed from return of the request from the channel subsystem, until the request is returned to the block layer. A unit is a microsecond.

Scenario: Verifying that PAV and HPF are used

Use the **dasdstat** command to display DASD performance statistics, including statistics about Parallel Access Volume (PAV) and High Performance FICON (HPF).

Procedure

1. Enable DASD statistics for the device of interest.

   Example:

   ```
   # dasdstat -e dasdc
   enable statistic "/sys/kernel/debug/dasd/dasdc/statistics"
   ```

2. Assure that I/O requests are directed to the device.

   Hints:

   - Access a partition, rather than the whole device, to avoid directing the I/O request towards the first 2 tracks of a CDL formatted DASD. Requests to the first 2 tracks of a CDL formatted DASD are exceptional in that they never use High Performance FICON.
   - Assure that a significant I/O load is applied to the device. PAV aliases are used only if multiple I/O requests for the device are processed simultaneously.

   Example:

   ```
   # dd if=/dev/dasdc1 of=/dev/null bs=4k count=256
   ```

3. Look for PAV and HPF in the statistics.

   Example:

   ```
   # dasdstat dasdc
   ------------------------------------------------------------------------------
   statistics data for statistic: dasdc
   start time of data collection: Fri Dec 11 14:22:18 CET 2015

   7 dasd I/O requests
   with 4000 sectors(512B each)
   3 requests used a PAV alias device
   7 requests used HPF
   ```

   In the example, `dasdc` uses both Parallel Access Volume and High Performance FICON.

Accessing full ECKD tracks

In raw-track access mode, the DASD device driver accesses full ECKD tracks, including record zero and the count and key data fields.

Before you begin

- This section applies to ECKD type DASD only.
- The DASD must be offline when you change the access mode.
- The DIAG access method must not be enabled for the device.

About this task

With this mode, Linux can access an ECKD device regardless of the track layout. In particular, the device does not need to be formatted for Linux.

For example, with raw-track access mode Linux can create a backup copy of any ECKD device. Full-track access can also enable a special program that runs on Linux to access and process data on an ECKD device that is not formatted for Linux.

By default, the DASD device driver accesses only the data fields of ECKD devices. In default access mode, you can work with partitions, file systems, and files in the file systems on the DASD.

When using a DASD in raw-track access mode be aware that:

- In memory, each track is represented by 64 KB of data, even if the track occupies less physical disk space. Therefore, a disk in raw-track access mode appears bigger than in default mode.
- Programs must write and should read data in multiples of complete 64 KB tracks. The minimum is a single track. The maximum is eight tracks by default but can be extended to up to 16 tracks.

  The maximum number of tracks depends on the maximum number of sectors as specified in the `max_sectors_kb` sysfs attribute of the DASD. This attribute is located in the block device branch of sysfs at `/sys/block/dasd<x>/queue/max_sectors_kb`. In the path, `dasd<x>` is the device name assigned by the DASD device driver.

  To extend the maximum beyond eight tracks, set the `max_sectors_kb` to the maximum amount of data to be processed in a single read or write operation. For example, to extend the maximum to reading or writing 16 tracks at a time, set `max_sectors_kb` to `1024` (16 x 64).
- Programs must write only valid ECKD tracks of 64 KB.
- Programs must use direct I/O to prevent the Linux block layer from splitting tracks into fragments. Open the block device with option O_DIRECT or work with programs that use direct I/O.

  For example, the options `iflag=direct` and `oflag=direct` cause **dd** to use direct I/O. When using **dd**, also specify the block size with the `bs=` option. The block size determines the number of tracks that are processed in a single I/O operation. The block size must be a multiple of 64 KB and can be up to 1024 KB. Specifying a larger block size often results in better performance. If you receive disk image data from a pipe, also use the option `iflag=fullblock` to ensure that full blocks are written to the DASD device.

  Tools cannot directly work with partitions, file systems, or files within a file system. For example, **fdasd** and **dasdfmt** cannot be used.

Procedure

To change the access mode, issue a command of this form:

```
# echo <switch> > /sys/bus/ccw/devices/<device_bus_id>/raw_track_access
```

where:

<switch>
is 1 to activate raw data access and 0 to deactivate raw data access.

<device_bus_id>
identifies the DASD.

Example

The following example creates a backup of a DASD `0.0.7009` on a DASD `0.0.70a1`.

The initial commands ensure that both devices are offline and that the DIAG access method is not enabled for either of them. The subsequent commands activate the raw-track access mode for the two devices and set them both online. The **lsdasd** command that follows shows the mapping between device bus-IDs and device names.

The **dd** command for the copy operation specifies direct I/O for both the input and output device and the block size of 1024 KB. After the copy operation is completed, both devices are set offline. The access mode for the original device is then set back to the default and the device is set back online.

```
#cat /sys/bus/ccw/devices/0.0.7009/online
1
# chccwdev -d 0.0.7009
# cat /sys/bus/ccw/devices/0.0.7009/use_diag
0
# cat /sys/bus/ccw/devices/0.0.70a1/online
0
# cat /sys/bus/ccw/devices/0.0.70a1/use_diag
0
# echo 1 > /sys/bus/ccw/devices/0.0.7009/raw_track_access
# echo 1 > /sys/bus/ccw/devices/0.0.70a1/raw_track_access
# chccwdev -e 0.0.7009,0.0.70a1
# lsdasd 0.0.7009 0.0.70a1
Bus-ID     Status      Name      Device  Type  BlkSz  Size      Blocks
==============================================================================
0.0.7009   active      dasdf     94:20   ECKD  4096   7043MB    1803060
0.0.70a1   active      dasdj     94:36   ECKD  4096   7043MB    1803060
# echo 1024 > /sys/block/dasdf/queue/max_sectors_kb
# echo 1024 > /sys/block/dasdj/queue/max_sectors_kb
# dd if=/dev/dasdf of=/dev/dasdj bs=1024k iflag=direct oflag=direct
# chccwdev -d 0.0.7009,0.0.70a1
# echo 0 > /sys/bus/ccw/devices/0.0.7009/raw_track_access
# chccwdev -e 0.0.7009
```

Handling lost device reservations

A DASD reservation by your Linux instance can be lost if another system unconditionally reserves this DASD.

About this task

This other system then has exclusive I/O access to the DASD for the duration of the unconditional reservation. Such unconditional reservations can be useful for handling error situations where:

- Your Linux instance cannot gracefully release the DASD.
- Another system requires access to the DASD, for example, to perform recovery actions.

After the DASD is released by the other system, your Linux instance might process pending I/O requests and write faulty data to the DASD. How to prevent pending I/O requests from being processed depends on the reservation policy. There are two reservation policies:

ignore
: All I/O operations for the DASD are blocked until the DASD is released by the second system. When using this policy, reboot your Linux instance before the other system releases the DASD. This policy is the default.

fail
: All I/O operations are returned as failed until the DASD is set offline or until the reservation state is reset. When using this policy, set the DASD offline and back online after the problem is resolved. See “Reading and resetting the reservation state” on page 165 about resetting the reservation state to resume operations.

Procedure

Set the reservation policy with a command of this form:

```
# echo <policy> > /sys/bus/ccw/devices/<device_bus_id>/reservation_policy
```

where:

<device_bus_id>
: specifies the DASD.

<policy>
is one of the available policies, `ignore` or `fail`.

Examples

- The command of this example sets the reservation policy for a DASD with bus ID `0.0.7009` to `fail`.

```
# echo fail > /sys/bus/ccw/devices/0.0.7009/reservation_policy
```

- This example shows a small scenario. The first two commands confirm that the reservation policy of the DASD is `fail` and that the reservation has been lost to another system. Assuming that the error that had occurred has already been resolved and that the other system has released the DASD, operations with the DASD are resumed by setting it offline and back online.

```
# cat /sys/bus/ccw/devices/0.0.7009/reservation_policy
fail
# cat /sys/bus/ccw/devices/0.0.7009/last_known_reservation_state
lost
# chccwdev -d 0.0.7009
# chccwdev -e 0.0.7009
```

Reading and resetting the reservation state

How the DASD device driver handles I/O requests depends on the `last_known_reservation_state` sysfs attribute of the DASD.

About this task

The `last_known_reservation_state` attribute reflects the reservation state as held by the DASD device driver and can differ from the actual reservation state. Use the **tunedasd -Q** command to find out the actual reservation state. The `last_known_reservation_state` sysfs attribute can have the following values:

none
The DASD device driver has no information about the device reservation state. I/O requests are processed as usual. If the DASD is reserved by another system, the I/O requests remain in the queue until they time out, or until the reservation is released.

reserved
The DASD device driver holds a valid reservation for the DASD and I/O requests are processed as usual. The DASD device driver changes this state if notified that the DASD is no longer reserved to this system. The new state depends on the reservation policy (see “Handling lost device reservations” on page 164).

ignore
The state is changed to none.

fail
The state is changed to `lost`.

lost
The DASD device driver had reserved the DASD, but subsequently another system has unconditionally reserved the DASD (see “Handling lost device reservations” on page 164). The device driver processes only requests that query the actual device reservation state. All other I/O requests for the device are returned as failed.

When the error that led another system to unconditionally reserve the DASD is resolved and the DASD has been released by this other system there are two methods for resuming operations:

- Setting the DASD offline and back online.
- Resetting the reservation state of the DASD.

Attention: Do not resume operations by resetting the reservation state unless your system setup maintains data integrity on the DASD despite:

- The I/O errors that are caused by the unconditional reservation
- Any changes to the DASD through the other system

You reset the reservation state by writing `reset` to the `last_known_reservation_state` sysfs attribute of the DASD. Resetting is possible only for the `fail` reservation policy (see "Handling lost device reservations" on page 164) and only while the value of the `last_known_reservation_state` attribute is `lost`.

To find out the reservation state of a DASD issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/last_known_reservation_state
```

where *<device_bus_id>* specifies the DASD.

Example

The command in this example queries the reservation state of a DASD with bus ID `0.0.7009`.

```
# cat /sys/bus/ccw/devices/0.0.7009/last_known_reservation_state
reserved
```

Setting defective channel paths offline automatically

Control the automatic removal of defective channel paths through the `path_threshold`, `path_interval`, and `path_autodisable` sysfs attributes.

About this task

A channel control check (CCC) is caused by any machine malfunction that affects channel-subsystem controls. An interface control check (IFCC) indicates that an incorrect signal occurred on the channel path. Usually, these errors can be recovered automatically.

However, if IFCC or CCC errors occur frequently on a particular channel path, these errors indicate a failure of this channel path. Error recovery processing on defective channel paths can result in performance degradation. If at least one operational channel path remains, overall device performance might improve if a defective channel path is excluded from I/O.

By default, automatic path removal is enabled with an error threshold of 256 and a reset interval of 300 s (5 minutes). Accordingly, a channel path is set offline automatically, when the error count reaches 256 and if at least one other channel path remains. If 300 seconds elapse without an error, the error count is reset to 0.

You can change the error threshold and reset interval, or you can prevent automatic removal of channel paths altogether.

Procedure

- To specify the number of errors that must occur before the channel path is taken offline, issue a command of this form:

  ```
  # echo <no_of_errors> > /sys/bus/ccw/devices/<device_bus_id>/path_threshold
  ```

  where `/sys/bus/ccw/devices/`*<device_bus_id>* represents the device in sysfs and *<no_of_errors>* is an integer that specifies the error threshold.

  To disable detecting defective paths, and to suppress messages about IFCC or CCC errors, set *<no_of_errors>* to `0`.

- To specify the time that must elapse without errors to trigger a counter reset, issue a command of this form:

```
# echo <time> > /sys/bus/ccw/devices/<device_bus_id>/path_interval
```

  where *<time>* is the reset interval in seconds.
- To prevent defective paths from being set offline automatically, issue a command of this form:

```
# echo <flag> > /sys/bus/ccw/devices/<device_bus_id>/path_autodisable
```

  where *<flag>* can be "1" to enable automatic path removal, or "0" to prevent automatic path removal. By default, automatic path removal is enabled.

Examples

- Setting 512 for the error threshold and 6 minutes (360 s) for the reset interval:

```
echo 512 > /sys/bus/ccw/devices/0.0.4711/path_threshold
echo 360 > /sys/bus/ccw/devices/0.0.4711/path_interval
```

  According to this example, a channel path is automatically removed if a count of 512 IFCCs or CCCs is reached. Any 6-minute interval without a IFCCs or CCCs causes the counter to be reset to zero.
- Preventing automatic removal of defective channel paths:

```
# echo 0 > /sys/bus/ccw/devices/0.0.4711/path_autodisable
```

  In this example, messages about defective paths are issued according to the settings for the error threshold and the reset interval, but defective paths are not removed automatically.

What to do next

After you repair the faulty channel path, set it online again by using the **tunedasd** command with the `-p` option. See "tunedasd - Adjust low-level DASD settings" on page 741 for details.

Querying the HPF setting of a channel path

Query the High Performance FICON (HPF) state of a channel path through the `hpf` sysfs attribute. The HPF function can be lost if the device cannot provide the function, or if the channel path is not able to do HPF.

About this task

The HPF channel-path is deactivated if an HPF error occurs indicating that HPF is not available if there are other channel paths available. If no other channel paths are available, the path remains operational with HPF deactivated.

If the device loses HPF functionality, HPF is disabled for all channel paths defined for the device.

Procedure

To query the HPF function for a channel path, issue a command of this form:

```
# lsdasd -l <device_bus_id>
```

Alternatively, you can query the sysfs attribute directly:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/hpf
```

where /sys/bus/ccw/devices/*<device_bus_id>* represents the device in sysfs.

Example

To query the availability of HPF for a device 0.0.4711, issue:

```
lsdasd -l 0.0.4711

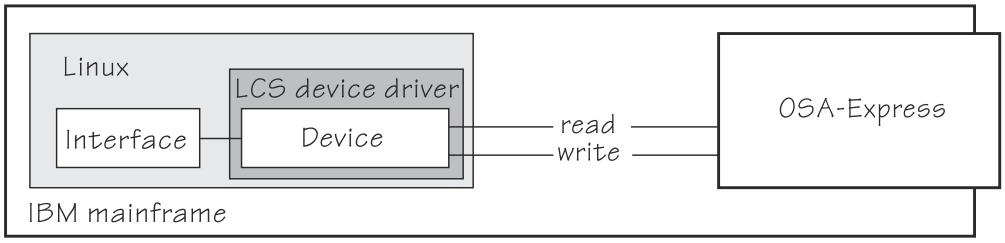
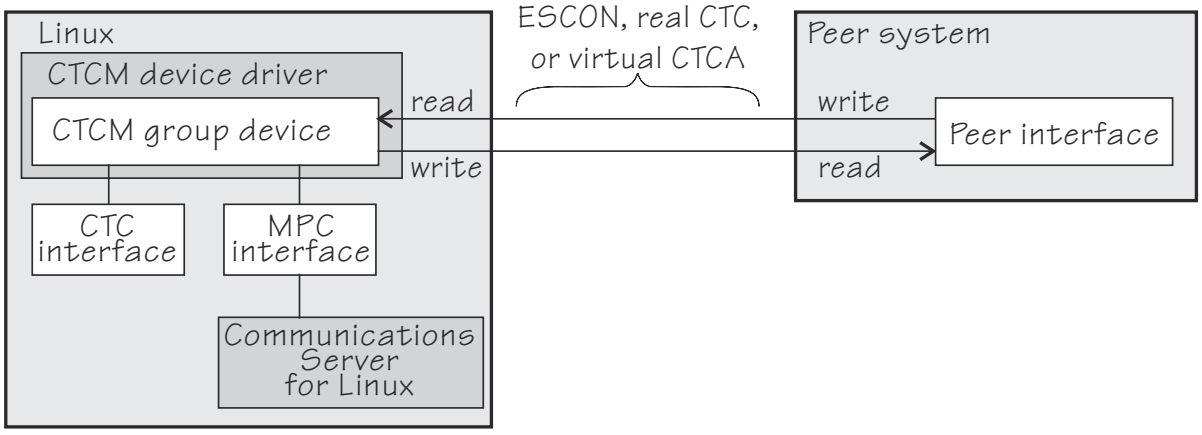
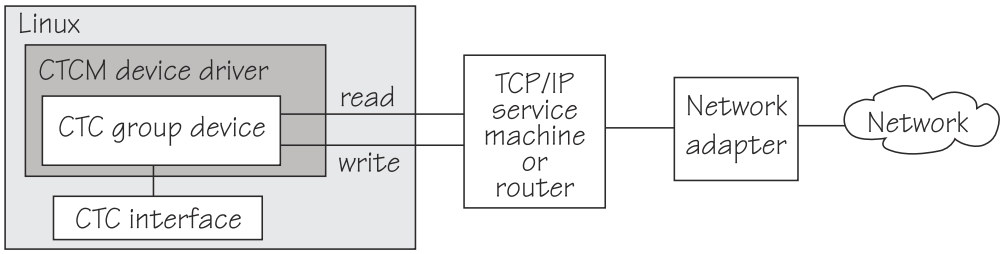
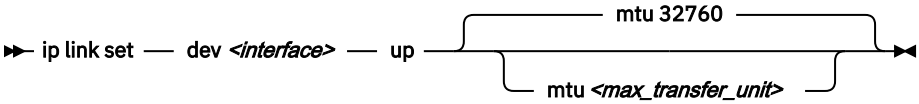
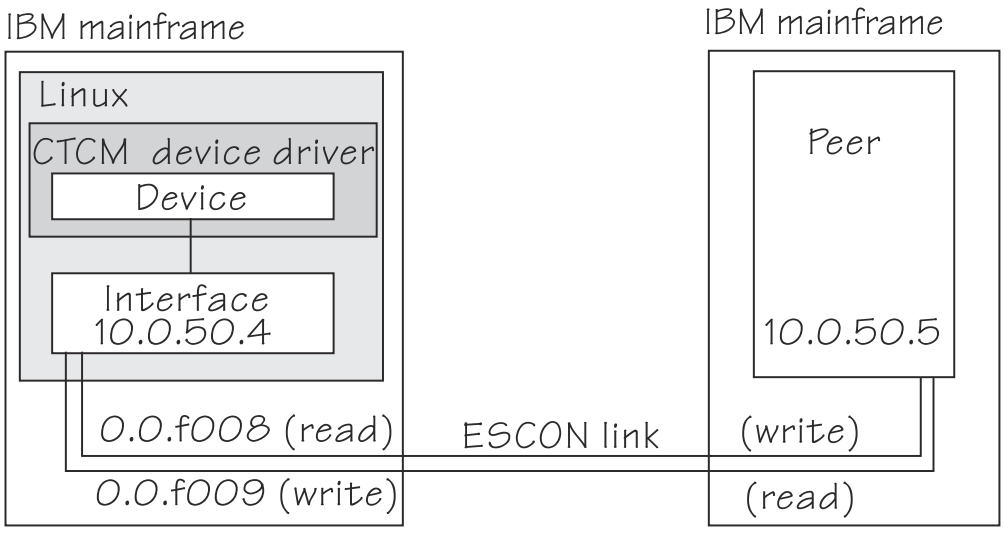
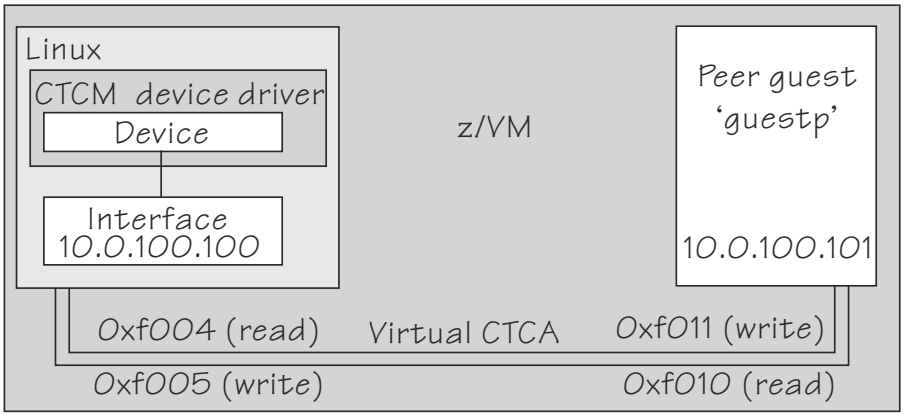
0.0.4711/dasdc/94:8
  status:                           active
  type:                             ECKD
  ...                               ...
  hpf:                              1
```

This example indicates that HPF is enabled for the device.

Alternatively, read from the `hpf` sysfs attribute:

```
cat /sys/bus/ccw/devices/0.0.4712/hpf
0
```

This example indicates that HPF is disabled for device 0.0.4712.

What to do next

You can now reset the paths to the device. You can use the **tunedasd** command to reset all or one channel path.

To re-validate all paths for one device and if possible reset HPF:

```
# tunedasd --path_reset_all /dev/dasdc
Resetting all chpids for device </dev/dasdc>...
Done.
```

See “tunedasd - Adjust low-level DASD settings” on page 741 for details.

You can also use sysfs to reset a path. sysfs expects a path mask. For example to reset CHPID 44, you can use **tunedasd**:

```
tunedasd -p 44 /dev/dasde
```

This would be the same as specifying the following in sysfs:

```
echo 08 > /sys/bus/ccw/devices/0.0.9330/path_reset
```

Both commands will reset CHPID 44 (path mask 08).

Checking for access by other operating system instances

Query if a DASD volume is online to another operating system instance by reading the `host_access_count` attribute.

Before you begin

To query the number of operating system instances that use the DASD device, the DASD must be online.

About this task

Storage servers that support this feature know about the online status of the device on all attached operating system instances in an LPAR (so called hosts). If a DASD device is set online it might potentially be used on another operating system instance. This information can help to reduce the chance for

outages or possible data corruption due to concurrent access to DASD volumes from different operating system instances.

Procedure

To check whether a DASD device is being used by other operating system instances, issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/host_access_count
```

where /sys/bus/ccw/devices/*<device_bus_id>* represents the device in sysfs.

For example, to query how many operating system instances have access to a device 0.0.bf45, issue:

```
# cat /sys/bus/ccw/devices/0.0.bf45/host_access_count
13
```

In the example, 13 operating system instances have access to the device, including the current Linux instance.

What to do next

To see details for each host connected to the DASD device, use the **lsdasd** command with the `--host-access-list` option. For more information and an example, see "lsdasd - List DASD devices" on page 661.

Managing extent space efficient DASDs

Thin provisioning is a method of optimizing space in a SAN, and can therefore affect the amount of storage capacity that you need. Storage systems with the thin provisioning feature can provide Extent Space Efficient (ESE) ECKD DASD.

About this task

The Linux kernel recognizes ESE DASDs. You can set them online and use them as usual, without taking any special action.

Procedure

To find out if a DASD is an ESE DASD, check the `ese` attribute of the device in sysfs. An attribute value of 1 identifies the device as an ESE DASD.
For example, for a DASD with device-bus ID 0.0.95d0:

```
# cat /sys/bus/ccw/devices/0.0.95d0/ese
1
```

Alternatively, you can use the **lsdasd** command, see "lsdasd - List DASD devices" on page 661.

Formatting ESE DASD

You can use the **dasdfmt** command to format an ESE DASD.

About this task

A new ESE DASD always has 0 allocated capacity, shown in the `space_allocated` sysfs attribute. When data is written to the DASD, this value increases as the storage server allocates storage to this device.

If you re-use an ESE DASD, and run **dasdfmt**, by default any space already allocated is released to reset the allocated capacity to 0. With the `--no-discard` option you can avoid the release of the allocated space, for example, if you wish to keep all data on the DASD, but want to re-initialize the VTOC.

Procedure

The **dasdfmt** command recognizes ESE DASD and, by default, sets the mode to quick.
All storage allocated for the DASD on the storage server is discarded.
For example, to use the quick format mode for an ESE DASD with device node `/dev/dasdc`, issue:

```
# dasdfmt /dev/dasdc
```

Use the `--no-discard` option to keep the storage allocated:

```
# dasdfmt --no-discard /dev/dasdc
```

See "dasdfmt - Format a DASD" on page 615 for detailed examples.

You can use `mode=full` to format the entire DASD. The entire available capacity is occupied.

Gathering information about ESE DASD

Several sysfs attributes provide information about ESE DASDs.

Procedure

- To examine the capacity of an ESE DASD, use the `capacity` sysfs attribute and its subentries.
  The following examples use an ESE DASD with device-bus ID 0.0.95d0.
  a) To see the logical size in cylinders, read the `logical_capacity` attribute, for example:

  ```
  # cat /sys/bus/ccw/devices/0.0.95d0/capacity/logical_capacity
  60102
  ```

  b) To see the currently used capacity in cylinders, read the `space_allocated` attribute, for example:

  ```
  # cat /sys/bus/ccw/devices/0.0.95d0/capacity/space_allocated
  27825
  ```

  For a non-ESE DASD, `space_allocated` is always the same as the `logical_capacity`.
  c) To see total available space in cylinders, read the `space_configured` attribute, for example:

  ```
  # cat /sys/bus/ccw/devices/0.0.95d0/capacity/space_configured
  34103433
  ```

- To examine the pool of extents in which the DASD is defined, use the `extent_pool` sysfs attribute, and its subentries.
  For example, to see the pool ID, read the pool_id attribute:

  ```
  # cat /sys/bus/ccw/devices/0.0.95d0/extent_pool/pool_id
  0
  ```

  To see the extent size, use the `extent_size` attribute, which shows the size of the extent in cylinders. 21 or 1113 cylinders are possible:

  ```
  # cat /sys/bus/ccw/devices/0.0.95d0/extent_pool/extent_size
  1113
  ```

  For information about how to define the number of cylinders, see the documentation of your storage server.
- To examine the utilization, use the `cap_at_warnlevel` and `warn_threshold` attributes.
  For example, to see whether the available capacity has reached the warning level:

```
# cat /sys/bus/ccw/devices/0.0.95d0/extent_pool/cap_at_warnlevel
0
# cat /sys/bus/ccw/devices/0.0.95d0/extent_pool/warn_threshold
15
```

where the value 15 means that the warning level is set to 85% for this extent pool on the storage server.
Should no space be left in the pool, the `pool_oos` attribute is set to 1. If so, all I/O traffic is stopped:

```
/sys/bus/ccw/devices/0.0.95d0/extent_pool/pool_oos
1
```

Otherwise the attribute shows `0`.

Querying the encryption setting of a channel path

A read-only attribute shows the Fibre Channel Endpoint Security status of the connection to the DASD device.

About this task

Fibre Channel Endpoint Security (FCES) is a hardware feature that encrypts traffic between the Z host system and storage server transparently. You can read the current state of the FCES for a DASD from the `fc_security` attribute. The attribute is available per DASD device and per path.

For a device, the attribute can take the following values:

Authentication
The connection is authenticated.

Encryption
The connection is encrypted.

Inconsistent
At least one of the operational paths is in a different state from all others.

Unsupported
The DASD device does not support FCES.

The sysfs attributes per path are organized in a directory called `paths_info` with sub-directories for each path. For example:

```
/sys/bus/ccw/devices/0.0.4711/paths_info/
|-- 0.38
| '-- fc_security
|-- 0.39
| '-- fc_security
|-- 0.3a
| '-- fc_security
```

For a path, the `fc_security` attribute can be `Authentication`, `Encrypted`, and `Unsupported`.

Procedure

To query the FCES status of a DASD device, issue a command of this form:

```
# lsdasd -l <device_bus_id>
```

or, using **lszdev**:

```
# lszdev <device_bus_id> -a -c TYPE,ID,ATTR:fc_security,ATTRPATH:fc_security
```

Alternatively, you can read the sysfs attribute directly:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/fc_security
```

where /sys/bus/ccw/devices/*<device_bus_id>* represents the device in sysfs.

Examples

- To query the FCES status for a device 0.0.4711, issue:

```
# lsdasd -l 0.0.4711

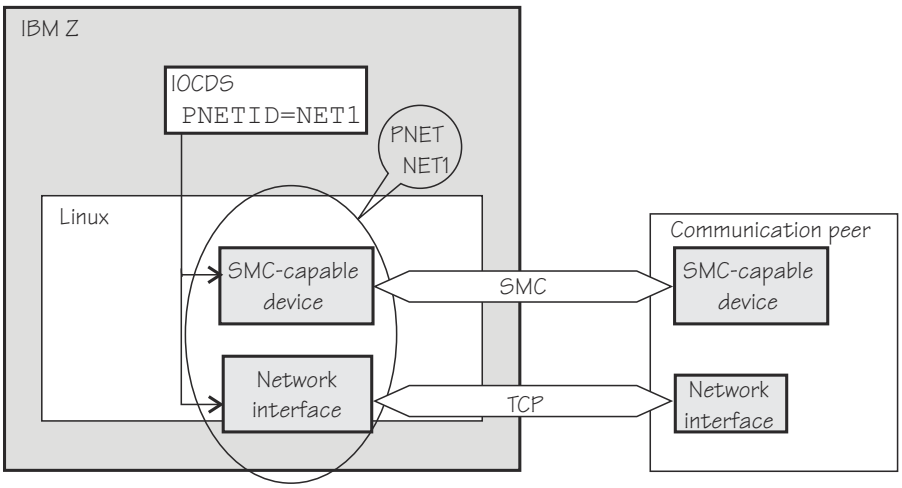
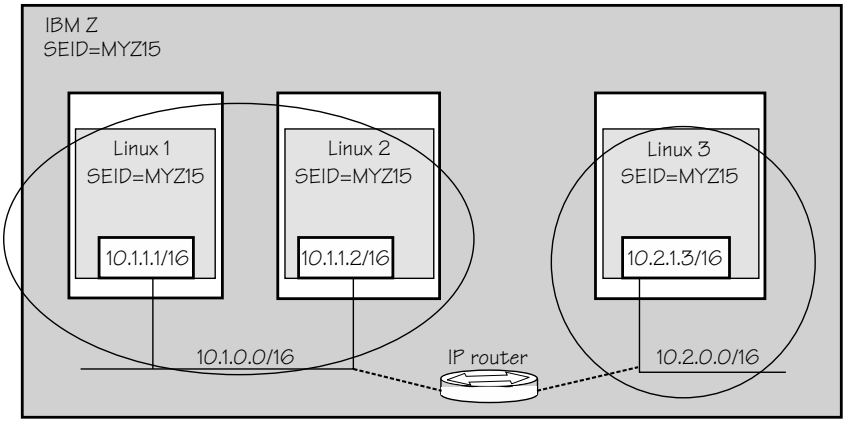
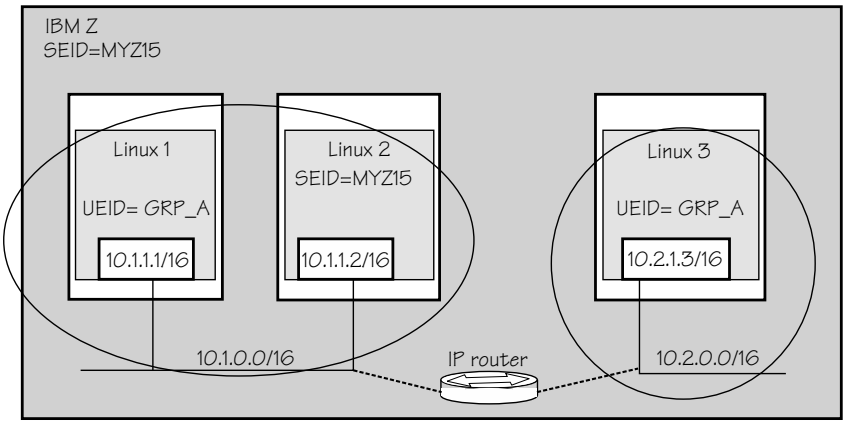
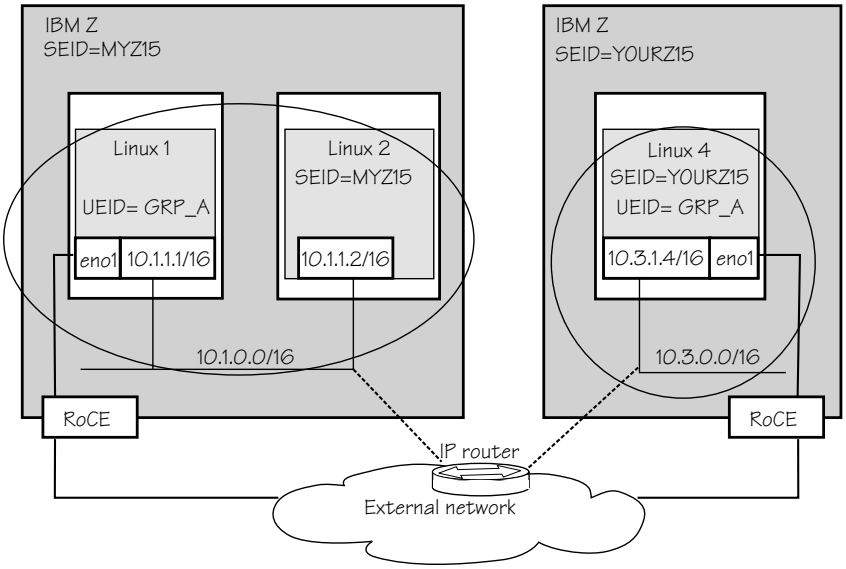
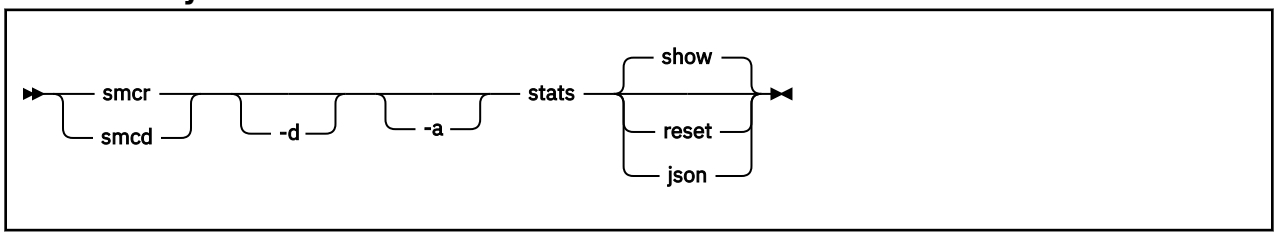
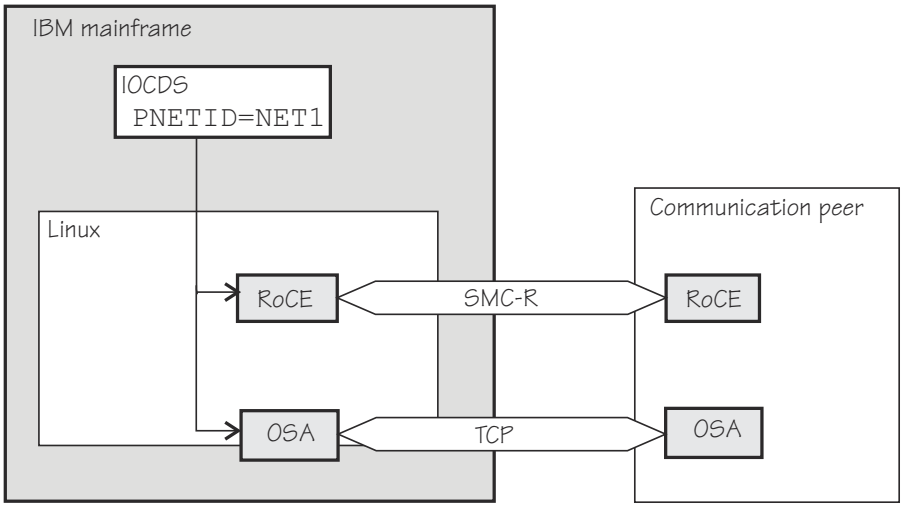
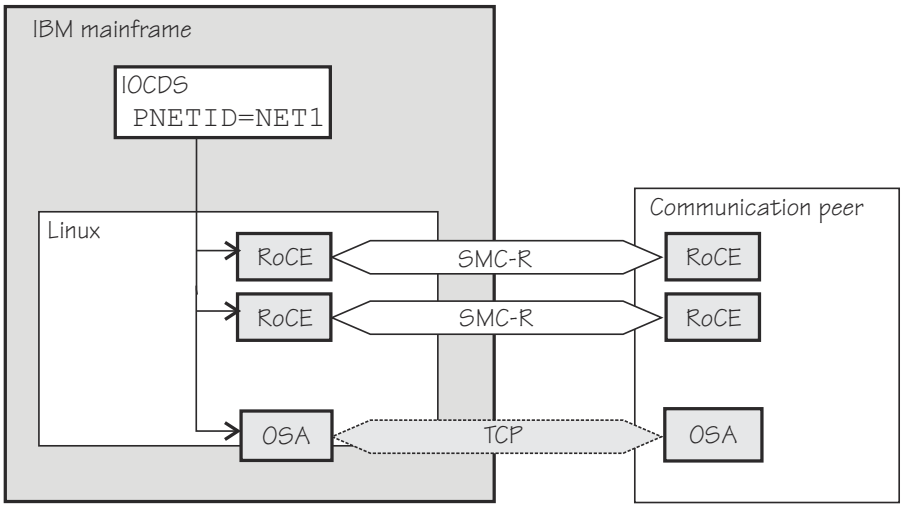
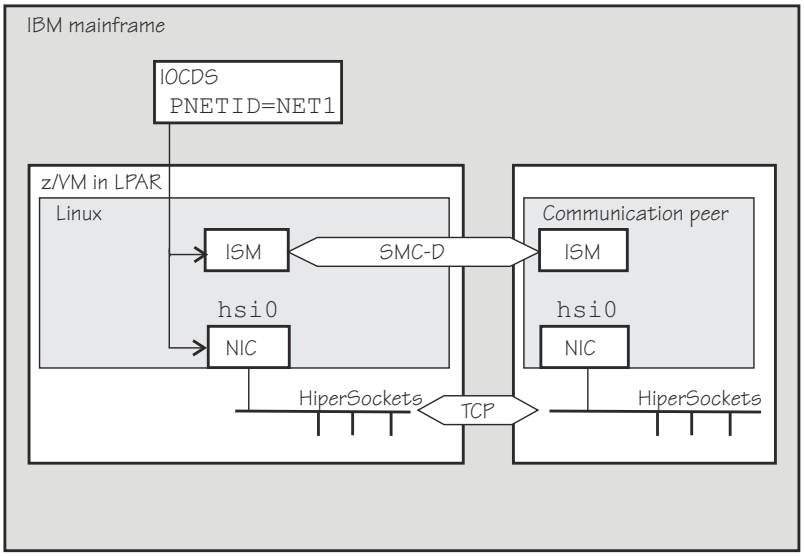
0.0.4711/dasdc/94:8
  status:                           active
  type:                             ECKD
  ...                               ...
  fc_security:                      Encryption
```

  This example indicates that the connection to the DASD is encrypted.
- To query the FCES status for a device 0.0.4711 using the **lszdev** command, issue:

```
$ lszdev 0.0.4711 -a -c TYPE,ID,ATTR:fc_security,ATTRPATH:fc_security
TYPE       ID       ATTR:fc_security  ATTRPATH:fc_security
dasd-eckd  0.0.4711 Encryption        /sys/bus/ccw/drivers/dasd-eckd/0.0.4711/fc_security
```

- To query the FCES status for a device 0.0.4712 by reading from the fc_security sysfs attribute:

```
# cat /sys/bus/ccw/devices/0.0.4712/fc_security
Unsupported
```

  This example indicates that DASD 0.0.4712 does not support FCES.
- To read the fc_security attribute of path 0.38 for DASD 0.0.4711, issue:

```
# cat /sys/bus/ccw/devices/0.0.4711/paths_info/0.38/fc_security
Encrypted
```

Displaying DASD information

Use tools to display information about your DASDs, or read the attributes of the devices in sysfs.

About this task

There are several methods to display DASD information:

- Use **lsdasd -l** (see “lsdasd - List DASD devices” on page 661) to display summary information about the device settings and the device geometry of multiple DASDs.
- Use **dasdview** (see “dasdview - Display DASD structure” on page 622) to display details about the contents of a particular DASD.
- Read information about a particular DASD from sysfs, as described in this section.

The sysfs representation of a DASD is a directory of the form /sys/bus/ccw/devices/*<device_bus_id>*, where *<device_bus_id>* is the bus ID of the DASD. This sysfs directory contains a number of attributes with information about the DASD.

Table 18. Attributes with DASD information

Attribute	Explanation
alias	1 if the DASD is a parallel access volume (PAV) alias device. 0 if the DASD is a PAV base device or has not been set up as a PAV device. For an example of how to use PAV see *How to Improve Performance with PAV*, SC33-8414 on IBM Documentation at https://www.ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_hp.html This attribute is read-only.
discipline	Indicates the base discipline, ECKD or FBA, that is used to access the DASD. If DIAG is enabled, this attribute might read DIAG instead of the base discipline. This attribute is read-only.
eer_enabled	1 if the DASD is enabled for extended error reporting, 0 if it is not enabled (see "Using extended error reporting for ECKD type DASD" on page 153).
erplog	1 if error recovery processing (ERP) logging is enabled, 0 if ERP logging is not enabled (see "Enabling and disabling logging" on page 156).
ese	Read-only attribute that contains 1 if the DASD is extent space efficient (ESE). Otherwise 0. For details, see "Managing extent space efficient DASDs" on page 169.
expires	Indicates the time, in seconds, that Linux waits for a response to an I/O request for the DASD. If this time expires, Linux considers a request as failed and cancels it (see "Setting the timeout for I/O requests" on page 157).
failfast	1 if I/O operations are returned as failed immediately when the last path to the DASD is lost. 0 if a wait period for a path to return expires before an I/O operation is returned as failed. (see "Enabling and disabling immediate failure of I/O requests" on page 156).
fc_security	Read-only attribute that contains `Encryption` if the connection to the DASD is encrypted. For details, see "Querying the encryption setting of a channel path" on page 171.
host_access_count	Shows how many operating system instances have access to the device. See "Checking for access by other operating system instances" on page 168.
hpf	1 if High Performance FICON is available for the device. See "Querying the HPF setting of a channel path" on page 167.
last_known_reservation_state	The reservation state as held by the DASD device driver. Values can be: **none** The DASD device driver has no information about the device reservation state. **reserved** The DASD device driver holds a valid reservation for the DASD. **lost** The DASD device driver had reserved the device, but this reservation has been lost to another system. See "Reading and resetting the reservation state" on page 165 for details.
online	1 if the DASD is online, 0 if it is offline (see "Setting a DASD online or offline" on page 154).

Table 18. Attributes with DASD information (continued)

Attribute	Explanation
path_autodisable path_interval path_threshold	Control the automatic removal of defective channel path (see "Setting defective channel paths offline automatically" on page 166)
raw_track_access	1 if the DASD is in raw-track access mode, 0 if it is in default access mode (see "Accessing full ECKD tracks" on page 162).
readonly	1 if the DASD is read-only, 0 if it can be written to. This attribute is a device driver setting and does not reflect any restrictions that are imposed by the device itself. This attribute is ignored for PAV alias devices.
status	Reflects the internal state of a DASD device. Values can be: **unknown** Device detection has not started yet. **new** Detection of basic device attributes is in progress. **detected** Detection of basic device attributes has finished. **basic** The device is ready for detecting the disk layout. Low-level tools can set a device to this state when changing the disk layout, for example, when formatting the device. **unformatted** The disk layout detection found no valid disk layout. The device is ready for use with low-level tools like **dasdfmt**. **ready** The device is in an intermediate state. **online** The device is ready for use.
uid	A device identifier of the form *<vendor>.<serial>.<subsystem_id>.<unit_address>.<minidisk_identifier>* where ***<vendor>*** is the specification from the vendor attribute. ***<serial>*** is the serial number of the storage system. ***<subsystem_id>*** is the ID of the logical subsystem to which the DASD belongs on the storage system. ***<unit_address>*** is the address that is used within the storage system to identify the DASD. ***<minidisk_identifier>*** is an identifier that the z/VM system assigns to distinguish between minidisks on the DASD. This part of the uid is only present for Linux on z/VM and if the z/VM version and service level support this identifier. This attribute is read-only.
use_diag	1 if the DIAG access method is enabled, 0 if the DIAG access method is not enabled (see "Enabling the DASD device driver to use the DIAG access method" on page 152). Do not enable the DIAG access method for PAV alias devices.

Table 18. Attributes with DASD information (continued)

Attribute	Explanation
vendor	Identifies the manufacturer of the storage system that contains the DASD. This attribute is read-only.

Additionally, the representation in sysfs of a DASD device has the following subdirectories:

- `/sys/bus/ccw/devices/<device_bus_id>/extent_pool`

  The `extent_pool` subdirectory contains read-only attributes that, for an ESE DASD, provides information about the pool of extents in which the DASD is defined.

- `/sys/bus/ccw/devices/<device_bus_id>/capacity`

  The `capacity` subdirectory contains read-only attributes that shows the capacity of a DASD in cylinders.

For details about the attributes, see "Gathering information about ESE DASD" on page 170.

There are some more attributes that are common to all CCW devices (see "Device directories" on page 9).

Procedure

Issue a command of this form to read an attribute:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/<attribute>
```

where *<attribute>* is one of the attributes of Table 18 on page 173.

Example

The following sequence of commands reads the attributes for a DASD with a device bus-ID 0.0.b100:

```
# cat /sys/bus/ccw/devices/0.0.b100/alias
0
# cat /sys/bus/ccw/devices/0.0.b100/discipline
ECKD
# cat /sys/bus/ccw/devices/0.0.b100/eer_enabled
0
# cat /sys/bus/ccw/devices/0.0.b100/erplog
0
# cat /sys/bus/ccw/devices/0.0.b100/ese
0
# cat /sys/bus/ccw/devices/0.0.b100/expires
30
# cat /sys/bus/ccw/devices/0.0.b100/failfast
0
# cat /sys/bus/ccw/devices/0.0.b100/host_access_count
1
# cat /sys/bus/ccw/devices/0.0.b100/hpf
1
# cat /sys/bus/ccw/devices/0.0.b100/last_known_reservation_state
reserved
# cat /sys/bus/ccw/devices/0.0.b100/online
1
# cat /sys/bus/ccw/devices/0.0.b100/path_autodisable
1
# cat /sys/bus/ccw/devices/0.0.b100/path_interval
300
# cat /sys/bus/ccw/devices/0.0.b100/path_threshold
256
# cat /sys/bus/ccw/devices/0.0.b100/raw_track_access
0
# cat /sys/bus/ccw/devices/0.0.b100/readonly
1
# cat /sys/bus/ccw/devices/0.0.b100/status
online
# cat /sys/bus/ccw/devices/0.0.b100/uid
IBM.75000000092461.e900.8a
# cat /sys/bus/ccw/devices/0.0.b100/use_diag
1
# cat /sys/bus/ccw/devices/0.0.b100/vendor
IBM
```

Chapter 12. SCSI-over-Fibre Channel device driver

LPAR and z/VM: The SCSI-over-Fibre Channel device driver applies to Linux in LPAR mode and to Linux on z/VM.

The SCSI-over-Fibre Channel device driver for Linux on IBM Z (zfcp device driver) supports virtual QDIO-based SCSI-over-Fibre Channel adapters (FCP devices) and attached SCSI devices (LUNs).

IBM Z adapter hardware typically provides multiple channels, with one port each. You can configure a channel to use the Fibre Channel Protocol (FCP). This *FCP channel* is then virtualized into multiple FCP devices. Thus, an FCP device is a virtual QDIO-based SCSI-over-Fibre Channel adapter with a single port.

A single physical port supports multiple FCP devices. Using N_Port ID virtualization (NPIV) you can define virtual ports and establish a one-to-one mapping between your FCP devices and virtual ports (see "N_Port ID Virtualization for FCP channels" on page 181).

On Linux, an FCP device is represented by a CCW device that is listed under `/sys/bus/ccw/drivers/zfcp`. Do not confuse FCP devices with SCSI devices. A SCSI device is identified by a LUN.

Features

The zfcp device driver supports a wide range of SCSI devices, various hardware adapters, specific topologies, and specific features that depend on the IBM Z hardware.

- Linux on IBM Z can use various SAN-attached SCSI device types, including SCSI disks, tapes, CD-ROMs, and DVDs.
- SAN access through the following FCP adapters:
  - FICON Express32S (as of IBM z16)
  - FICON Express16SA (as of z15)
  - FCP Express32S (LinuxONE only, as of LinuxONE II)
  - FICON Express16S+ (as of z14)
  - FICON Express16S (as of z13)
  - FICON Express8S (z13)
  - FICON Express8 (z13)

  You can order hardware adapters as features for mainframe systems.

  See *Fibre Channel Protocol for Linux and z/VM on IBM System z*, SG24-7266 for more details about using FCP with Linux on IBM Z.
- The zfcp device driver supports switched fabric and point-to-point topologies.
- The zfcp device driver supports end-to-end data consistency checking.
- As of FICON Express8S, the zfcp device driver supports the data router hardware feature to improve performance by reducing the path length.

To find out whether a combination of device and IBM mainframe is supported for your distribution, see the individual interoperability matrix for each storage device. The interoperability matrices are available at: www.ibm.com/systems/support/storage/ssic/interoperability.wss

For information about the maximum number of configurable devices, NPIV-enabled subchannels, and other configurations per FCP channel path, see *Input/Output Configuration Program User's Guide for ICP IOCP*, SB10-7172. Search for "FCP Channel Path Limits".

For information about SCSI-3, the Fibre Channel Protocol, and Fibre Channel related information, see www.t10.org and www.t11.org

What you should know about zfcp

The zfcp device driver is a low-level driver or host-bus adapter driver that supplements the Linux SCSI stack.

Figure 48 on page 178 illustrates how the device drivers work together.



Figure 48. Device drivers that support the FCP environment

sysfs structures for FCP devices and SCSI devices

FCP devices are CCW devices. In the sysfs device driver view, remote target ports with their LUNs are nested below the FCP devices.

When Linux is booted, it senses the available FCP devices and creates directories of the form:

```
/sys/bus/ccw/drivers/zfcp/<device_bus_id>
```

where *<device_bus_id>* is the device bus-ID that corresponds to an FCP device. You use the attributes in this directory to work with the FCP device.

Example: `/sys/bus/ccw/drivers/zfcp/0.0.3d0c`

The zfcp device driver automatically adds port information when the FCP device is set online and when remote storage ports (*target ports*) are added. Each added target port extends this structure with a directory of the form:

```
/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>
```

where *<wwpn>* is the worldwide port name (WWPN) of the target port. You use the attributes of this directory to work with the port.

Example: `/sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562`

With NPIV-enabled FCP devices, Red Hat Enterprise Linux 9.1 uses automatic LUN scanning by default. The zfcp sysfs branch ends with the target port entries. FCP devices that are not NPIV-enabled, or if automatic LUN scanning is disabled, can be configured manually, see “Configuring SCSI devices” on page 202.

Information about zfcp objects and their associated objects in the SCSI stack is distributed over the sysfs tree. To ease the burden of collecting information about zfcp devices, ports, units, and their associated SCSI stack objects, a command that is called **lszfcp** is provided with the s390utils RPM. See "lszfcp - List zfcp devices" on page 693 for more details about the command.

See also "Mapping the representations of a SCSI device in sysfs" on page 204.

SCSI device nodes

User space programs access SCSI devices through device nodes.

SCSI device names are assigned in the order in which the devices are detected. In a typical SAN environment, this can mean a seemingly arbitrary mapping of names to actual devices that can change between boots. Therefore, using standard device nodes of the form `/dev/<device_name>` where *<device_name>* is the device name that the SCSI stack assigns to a device, can be a challenge.

Red Hat Enterprise Linux 9.1 provides udev to create device nodes for you. Use the device nodes to identify the corresponding actual device.

Device nodes that are based on device names
udev creates device nodes that match the device names used by the kernel. These standard device nodes have the form `/dev/<name>`.

The examples in this section use standard device nodes as assigned by the SCSI stack. These nodes have the form `/dev/sd<x>` for entire disks and `/dev/sd<x><n>` for partitions. In these node names *<x>* represents one or more letters and *<n>* is an integer. See `Documentation/devices.txt` in the Linux source tree for more information about the SCSI device naming scheme.

To help you identify a particular device, udev creates device nodes that are based on the device's bus ID, the device label, and information about the file system on the device. The file system information can be a universally unique identifier (UUID) and, if available, the file system label.

Device nodes that are based on bus IDs
udev creates device nodes of the form

```
/dev/disk/by-path/ccw-<device_bus_id>-fc-<wwpn>-lun-<lun>
```

for whole SCSI device and

```
/dev/disk/by-path/ccw-<device_bus_id>-fc-<wwpn>-lun-<lun>-part<n>
```

for the *<n>*th partition, where *<wwpn>* is the worldwide port number of the target port and *<lun>* is the logical unit number that represents the target SCSI device.

Note: The format of these udev-created device nodes has changed and now matches the common code format. Device nodes of the prior form, `ccw-<device_bus_id>-zfcp-<wwpn>:<lun>` or `ccw-<device_bus_id>-zfcp-<wwpn>:<lun>-part<n>`, are also created for compatibility reasons.

Device nodes that are based on file system information
udev creates device nodes of the form

```
/dev/disk/by-uuid/<uuid>
```

where *<uuid>* is a unique file-system identifier (UUID) for the file system in a partition.

If a file system label has been assigned, udev also creates a node of the form

```
/dev/disk/by-label/<label>
```

There are no device nodes for the whole SCSI device that are based on file system information.

Additional device nodes
`/dev/disk/by-id` contains additional device nodes for the SCSI device and partitions, that are all based on a unique SCSI identifier generated by querying the device.

Example

For a SCSI device that is assigned the device name sda, has two partitions labeled boot and SWAP-sda2, a device bus-ID 0.0.3c1b (device number 0x3c1b), and a UUID 7eaf9c95-55ac-4e5e-8f18-065b313e63ca for the first and b4a818c8-747c-40a2-bfa2-acaa3ef70ead for the second partition, udev creates the following device nodes:

For the whole SCSI device:

- `/dev/sda` (standard device node according to the SCSI device naming scheme)
- `/dev/disk/by-path/ccw-0.0.3c1b-fc-0x500507630300c562-lun-0x401040ea00000000`
- `/dev/disk/by-id/scsi-36005076303ffc56200000000000010ea`
- `/dev/disk/by-id/wwn-0x6005076303ffc56200000000000010ea`

For the first partition:

- `/dev/sda1` (standard device node according to the SCSI device naming scheme)
- `/dev/disk/by-path/ccw-0.0.3c1b-fc-0x500507630300c562-lun-0x401040ea00000000-part1`
- `/dev/disk/by-uuid/7eaf9c95-55ac-4e5e-8f18-065b313e63ca`
- `/dev/disk/by-label/boot`
- `/dev/disk/by-id/scsi-36005076303ffc56200000000000010ea-part1`
- `/dev/disk/by-id/wwn-0x6005076303ffc56200000000000010ea-part1`

For the second partition:

- `/dev/sda2` (standard device node according to the SCSI device naming scheme)
- `/dev/disk/by-path/ccw-0.0.3c1b-fc-0x500507630300c562-lun-0x401040ea00000000-part2`
- `/dev/disk/by-uuid/b4a818c8-747c-40a2-bfa2-acaa3ef70ead`
- `/dev/disk/by-label/SWAP-sda2`
- `/dev/disk/by-id/scsi-36005076303ffc56200000000000010ea-part2`
- `/dev/disk/by-id/wwn-0x6005076303ffc56200000000000010ea-part2`

Device nodes by-uuid use a unique file-system identifier that does not relate to the partition number.

Multipath

Users of SCSI-over-Fibre Channel attached devices should always consider setting up and using redundant paths through their Fibre Channel storage area network.

Path redundancy improves the availability of the LUNs. In Linux, you can set up path redundancy with the device-mapper multipath tool. For information about multipath devices and multipath partitions, see the chapter about multipathing in *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413.

Partitioning a SCSI device

You can partition SCSI devices that are attached through an FCP channel in the same way that you can partition SCSI attached devices on other platforms.

About this task

Use the **fdisk** command to partition a SCSI disk, not **fdasd**.

udev creates device nodes for partitions automatically. For the SCSI disk `/dev/sda`, the partition device nodes are called `/dev/sda1`, `/dev/sda2`, `/dev/sda3`, and so on.

Example

To partition a SCSI disk with a device node `/dev/sda` issue:

```
# fdisk /dev/sda
```

zfcp HBA API (FC-HBA) support

The zfcp host bus adapter API (HBA API) provides an interface for HBA management clients that run on IBM Z.

As shown in Figure 49 on page 181, the zfcp HBA API support includes a user space library.



Figure 49. zfcp HBA API support modules

The default method in Red Hat Enterprise Linux 9.1 is for applications to use the zFCP HBA API library directly. If you develop applications yourself, see "Developing applications" on page 219.

In a Linux on IBM Z environment HBAs are usually virtualized and are shown as *FCP devices*. FCP devices are represented by CCW devices that are listed in `/sys/bus/ccw/drivers/zfcp`. Do not confuse FCP devices with SCSI devices. A SCSI device is a disk device that is identified by a LUN.

For information about setting up the HBA API support, see "zfcp HBA API support" on page 219.

N_Port ID Virtualization for FCP channels

Through N_Port ID Virtualization (NPIV), the sole port of an FCP channel appears as multiple, distinct ports with separate port identification.

NPIV support can be configured on the SE per CHPID and LPAR for an FCP channel. The zfcp device driver supports NPIV error messages and adapter attributes. See "Displaying FCP channel and device information" on page 187 for the Fibre Channel adapter attributes.

See also the chapter on NPIV in *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413.

Automatic re-IPL path failover

The **chreipl-fcp-mpath** tool set helps you to use multipath information for re-IPL path failover on a running Linux instance.

When the configured re-IPL path becomes unavailable it automatically changes the configured re-IPL path to a different operational path to the same volume. For more information, see "Automatic path failover for re-IPL from an FC-attached SCSI disk" on page 118.

Setting up the zfcp device driver

Red Hat Enterprise Linux loads the zfcp device driver for you when an FCP device becomes available. Use anaconda or dracut to configure the zfcp device driver. You might also need to install the zfcp HBA API library.

Important: Configuration changes can directly or indirectly affect information that is required to mount the root file system. Such changes require an update of the initial RAM disk, followed by a re-write of the boot record (see "Rebuilding the initial RAM disk image" on page 96).

You have the following options for configuring FCP LUNs to attach SCSI devices:

- During installation, use the **anaconda** GUI, the **dracut** boot parameter `rd.zfcp=`, or the **kickstart** parameter `zfcp`.

  Important: Linux on IBM Z kernel parameters in the boot configuration have a size limit of 895 bytes. Automatically generated rd.zfcp parameters might cause this limit to be exceeded. For details, see "Preventing excessive kernel parameters during installation" on page 562.
- On an installed system, use the **dracut** boot parameter `rd.zfcp=` only for SCSI disks that are required for the root file system. Use the configuration file `/etc/zfcp.conf` for all other SCSI devices, such as data volumes or tape libraries.

For details about configuration, see the Red Hat documentation website https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

You can always specify additional zfcp module parameters as explained in Chapter 4, "Kernel and module parameters," on page 25.

zfcp module parameter syntax



where:

allow_lun_scan=*<value>*
disables the automatic LUN scan for FCP devices that run in NPIV mode if set to 0, n, or N. To enable the LUN scanning set the parameter to 1, y, or Y. When the LUN scan is disabled, all LUNs must be configured through the unit_add zfcp attribute in sysfs. LUN scan is enabled by default.

Warning: Only disable automatic LUN scan as a workaround according to "Preventing excessive kernel parameters during installation" on page 562 or if advised to do so by an IBM Support engineer. Running without the automated LUN scan can adversely affect resilience against path failures.

datarouter=
enables (if set to 1, y, or Y) or disables (if set to 0, n, or N) support for the hardware data routing feature. The default is 1.

Note: The hardware data routing feature becomes active only for FCP devices that are based on adapter hardware with hardware data routing support.

dbflevel=*<level>*
sets the initial log level of the debug feature. The value is an integer in the range 0 - 6, where greater numbers generate more detailed information. The default is 3.

dbfsize=*<pages>*
specifies the number of pages to be used for the debug feature.

The debug feature is available for each FCP device and the following areas:

hba
FCP device

san
Storage Area Network

rec
Error Recovery Process

scsi
SCSI

pay
Payloads for entries in the hba, san, rec, or scsi areas. The default is 8 pages.

The value given is used for all areas. The default is 4, that is, four pages are used for each area and FCP device. In the following example the `dbfsize` is increased to 6 pages:

```
dbfsize=6
```

This results in six pages being used for each area and FCP device. The payload is doubled to use 12 pages.

dif=
turns on end-to-end data consistency checking in DIF-only mode if set to 1, y, or Y (and off if set to 0, n, or N). The default is 0.

Interface change: As of Red Hat Enterprise Linux 8.2, the `dif=` module parameter enables DIF only and no longer includes DIX. Use the `dix=` module parameter to enable end-to-end data consistency checking in extended mode, with both DIF and DIX.

dix=
turns on end-to-end data consistency checking in extended mode if set to 1, y, or Y (and off if set to 0, n, or N). The default is 0.

Specifying `dix=1` enables both DIF and DIX. Enabling `dix=` overrides specifications for `dif=`.

Note: End-to-end data consistency checking in extended mode is experimental and can cause errors if enabled.

port_scan_ratelimit=*<limit>*
sets the minimum delay, in milliseconds, between automatic port scans of your Linux instance. The default value is 60000 milliseconds. To turn off the rate limit, specify 0. Use this parameter to avoid frequent scans, while you still ensure that a scan is conducted eventually.

port_scan_backoff=*<delay>*
sets additional random delay, in milliseconds, in which the port scans of your Linux instance are spread. The default value is 500 milliseconds. To turn off the random delay, specify 0. In an installation with multiple Linux instances, use this attribute for every Linux instance to spread scans to avoid potential multiple simultaneous scans.

no_auto_port_rescan=
turns the automatic port rescan feature off (if set to 1, y, or Y) or on (if set to 0, n, or N). The default is 0. Automatic rescan is always performed when setting an adapter online and when user-triggered writes to the sysfs attribute port_rescan occur.

queue_depth=*<depth>*
specifies the number of commands that can be issued simultaneously to a SCSI device. The default is 32. The value you set here is used as the default queue depth for new SCSI devices. You can set the queue depth for each SCSI device using the queue_depth sysfs attribute, see “Setting the queue depth” on page 210.

device=*<device_bus_id>, <wwpn>, <fcp_lun>*

Attention: The `device=` module parameter is reserved for internal use. Do not use.

<device_bus_id>
specifies the FCP device through which the SCSI device is attached.

<wwpn>
specifies the target port through which the SCSI device is attached.

<fcp_lun>
specifies the LUN of the SCSI device.

ber_stop=

sets the mode of handling FCP devices for which the FCP channel reports a bit-error count in excess of its threshold.

If set to 1, y, or Y, the zfcp device driver shuts down such FCP devices; this is the default. If set to 0, n, or N, such FCP devices keep running and might cause I/O command timeouts with an associated performance degradation.

Kernel message "All paths over this FCP device are disused because of excessive bit errors" indicates that the zfcp device driver shut down a device because of bit errors. To resolve the problem, ensure that fibre optics on the local fibre link are clean and functional, and that all cables are properly plugged. Then recover the FCP device by writing `0` to its `failed` sysfs attribute, see "Recovering a failed FCP device" on page 191. If recovery through sysfs is not possible, set the CHPID of the device offline and back online on the Support Element.

Example

Use the following module parameter to enable end-to-end data consistency checking:

```
modprobe zfcp dif=1
```

Working with FCP devices

Set an FCP device online before you attempt to perform any other tasks.

Working with FCP devices comprises the following tasks:

- "Setting an FCP device online or offline" on page 185
- "Displaying FCP channel and device information" on page 187
- "Recovering a failed FCP device" on page 191
- "Obtaining diagnostic data for FCP channels" on page 192
- "Finding out whether NPIV is in use" on page 194
- "Logging I/O subchannel status information" on page 195

Setting an FCP device online or offline

In traditional LPAR mode, by default, FCP devices are offline. Set an FCP device online before you perform any other tasks.

About this task

As of z14 and LinuxONE II in DPM mode:

For Linux in a DPM partition, FCP devices are set online automatically, see Chapter 3, "Device auto-configuration for Linux in LPAR mode," on page 21.

DPM device auto-configuration with zfcp automatic LUN scan manages the zfcp configuration, including FCP devices, remote ports, and LUNs. This automation has the following requirements:

- The SAN switches must use single-initiator zoning.
- LUN masking (host mapping) must be in place on the storage systems.
- Both the SAN switches and the LUN masking must use host NPIV WWPNs for access control.

Attention: Use the procedure described here for dynamic testing of configuration settings. For persistent configuration in a production system, use one of the following options:

- The configuration files `/boot/loader/entries/*.conf` for FCP devices that are part of the root file system.
- The configuration file `/etc/zfcp.conf` for data disks.

An example of how to define an FCP device persistently is in the Red Hat installation information. For a general discussion of configuration files, see the configuration documentation on the Red Hat documentation website https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux.

You have the following options for configuring FCP LUNs to attach SCSI devices:

- During installation, use the **anaconda** GUI, the **dracut** boot parameter `rd.zfcp=`, or the **kickstart** parameter `zfcp`.

  Important: Linux on IBM Z kernel parameters in the boot configuration have a size limit of 895 bytes. Automatically generated rd.zfcp parameters might cause this limit to be exceeded. For details, see "Preventing excessive kernel parameters during installation" on page 562.
- On an installed system, use the **dracut** boot parameter `rd.zfcp=` only for SCSI disks that are required for the root file system. Use the configuration file `/etc/zfcp.conf` for all other SCSI devices, such as data volumes or tape libraries.

For details about configuration, see the Red Hat documentation website https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

You can always specify additional zfcp module parameters as explained in Chapter 4, "Kernel and module parameters," on page 25

Important: Configuration changes can directly or indirectly affect information that is required to mount the root file system. Such changes require an update of the initial RAM disk, followed by a re-write of the boot record (see "Rebuilding the initial RAM disk image" on page 96).

See "Working with newly available devices" on page 10 to avoid errors when you work with devices that have become available to a running Linux instance.

Setting an FCP device online registers it with the Linux SCSI stack. It also automatically runs the scan for ports in the SAN and waits for this port scan to complete. To check if setting the FCP device online was successful, you can use a script that first sets the FCP device online and after this operation completes checks if the WWPN of a remote storage port has appeared in sysfs.

When you set an FCP device offline, the port and LUN subdirectories are preserved. Setting an FCP device offline in sysfs interrupts the communication between Linux and the FCP channel. After a timeout has expired, the port and LUN attributes indicate that the ports and LUNs are no longer accessible. The transition of the FCP device to the offline state is synchronous, unless the device is disconnected.

For disconnected devices, writing 0 to the online sysfs attribute triggers an asynchronous deregistration process. When this process is completed, the device with its ports and LUNs is no longer represented in sysfs.

When the FCP device is set back online, the SCSI device names and minor numbers are freshly assigned. The mapping of devices to names and numbers might be different from what they were before the FCP device was set offline.

Procedure

There are two methods for setting an FCP device online or offline:

- Use the **chzdev** command with the `-a` option, or the **chccwdev** command ("chccwdev - Set CCW device attributes" on page 575).
- Alternatively, you can write 1 to an FCP device's online attribute to set it online, or 0 to set it offline.

Examples

- To set an FCP device with bus ID 0.0.3d0c online issue:

```
# chzdev -e -a zfcp-host 0.0.3d0c
```

or

```
# chccwdev -e 0.0.3d0c
```

or

```
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/online
```

- To set an FCP device with bus ID 0.0.3d0c offline issue:

```
# chzdev -d -a zfcp-host 0.0.3d0c
```

or

```
# chccwdev -d 0.0.3d0c
```

or

```
# echo 0 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/online
```

Displaying FCP channel and device information

For each online FCP device, there is a number of read-only attributes in sysfs that provide information about the corresponding FCP channel and FCP device.

Before you begin

The FCP device must be online for the FCP channel information to be valid.

About this task

The following tables summarize the relevant attributes.

Table 19. Attributes with Fibre Channel adapter hardware information

Attribute	Explanation
card_version	Version number that identifies a particular hardware feature. Same as `model` in Table 21 on page 188.
fc_security	IBM Fibre Channel Endpoint Security capabilities of the FCP channel. See "Investigating IBM Fibre Channel Endpoint Security" on page 217.
hardware_version	Number that identifies a hardware version for a particular feature. The initial hardware version of a feature is zero. This version indicator is increased only for hardware modifications of the same feature. Appending hardware_version to card_version results in a hierarchical version indication for a physical adapter. Same as `hardware_version` in Table 21 on page 188.
lic_version	Microcode level. Same as `firmware_version` in Table 21 on page 188.
peer_wwnn	WWNN of peer for a point-to-point connection.
peer_wwpn	WWPN of peer for a point-to-point connection.
peer_d_id	Destination ID of the peer for a point-to-point connection.

Table 20. Attributes with FCP device information

Attribute	Explanation
in_recovery	Shows if the FCP device is in recovery (0 or 1).

For the attributes availability, cmb_enable, and cutype, see “Device directories” on page 9. The status attribute is reserved.

Table 21. Relevant transport class attributes, fc_host attributes

Attribute	Explanation
fabric_name	Name of the attached fabric. The name is a 64-bit hexadecimal value. For z13 and z14 hardware with FICON Express16S or FICON Express16S+ features, this attribute contains valid fabric names only if the following minimum firmware level requirements are met. **z13** FICON Express16S: MCL P08424.005, LIC version 0x00000721 **z14** FICON Express16S: MCL P42611.008, LIC version 0x10200069 FICON Express16S+: MCL P42625.010, LIC version 0x10300147
firmware_version	Microcode level. Same as `lic_version` in Table 19 on page 187
hardware_version	Number that identifies a hardware version for a particular feature. The initial hardware version of a feature is zero. This version indicator is increased only for hardware modifications of the same feature. Appending hardware_version to card_version results in a hierarchical version indication for a physical adapter. Same as `hardware_version` in Table 19 on page 187
manufacturer	Manufacturer of the FCP channel. The value is "IBM".
maxframe_size	Maximum frame size.
model	Version number that identifies a particular hardware feature. Same as `card_version` in Table 19 on page 187
permanent_port_name	WWPN associated with the physical port of the FCP channel.
port_id	A unique ID (N_Port_ID) assigned by the fabric. In an NPIV setup, each virtual port is assigned a different port_id.
port_name	WWPN associated with the FCP device. If N_Port ID Virtualization is not available, the WWPN of the physical port (see permanent_port_name).
port_type	The port type indicates the topology of the port.
serial_number	The 32-byte serial number of the adapter hardware that provides the FCP channel.
speed	Speed of FC link.
supported_classes	Supported FC service class.
supported_speeds	Supported speeds.
symbolic_name	The symbolic port name that is registered with the FC name server.
tgid_bind_type	Target binding type.

Table 22. Relevant transport class attributes, fc_host statistics

Attribute	Explanation
reset_statistics	Writeable attribute to reset statistic counters.
seconds_since_last_reset	Seconds since last reset of statistic counters.
tx_frames	Transmitted FC frames.
tx_words	Transmitted FC words.
rx_frames	Received FC frames.
rx_words	Received FC words.
lip_count	Number of LIP sequences.
nos_count	Number of NOS sequences.
error_frames	Number of frames that are received in error.
dumped_frames	Number of frames that are lost because of lack of host resources.
link_failure_count	Link failure count.
loss_of_sync_count	Loss of synchronization count.
loss_of_signal_count	Loss of signal count.
prim_seq_protocol_err_count	Primitive sequence protocol error count.
invalid_tx_word_count	Invalid transmission word count.
invalid_crc_count	Invalid CRC count.
fcp_input_requests	Number of FCP operations with data input.
fcp_output_requests	Number of FCP operations with data output.
fcp_control_requests	Number of FCP operations without data movement.
fcp_input_megabytes	Megabytes of FCP data input.
fcp_output_megabytes	Megabytes of FCP data output.

Procedure

Use the **cat** command to read an attribute.

- Issue a command of this form to read an attribute:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<attribute>
```

  where:

  <device_bus_id>
  specifies an FCP device that corresponds to the FCP channel.

  <attribute>
  is one of the attributes in Table 19 on page 187 or Table 20 on page 188.

- To read attributes of the associated Fibre Channel host use:

```
# cat /sys/class/fc_host/<host_name>/<attribute>
```

  where:

<host_name>
is the ID of the Fibre Channel host.

<attribute>
is one of the attributes in Table 21 on page 188.

- To read statistics attributes of the FCP channel associated with this Fibre Channel host, use:

```
# cat /sys/class/fc_host/<host_name>/statistics/<attribute>
```

where:

<host_name>
is the ID of the Fibre Channel host.

<attribute>
is one of the attributes in Table 22 on page 189.

Examples

- In this example, information is displayed about an FCP channel that corresponds to an FCP device with bus ID 0.0.3d0c:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/hardware_version
0x00000000
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/lic_version
0x16500124
```

- Alternatively you can use **lszfcp** (see “lszfcp - List zfcp devices” on page 693) to display attributes of an FCP channel:

```
# lszfcp -b 0.0.3d0c -a
0.0.3d0c host0
Bus = "ccw"
    availability        = "good"
    card_version        = "0x000b"
    cmb_enable          = "0"
    cutype              = "1731/03"
    devtype             = "1732/03"
    diag_max_age        = "5000"
    failed              = "0"
    fc_security         = "Authentication, Encryption"
    hardware_version    = "0x00000000"
    in_recovery         = "0"
    lic_version         = "0x24500099"
    modalias            = "ccw:t1731m03dt1732dm03"
    online              = "1"
    peer_d_id           = "0x000000"
    peer_wwnn           = "0x0000000000000000"
    peer_wwpn           = "0x0000000000000000"
    status              = "0x5400040b"
    uevent              = "DRIVER=zfcp"
Class = "fc_host"
    active_fc4s         = "0x00 0x00 0x01 0x00 ..."
    dev_loss_tmo        = "60"
    fabric_name         = "0x100000051e4a8f00"
    firmware_version    = "0x24500099"
    hardware_version    = "0x00000000"
    manufacturer        = "IBM"
    maxframe_size       = "2112 bytes"
    model               = "0x000b"
    node_name           = "0x5005076400c7ec87"
    permanent_port_name = "0xc05076ffd6801981"
    port_id             = "0x671a29"
    port_name           = "0xc05076ffd6801e10"
    port_state          = "Online"
    port_type           = "NPIV VPORT"
    serial_number       = "IBM02000000007EC87"
    speed               = "32 Gbit"
    supported_classes   = "Class 2, Class 3"
    supported_fc4s      = "0x00 0x00 0x01 0x00 ..."
    supported_speeds    = "8 Gbit, 16 Gbit, 32 Gbit"
    symbolic_name       = "IBM type serial PCHID: 0198 NPIV UlpId: 05600300 DEVNO: 0.0.3d0c NAME:
hostname.domain"
    tgtid_bind_type     = "wwpn (World Wide Port Name)"
Class = "scsi_host"
    active_mode         = "Initiator"
    can_queue           = "4096"
    cmd_per_lun         = "0"
    eh_deadline         = "off"
    host_busy           = "0"
    megabytes           = "15 0"
    proc_name           = "zfcp"
    prot_capabilities   = "0"
    prot_guard_type     = "0"
    queue_full          = "0 2357653"
    requests            = "815 0 66"
    seconds_active      = "11"
    sg_prot_tablesize   = "0"
    sg_tablesize        = "574"
    state               = "running"
    supported_mode      = "Initiator"
    unchecked_isa_dma   = "0"
    unique_id           = "6400"
    use_blk_mq          = "1"
    utilization         = "0 0 0"
```

Recovering a failed FCP device

Failed FCP devices are automatically recovered by the zfcp device driver. You can read the `in_recovery` attribute to check whether recovery is under way.

Before you begin

The FCP device must be online.

Procedure

Perform these steps to find out the recovery status of an FCP device and, if needed, start or restart recovery:

1. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/in_recovery
```

   The value is `1` if recovery is under way and `0` otherwise. If the value is `0` for a non-operational FCP device, recovery might have failed. Alternatively, the device driver might have failed to detect that the FCP device is malfunctioning.
2. To find out whether recovery failed, read the `failed` attribute. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/failed
```

   The value is `1` if recovery failed and `0` otherwise.
3. You can start or restart the recovery process for the FCP device by writing `0` to the `failed` attribute. Issue a command of this form:

```
# echo 0 > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/failed
```

Example

In the following example, an FCP device with a device bus-ID 0.0.3d0c is malfunctioning. The first command reveals that recovery is not already under way. The second command manually starts recovery for the FCP device:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/in_recovery
0
# echo 0 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/failed
```

Obtaining diagnostic data for FCP channels

Diagnostic data about FCP channels is available as of z14 and FICON Express8S.

About this task

FCP channel diagnostic data is available through sysfs attributes in `/sys/bus/ccw/drivers/zfcp/<device_bus_id>/diagnostics`, where *<device_bus_id>* is the device-bus ID of the FCP device that corresponds to the FCP channel.

After the diagnostic data is retrieved from the FCP channel, it is cached for a specific expiration interval, 5 seconds by default. Reading attributes within this interval results in the cached values. If you read attributes after the cache has expired, current values are retrieved from the FCP channel. You can use the `diag_max_age` sysfs attribute of the FCP device to change the expiration interval.

The following table summarizes the available attributes with diagnostic data.

Table 23. Attributes with diagnostic data about FCP channels

Attribute	Explanation
sfp_invalid	Flag that indicates whether the attributes with physical properties of the FCP channel provide valid (0) or useless (1) data. These attributes are: temperature, vcc, tx_bias, tx_power, and rx_power.
temperature	Temperature of the transceiver. The value is a signed integer in units of 1/256 ℃. For example, interpret 1024 as 4 ℃.
vcc	Supply voltage of the transceiver. The value is in units of 100 µV.

Table 23. Attributes with diagnostic data about FCP channels (continued)

Attribute	Explanation
tx_bias	Bias current of the transmitter laser. The value is in units of 2 µA.
tx_power	Coupled output power of the transmitter laser. The value is in units of 0.1 µW.
rx_power	Optical power that is measured at the receiving element. The value is in units of 0.1 µW.
optical_port	Flag that indicates whether the transceiver uses an optical element (1) or does not use an optical element (0).
fec_active	Flag that indicates whether forward error correction (FEC) is active (1) or inactive (0).
port_tx_type	Type of the transmitting element. Possible values are: **0** Unknown **1** Short wave **2** Long wave, LC 1310 nm **3** Long wave, LL 1550 nm
connector_type	Connector type. Possible values are: **0** Unknown **1** SFP+
b2b_credit	Number of buffers available for receiving Class 2, or Class 3 frames on the local FC port.

Procedure

1. Optional: Verify the validity of the data points with physical properties of the FCP channel.

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/diagnostics/sfp_invalid
```

   For valid data, the attribute value must be 0.
2. Optional: Adjust the expiration interval of the cached diagnostic data.
   a) Read the current expiration interval from the `diag_max_age` attribute.

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/diag_max_age
```

      The attribute value is the expiration interval in milliseconds.
   b) Write the value, in milliseconds, of the new expiration interval to the `diag_max_age` attribute.

```
# echo <expiration_interval> > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/diag_max_age
```

   Setting the expiration interval to 0 disables caching. The combination of short expiration intervals or disabling caching and frequent reading of diagnostic data can adversely affect performance.
3. Read the data of interest.

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/diagnostics/<attribute>
```

Where *<attribute>* is one of the attributes of Table 23 on page 192.

Examples

- In this example, the expiration interval for an FCP device that corresponds to bus ID 0.0.3d5c is changed from 5 seconds to 10 seconds.

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d5c/diag_max_age
5000
# echo 10000 > /sys/bus/ccw/drivers/zfcp/0.0.3d5c/diag_max_age
```

- In this example, the first command confirms that the attributes with physical properties of the FCP channel contain valid data. The next commands display data about the transceiver temperature, supply voltage, and type of the transmitting element.

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d5c/diagnostics/sfp_invalid
0
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d5c/diagnostics/temperature
7822
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d5c/diagnostics/vcc
33000
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d5c/diagnostics/connector_type
1
```

The 0 returned from the first command confirms that the physical data is valid. The transceiver temperature is 30.55 ℃ (7822 / 256) and the supply voltage is 3.3000 V (33000 × 10^{-4}). The `connector_type` is SFP+ (value 1).

Finding out whether NPIV is in use

An FCP device runs in NPIV mode if the `port_type` attribute of the FCP device attribute contains the string "NPIV". Alternatively, if the applicable `permanent_port_name` and `port_name` are not the same and are not NULL.

Procedure

Read the `port_type` attribute of the FCP device.

Issue a command of the following form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<host_no>/fc_host/<host_no>/port_type
NPIV VPORT
```

For example:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/host0/fc_host/host0/port_type
NPIV VPORT
```

Alternatively, compare the values of the `permanent_port_name` attribute and the `port_name`.

Tip: You can use **`lszfcp`** (see “lszfcp - List zfcp devices” on page 693) to list the FCP device attributes.

Example

```
# lszfcp -b 0.0.3d0c -a
0.0.3d0c host0
Bus = "ccw"
    availability        = "good"
    ...
Class = "fc_host"
    ...
    node_name           = "0x5005076400c829e7"
    permanent_port_name = "0xc05076ffeb001201"
    port_id             = "0x67e35d"
    port_name           = "0xc05076ffeb001b48"
    port_state          = "Online"
    port_type           = "NPIV VPORT"
    ...
    symbolic_name       = "IBM type serial  PCHID: 0120 NPIV UlpId: 02600F18   DEVNO: 0.0.3d0c NAME: hostname.domain"
    ...
```

The `port_type` attribute directly indicates that NPIV is used. The example also shows that `permanent_port_name` is different from `port_name` and neither is NULL. The example also shows the `symbolic_name` attribute that shows the symbolic port name that was registered on the FC name server.

Logging I/O subchannel status information

When severe errors occur for an FCP device, the FCP device driver triggers a set of log entries with I/O subchannel status information.

The log entries are available through the SE Console Actions Work Area with the View Console Logs function. In the list of logs, these entries have the prefix `1F00`. The content of the entries is intended for support specialists.

Working with target ports

You can scan for ports, display port information, recover a port, or remove a port.

Working with target ports comprises the following tasks:

- “Scanning for ports” on page 195
- “Controlling automatic port scanning” on page 196
- “Displaying port information” on page 198
- “Recovering a failed port” on page 200
- “Removing ports” on page 201

Scanning for ports

Newly available target ports are discovered. However, you might want to trigger a port scan to re-create accidentally removed port information or to assure that all ports are present.

Before you begin

The FCP device must be online.

About this task

The zfcp device driver automatically adds port information to sysfs when:

- The FCP device is set online
- Target ports are added to the Fibre Channel fabric, unless the module parameter `no_auto_port_rescan` is set to 1. See “Setting up the zfcp device driver” on page 182.

Scanning for ports might take some time to complete. Commands that you issue against ports or LUNs while scanning is in progress are delayed and processed when port scanning is completed.

Use the `port_rescan` attribute if a remote storage port was accidentally deleted from the adapter configuration or if you are unsure whether all ports were added to sysfs.

Procedure

Issue a command of this form:

```
# echo 1 > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/port_rescan
```

where *<device_bus_id>* specifies the FCP device through which the target ports are attached.

Tip: List the contents of `/sys/bus/ccw/drivers/zfcp/<device_bus_id>` to find out which ports are currently configured for the FCP device.

Example

In this example, a port with WWPN 0x500507630303c562 is already configured for an FCP device with bus ID 0.0.3d0c. An additional target port with WWPN 0x500507630300c562 is automatically configured by triggering a port scan.

```
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/port_rescan
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
0x500507630300c562
```

Controlling automatic port scanning

Automatic port scanning includes two zfcp parameters that improve the behaviour of Linux instances in SANs. These zfcp parameters are set to default values that work well for most installations. If needed, you can fine-tune the frequency and timing of automatic port scans with the zfcp parameters `port_scan_backoff` and `port_scan_ratelimit`. You can enable automatic port scanning with the zfcp parameter `no_auto_port_rescan=0`. This value is the default.

About this task

In a large installation, where many Linux instances receive the same notifications of SAN changes, multiple instances might trigger scans simultaneously and too frequently. See Figure 50 on page 196



Figure 50. Numerous port scans in a Linux installation

These scans might put unnecessary load on the name server function of fabric switches and potentially result in late or inconclusive results.

You can avoid excessive scanning, yet still ensure that a port scan is eventually conducted. You can control port scanning with the zfcp parameters:

port_scan_ratelimit
sets the minimum delay, in milliseconds, between automatic port scans of your Linux instance. The default value is 60000 milliseconds. To turn off the rate limit, specify 0.

port_scan_backoff
sets an additional random delay, in milliseconds, in which the port scans of your Linux instance are spread. In an installation with multiple Linux instances, use this zfcp parameter for every Linux instance to spread scans to avoid potential multiple simultaneous scans. The default value is 500 milliseconds. To turn off the random delay, specify 0.

Use module parameters (see "Setting up the zfcp device driver" on page 182). On a running Linux system, you can also query or set these values by using the sysfs attributes with the same names.

Using `port_scan_ratelimit` reduces the number of scans, as shown in Figure 51 on page 197



Figure 51. Port scan behavior with scan rate limit.

However, if the rate limit is set to the same value, the scans can still occur almost simultaneously, as for FCP device A and B in Linux 1.

Using `port_scan_backoff` and `port_scan_ratelimit` together delays port scans even further and avoids simultaneous scans, as shown in Figure 52 on page 197. In the figure, FCP devices A and B in Linux 1 have the same rate limit and the same backoff values. The random element in the backoff value causes the scans to occur at slightly different times.



Figure 52. Port scan behavior with backoff and scan rate limit.

Procedure

Use `port_scan_backoff` and `port_scan_ratelimit` together or separately to tune the behavior of port scanning:

- To avoid too frequent scanning, set a minimum wait time between two consecutive scans for the same Linux instance. Use the `port_scan_ratelimit` sysfs attribute.

  By default, `port_scan_ratelimit` is turned on and has a value of 60000 milliseconds.

  For example, to specify an attribute value of 12 seconds, issue:

  ```
  # echo 12000 > /sys/module/zfcp/parameters/port_scan_ratelimit
  ```

- To further spread scans over a certain time and thus avoid multiple simultaneous scans, set the `port_scan_backoff` sysfs attribute.

  By default, `port_scan_backoff` is turned on and has a value of 500 milliseconds.

For example, to query the setting, issue a command of this form:

```
# cat /sys/module/zfcp/parameters/port_scan_backoff
500
```

To set the attribute to 1 second, issue:

```
# echo 1000 > /sys/module/zfcp/parameters/port_scan_backoff
```

Results

The automatic port scans are delayed by the values specified. If a SAN notification is received during the rate limit time, a port scan is conducted immediately after the delay time passed.

Setting the attributes in sysfs is a useful method on a running system where you want to make dynamic changes. If you want to make the changes persistent across IPLs, use the kernel or module parameter.

Depending on the port event, one or more of the three zfcp parameters are evaluated to schedule a port scan. For example, port scans that are triggered manually through sysfs are not delayed. Table 24 on page 198 shows which events evaluate which zfcp parameters.

Table 24. Port events and their use of port scanning zfcp parameters

zfcp parameter	**no_auto_port_rescan**	**port_scan_backoff**	**port_scan_ratelimit**
Event			
FCP device resume	Yes	Yes	No
User sets FCP device online	No	Yes	No
User initiates a port scan	No	No	No
User starts FCP device recovery	Yes	Yes	Yes
Automatic FCP device recovery	Yes	Yes	Yes
SAN change notification	Yes	Yes	Yes

Displaying port information

For each target port, there is a number of read-only sysfs attributes with port information.

About this task

Table 25 on page 198 and Table 26 on page 199 summarize the relevant attributes.

Table 25. zfcp-specific attributes with port information within the FCP device sysfs tree

Attribute	**Explanation**
access_denied	This attribute is obsolete. The value is always 0.
fc_security	IBM Fibre Channel Endpoint Security status of the connection between an FCP device and the port. See “Investigating IBM Fibre Channel Endpoint Security” on page 217.
in_recovery	Shows if port is in recovery (0 or 1).

Table 26. Transport class attributes with port information

Attribute	Explanation
node_name	WWNN of the remote port (target port).
port_name	WWPN of remote port.
port_id	Destination ID of remote port.
port_state	State of remote port.
roles	Role of remote port (usually FCP target).
scsi_target_id	Linux SCSI ID of remote port.
supported_classes	Supported classes of service.

Procedure

Use the **cat** command to read an attribute.

- Issue a command of this form to read a zfcp-specific attribute:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<attribute>
```

  where:

  <device_bus_id>
  specifies the FCP device.

  <wwpn>
  is the WWPN of the target port.

  <attribute>
  is one of the attributes in Table 25 on page 198.

- To read transport class attributes of the associated target port, use a command of this form:

```
# cat /sys/class/fc_remote_ports/<rport_name>/<attribute>
```

  where:

  <rport_name>
  is the name of the remote port.

  <attribute>
  is one of the attributes in Table 26 on page 199.

Tip: With the HBA API package installed, you can also use the **zfcp_ping** and **zfcp_show** commands to find out more about your ports. See "Tools for investigating your SAN configuration" on page 221.

Examples

- In this example, information is displayed for a target port 0x500507630300c562 that is attached through an FCP device with bus ID 0.0.3d0c:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/in_recovery
0
```

- To display transport class attributes of a target port you can use **lszfcp**:

```
# lszfcp -p 0x500507630300c562 -a
0.0.3d0c/0x500507630300c562 rport-0:0-0
Class = "fc_remote_ports"
    dev_loss_tmo        = "2147483647"
    fast_io_fail_tmo    = "5"
    maxframe_size       = "2048 bytes"
    node_name           = "0x5005076303ffc562"
    port_id             = "0x652113"
    port_name           = "0x500507630300c562"
    port_state          = "Online"
    roles               = "FCP Target"
    scsi_target_id      = "0"
    supported_classes   = "Class 2, Class 3"
```

Recovering a failed port

Failed target ports are automatically recovered by the zfcp device driver. You can read the `in_recovery` attribute to check whether recovery is under way.

Before you begin

The FCP device must be online.

Procedure

Perform these steps to find out the recovery status of a port and, if needed, start or restart recovery:

1. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/in_recovery
```

   where:

   <device_bus_id>
   specifies the FCP device.

   <wwpn>
   is the WWPN of the target port.

   The value is `1` if recovery is under way and `0` otherwise. If the value is `0` for a non-operational port, recovery might have failed or the device driver might have failed to detect that the port is malfunctioning.

2. To find out whether recovery failed, read the `failed` attribute. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/failed
```

   The value is `1` if recovery has failed and `0` otherwise.

3. You can start or restart the recovery process for the port by writing `0` to the `failed` attribute. Issue a command of this form:

```
# echo 0 > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/failed
```

Example

In the following example, a port with WWPN 0x500507630300c562 that is attached through an FCP device with bus ID 0.0.3d0c is malfunctioning. The first command reveals that recovery is not already under way. The second command manually starts recovery for the port:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/in_recovery
0
# echo 0 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/failed
```

Removing ports

Removing unused ports can save FCP channel resources. Additionally setting the `no_auto_port_rescan` attribute avoids unnecessary attempts to recover unused remote ports.

Before you begin

The FCP device must be online.

About this task

List the contents of `/sys/bus/ccw/drivers/zfcp/<device_bus_id>` to find out which ports are currently configured for the FCP device.

You cannot remove a port while SCSI devices are configured for it (see "Configuring SCSI devices" on page 202) or if the port is in use, for example, by error recovery.

Note: The next port scan will attach all available ports, including any previously removed ports. To prevent removed ports from being reattached automatically, use zoning or the no_auto_port_rescan module parameter, see "Setting up the zfcp device driver" on page 182.

Procedure

Issue a command of this form:

```
# echo <wwpn> > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/port_remove
```

where:

<device_bus_id>
specifies the FCP device.

<wwpn>
is the WWPN of the port to be removed.

Example

In this example, two ports with WWPN 0x500507630303c562 and 0x500507630300c562 are configured for an FCP device with bus ID 0.0.3d0c. The port with WWPN 0x500507630303c562 is then removed.

```
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
0x500507630300c562
# echo 0x500507630303c562 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/port_remove
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630300c562
```

Working with SCSI devices

In an NPIV setup with auto lun scan, the SCSI devices are configured automatically. Otherwise, you must configure FCP LUNs to obtain SCSI devices. In both cases, you can configure SCSI devices, display information, and remove SCSI devices.

Working with SCSI devices comprises the following tasks:

- "Updating the information about SCSI devices" on page 212
- "Setting the SCSI command timeout" on page 213
- "Controlling the SCSI device state" on page 214
- "Removing SCSI devices" on page 214

Configuring SCSI devices

FCP devices that use NPIV mode detect the LUNs automatically and no configuring is necessary. If needed, configure the LUN manually.

For each FCP device that uses NPIV mode and if you did not disable automatic LUN scanning (see "Setting up the zfcp device driver" on page 182), the LUNs are configured for you. In this case, *no* FCP LUN entries are created under `/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>`.

To find out whether an FCP device is using NPIV mode, check the `port_type` attribute, for example:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.1901/host*/fc_host/host*/port_type
NPIV VPORT
```

To find out whether automatic LUN scanning is enabled, check the current setting of the module parameter zfcp.allow_lun_scan. The example below shows automatic LUN scanning as turned on.

```
# cat /sys/module/zfcp/parameters/allow_lun_scan
Y
```

Important: Configuration changes can directly or indirectly affect information that is required to mount the root file system. Such changes require an update of the initial RAM disk, followed by a re-write of the boot record (see "Rebuilding the initial RAM disk image" on page 96).

Automatically attached SCSI devices

FCP devices that use NPIV mode detect the LUNs automatically and no configuring is necessary. In this case, *no* FCP LUN entries are created under `/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>`.

What to do next

To check whether a SCSI device is registered, check for a directory with the name of the LUN in `/sys/bus/scsi/devices`. If there is no SCSI device for this LUN, the LUN is not valid in the storage system, or the FCP device is offline in Linux.

Manually configured FCP LUNs and their SCSI devices

For FCP devices that do not use NPIV mode, or if automatic LUN scanning is disabled, FCP LUNs must be configured manually to obtain SCSI devices.

Before you begin

Attention: Use this procedure only to dynamically test configuration settings.

You have the following options for configuring FCP LUNs to attach SCSI devices:

- During installation, use the **anaconda** GUI, the **dracut** boot parameter `rd.zfcp=`, or the **kickstart** parameter `zfcp`.

  Important: Linux on IBM Z kernel parameters in the boot configuration have a size limit of 895 bytes. Automatically generated rd.zfcp parameters might cause this limit to be exceeded. For details, see "Preventing excessive kernel parameters during installation" on page 562.

- On an installed system, use the **dracut** boot parameter `rd.zfcp=` only for SCSI disks that are required for the root file system. Use the configuration file `/etc/zfcp.conf` for all other SCSI devices, such as data volumes or tape libraries.

For details about configuration, see the Red Hat documentation website https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

You can always specify additional zfcp module parameters as explained in Chapter 4, “Kernel and module parameters,” on page 25

Procedure

If your FCP device does not use NPIV mode, or if you have disabled automatic LUN scanning, proceed as follows:

- Use the **chzdev** command.

```
# chzdev -e -a zfcp-lun <device_bus_id>:<wwpn>:<fcp_lun>
```

  where:

  <fcp_lun>
  : is the LUN of the SCSI device to be configured. The LUN is a 16 digit hexadecimal value padded with zeroes, for example 0x4010403300000000.

  <device_bus_id>
  : specifies the FCP device.

  <wwpn>
  : is the WWPN of the target port.

- Alternatively, using sysfs, write the device's LUN to the port's `unit_add` attribute. Issue a command of this form:

```
# echo <fcp_lun> > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/unit_add
```

Results

This command starts a process with multiple steps:

1. It creates a directory in `/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>` with the LUN as the directory name. The directory is part of the list of all LUNs to configure. Without NPIV or with auto LUN scanning disabled, zfcp registers only FCP LUNs contained in this list with the Linux SCSI stack in the next step.
2. It initiates the registration of the SCSI device with the Linux SCSI stack. The FCP device must be online for this step.
3. It waits until the Linux SCSI stack registration completes successfully or returns an error. It then returns control to the shell. A successful registration creates a sysfs entry in the SCSI branch (see “Mapping the representations of a SCSI device in sysfs” on page 204).

Example

Using **chzdev**: In this example, an FCP device with bus ID 0.0.3d0c is enabled. The WWPN of the target port is 0x500507630300c562. A SCSI device with LUN 0x4010403300000000 is added to the port.

```
# chzdev -e -a zfcp-lun 0.0.3d0c:0x500507630300c562:0x4010403300000000
```

Alternatively, using sysfs: In this example, a target port with WWPN 0x500507630300c562 is attached through an FCP device with bus ID 0.0.3d0c. A SCSI device with LUN 0x4010403200000000 is already configured for the port. An additional SCSI device with LUN 0x4010403300000000 is added to the port.

```
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x*
0x4010403200000000
# echo 0x4010403300000000 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/unit_add
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x*
0x4010403200000000
0x4010403300000000
```

What to do next

To check whether a SCSI device is registered for the configured LUN, check for a directory with the name of the LUN in `/sys/bus/scsi/devices`. If there is no SCSI device for this LUN, the LUN is not valid in the storage system, or the FCP device is offline in Linux.

To see which LUNs are currently configured for the port, list the contents of `/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>`.

Mapping the representations of a SCSI device in sysfs

Each SCSI device that is configured is represented by multiple directories in sysfs, in particular, within the SCSI branch. Only manually configured LUNs are also represented within the zfcp branch.
You can find the FCP device bus-ID, the target WWPN, and the FCP LUN triplet that corresponds to a SCSI device in two ways: By traversing the sysfs directory tree or by using commands.

Note: The zfcp-specific sysfs attributes `hba_id`, `wwpn`, and `fcp_lun` are deprecated. Use the methods described here instead to find the addressing of a SCSI device.

About this task

The directory in the sysfs SCSI branch has the following form:

```
/sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>
```

where:

<scsi_host_no>
is the SCSI host number that corresponds to the FCP device.

<scsi_id>
is the SCSI ID of the target port.

<scsi_lun>
is the LUN of the SCSI device.

The value for *<scsi_lun>* depends on the storage device. Often, it is a single-digit number, but for some storage devices it has numerous digits.

For manually configured FCP LUNs, see “Manually configured FCP LUNs and their SCSI devices” on page 202 for details about the directory in the zfcp branch.

Before you begin

You must identify the SCSI device in sysfs. For example, use **`readlink`** to find the path in sysfs with all symbolic links resolved:

```
# readlink -e /sys/bus/scsi/devices/2:0:1:1074741413
/sys/devices/css0/0.0.000a/0.0.1800/host2/rport-2:0-1/target2:0:1/2:0:1:1074741413
```

Using sysfs

Note: Do not assume a stable sysfs structure. The following procedure accommodates changes in sysfs.

This example shows how you can traverse the directory tree to find the FCP device bus-ID, the target WWPN, and the FCP LUN that correspond to a SCSI device name. The example assumes:

SCSI device
2:0:1:1074741413

FCP LUN
0x40a5400f00000000

target WWPN
0x50050763030bd327

FCP device bus-ID
0.0.1800

1. Obtain the hexadecimal FCP LUN.

   a. Start at the SCSI device directory or anywhere in the subtree below the SCSI device. Ascend the sysfs tree until you find the SCSI device. To do this, test every subdirectory for a symbolic link named "subsystem" that points to a relative directory path whose last entry is `scsi`. Search for the symbolic link named "subsystem":

   ```
   # ls -dl subsystem
   lrwxrwxrwx 1 root root 0 Oct 19 16:08 subsystem -> ../../../../../../../../bus/scsi
   ```

   The subsystem symbolic link points to a directory tree where the last subdirectory is `scsi`.

   b. Confirm that this is a SCSI device by reading the DEVTYPE line within the uevent attribute. The value must be "scsi_device".

   ```
   # grep "^DEVTYPE=" uevent
   DEVTYPE=scsi_device
   ```

   The last part of the current directory name is then the decimal SCSI LUN, for example, assuming you have found this directory:

   ```
   # pwd
   /sys/devices/css0/0.0.000a/0.0.1800/host2/rport-2:0-1/target2:0:1/2:0:1:1074741413
   ```

   Here, the SCSI LUN is 1074741413.

   c. Transform the SCSI LUN to the FCP LUN as follows:

Step	Example
Take decimal LUN in decimal notation:	1074741413
Convert to hexadecimal notation:	0x400f40a5
Pad with 0 from the left to obtain a 64-bit value:	0x00000000400f40a5
Divide into 16-bit blocks (LUN levels):	0x0000|0000|400f|40a5
Reverse the order of the blocks:	0x40a5|400f|0000|0000
The resulting hexadecimal number is the FCP LUN:	0x40a5400f00000000

   The Linux kernel function **`int_to_scsilun()`** in `drivers/scsi/scsi_common.c` converts a decimal SCSI LUN to obtain the hexadecimal FCP LUN according to this algorithm. The conversion works in both directions.

   d. Confirm that the path includes a directory "rport-*<no>*". For example, assuming you have found this directory:

   ```
   # pwd
   /sys/devices/css0/0.0.000a/0.0.1800/host2/rport-2:0-1/target2:0:1/2:0:1:1074741413
   ```

   If there is no rport directory, the transport is not fibre channel and thus not zfcp-related. Abandon the search.

Table 27 on page 206 lists the libudev functions that you can use instead of manually traversing the sysfs.

Table 27. Useful udev functions

Name	Task
udev_device_get_parent()	Ascend the sysfs tree.
udev_device_get_subsystem()	Retrieve subsystem name.
udev_device_get_devtype()	Retrieve device type.
udev_device_get_syspath()	Check if `rport` is a subdirectory.

2. Obtain the target WWPN.
   a. Continue ascending the sysfs tree the same way until you find the SCSI target. To do this, test every subdirectory for a symbolic link named "subsystem" that points to a relative directory path whose last entry is `scsi`. Search for the symbolic link named "subsystem":

```
# ls -dl subsystem
lrwxrwxrwx 1 root root 0 Oct 19 16:08 subsystem -> ../../../../../../../bus/scsi
```

   b. Confirm that this is a SCSI target by reading the DEVTYPE line within the uevent attribute. The value must be "scsi_target".

```
# grep "^DEVTYPE=" uevent
DEVTYPE=scsi_target
```

      For example, assuming you have found this directory:

```
# pwd
/sys/devices/css0/0.0.000a/0.0.1800/host2/rport-2:0-1/target2:0:1
```

   c. The SCSI target has a subdirectory `fc_transport`. Descend this subtree until you find a subdirectory that matches the SCSI target name. In this example, you would descend to `fc_transport/target2:0:1`.
   d. In the found target, read the `port_name` attribute:

```
# cat port_name
0x50050763030bd327
```

      The value of the `port_name` is the target WWPN.

Table 28. Useful udev functions

Name	Task
udev_device_get_parent_with_subsystem_devtype(dev, "scsi", "scsi_target")	Find the SCSI target.
udev_device_new_from_subsystem_sysname (udev_device_get_udev(scsidev), "fc_transport", udev_device_get_sysname(targetdev))	Find a matching target in the `fc_transport` branch.
udev_device_get_sysattr_value()	Read the port_name attribute.

3. Obtain the FCP device-bus ID. Keep ascending the sysfs tree. Search for the symbolic link "subsystem" that points to a relative path where the last subdirectory is `ccw`.

   For example:

```
# ls -dl subsystem
lrwxrwxrwx 1 root root 0 Oct 19 16:08 subsystem -> ../../../../bus/ccw
```

Then the name of the last directory in the current path is the FCP device-bus ID, for example:

```
# pwd
/sys/devices/css0/0.0.000a/0.0.1800
```

Here, 0.0.1800 is the FCP device-bus ID.

Using commands

To map a SCSI device name to its corresponding FCP device bus-ID, target WWPN, and LUN, you can use one of the following commands. The example assumes:

SCSI device
2:0:1:1074741413

FCP LUN
0x40a5400f00000000

target WWPN
0x50050763030bd327

FCP device bus-ID
0.0.1800

- Use the **lszfcp** with the `-D` option to list the FCP device-bus ID, the target WWPN, and the FCP LUN for all SCSI devices. For example:

```
lszfcp -D
....
0.0.1800/0x50050763030bd327/0x40a5400f00000000 2:0:1:1074741413
....
```

For details about the **lszfcp** command, see "lszfcp - List zfcp devices" on page 693.

- Use the **lszdev** command on device type zfcp-lun devices, and display the `ID` and `ATTR:scsi_dev` columns. For example:

```
# lszdev zfcp-lun -a -c ID,ATTR:scsi_dev
ID                                             ATTR:scsi_dev
...
0.0.1800:0x50050763030bd327:0x40a5400f00000000 2:0:1:1074741413
...
```

For details about the **lszdev** command, see "lszdev - Display IBM Z device configurations" on page 688.

- Use the **lsscsi** command with the `--transport` and `--lunhex` options in verbose mode to get information about a SCSI device:

```
# lsscsi -xxtv
[2:0:1:0x40a5400f00000000]  disk    fc:0x50050763030bd327,0x249900   /dev/sda
  dir: /sys/bus/scsi/devices/2:0:1:1074741413  [/sys/devices/css0/0.0.000a/0.0.1800/host2
                                                /rport-2:0-1/target2:0:1/2:0:1:1074741413]
...
```

For details about the **lsscsi** command, see the man page.

Note: The details of the command output is subject to change. Do not rely on the output always being exactly as shown.

Figure 53 on page 208 illustrates the sysfs structure of a SCSI device and how it corresponds to the **lszfcp** command output.

Warning: Do not rely on the sysfs structure in the example. The sysfs structure changes without notice.



Figure 53. Example SCSI device in sysfs and command output

Displaying information about SCSI devices

For each SCSI device, there is a number of read-only attributes in sysfs that provide information for the device.

About this task

Table 29 on page 209 summarizes the read-only attributes for manually configured FCP LUNs, including those attributes that indicate whether the device access is restricted by access control software on the FCP channel. These attributes can be found in the zfcp branch of sysfs. The path has the form:

```
/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<fcp_lun>/<attribute>
```

Table 29. Attributes of manually configured FCP LUNs with device access information

Attribute	Explanation
access_denied	Flag that indicates whether access to the device is restricted by the FCP channel. The value is 1 if access is denied and 0 if access is permitted. If access is denied to your Linux instance, confirm that your SCSI devices are configured as intended. Also, be sure that you really want to share a SCSI device. For shared access to a SCSI device, preferably use NPIV (see "N_Port ID Virtualization for FCP channels" on page 181). You might also use different FCP channels or target ports.
access_shared	This attribute is obsolete. The value is always 0.
access_readonly	This attribute is obsolete. The value is always 0.
in_recovery	Shows if unit is in recovery (0 or 1)

Table 30 on page 209 lists further read-only attributes with information about the SCSI device. These attributes can be found in the SCSI branch of sysfs. The path has the form:

```
/sys/class/scsi_device/<device_name>/device/<attribute>
```

Table 30. SCSI device class attributes

Attribute	Explanation
device_blocked	Flag that indicates whether the device is in blocked state (0 or 1).
iocounterbits	The number of bits used for I/O counters.
iodone_cnt	The number of completed or rejected SCSI commands.
ioerr_cnt	The number of SCSI commands that completed with an error.
iorequest_cnt	The number of issued SCSI commands.
model	The model of the SCSI device, received from inquiry data.
rev	The revision of the SCSI device, received from inquiry data.
scsi_level	The SCSI revision level, received from inquiry data.
type	The type of the SCSI device, received from inquiry data.
vendor	The vendor of the SCSI device, received from inquiry data.
zfcp_access_denied	Flag that indicates whether access to the device is restricted by the FCP channel. The value is 1 if access is denied and 0 if access is permitted. If access is denied to your Linux instance, confirm that your SCSI devices are configured as intended. Also, be sure that you really want to share a SCSI device. For shared access to a SCSI device, preferably use NPIV (see "N_Port ID Virtualization for FCP channels" on page 181). You might also use different FCP channels or target ports.
zfcp_in_recovery	Shows if unit is in recovery (0 or 1).

Procedure

Use the **lszfcp** command (see "lszfcp - List zfcp devices" on page 693) to display information about the associated SCSI device.

Alternatively, you can use sysfs to read the information. To read attributes of the associated SCSI device, use a command of this form:

```
# cat /sys/class/scsi_device/<device_name>/device/<attribute>
```

where:

<device_name>
is the name of the associated SCSI device.

<attribute>
is one of the attributes in Table 30 on page 209.

Tip: For SCSI-attached tape devices, you can display a summary of this information by using the **lstape** command (see "lstape - List tape devices" on page 676).

Examples

- In this example, information is displayed for a manually configured FCP LUN with LUN 0x4010403200000000 that is accessed through a target port with WWPN 0x500507630300c562 and is attached through an FCP device 0.0.3d0c. For the device, access is permitted.

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000/access_denied
0
```

  For the device to be accessible, the access_denied attribute of the target port, 0x500507630300c562, must also be 0 (see "Displaying port information" on page 198).

- You can use **lszfcp** to display attributes of a SCSI device. The example shows the attributes listed in Table 30 on page 209 as well as other relevant attributes:

```
# lszfcp -l 0x4010403200000000 -a
0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:1077035024
Class = "scsi_device"
    ...
    device_blocked      = "0"
    ...
    iocounterbits       = "32"
    iodone_cnt          = "0xbe"
    ioerr_cnt           = "0x2"
    iorequest_cnt       = "0xbe"
    ...
    model               = "2107900"
    queue_depth         = "32"
    queue_ramp_up_period = "120000"
    ...
    rev                 = ".166"
    scsi_level          = "6"
    state               = "running"
    timeout             = "30"
    type                = "0"
    uevent              = "DEVTYPE=scsi_device"
    vendor              = "IBM"
    ...
    zfcp_access_denied  = "0"
    zfcp_failed         = "0"
    zfcp_in_recovery    = "0"
    zfcp_status         = "0x54000000"
```

Setting the queue depth

The Linux SCSI code automatically adjusts the queue depth as necessary. Changing the queue depth is usually a storage server requirement.

Before you begin

Check the documentation of the storage server used or contact your storage server support group to establish if there is a need to change this setting.

The information in this topic only applies to the SCSI layer. For block devices, such as SCSI disks, there is also a limit in the Linux block layer. To display the block device limit, issue:

```
# cat /sys/bus/scsi/devices/<scsi_device_name>/block/*/queue/nr_requests
```

Alternatively, issue:

```
# cat /sys/block/sd<X>/queue/nr_requests
```

The smaller of SCSI device `queue_depth` and block device `nr_requests` is the effective setting. For more details about block device requests, see www.kernel.org/doc/html/latest/block/queue-sysfs.html#nr-requests-rw and www.kernel.org/doc/html/latest/block/stat.html#in-flight.

About this task

The value of the queue_depth kernel parameter (see “Setting up the zfcp device driver” on page 182) is used as the default queue depth of new SCSI devices. You can query the queue depth by issuing a command of this form:

```
# cat /sys/bus/scsi/devices/<SCSI device>/queue_depth
```

Example:

```
# cat /sys/bus/scsi/devices/0:0:19:1086537744/queue_depth
16
```

You can change the queue depth of each SCSI device by writing to the queue_depth attribute, for example:

```
# echo 8 > /sys/bus/scsi/devices/0:0:19:1086537744/queue_depth
# cat /sys/bus/scsi/devices/0:0:19:1086537744/queue_depth
8
```

This is useful on a running system where you want to make dynamic changes. If you want to make the changes persistent across IPLs, you can:

- Use the module parameter `queue_depth` described in “Setting up the zfcp device driver” on page 182.
- Write a udev rule to change the setting for each new SCSI device.

Linux forwards SCSI commands to the storage server until the number of pending commands exceeds the queue depth. If the server lacks the resources to process a SCSI command, Linux queues the command for a later retry and decreases the queue depth counter. Linux then waits for a defined ramp-up period. If no indications of resource problems occur within this period, Linux increases the queue depth counter until reaching the previously set maximum value. To query the current value for the queue ramp-up period in milliseconds:

```
# cat /sys/bus/scsi/devices/0:0:13:1086537744/queue_ramp_up_period
120000
```

To set a new value for the queue ramp-up period in milliseconds:

```
# echo 1000 > /sys/bus/scsi/devices/0:0:13:1086537744/queue_ramp_up_period
```

Recovering failed SCSI devices

Failed SCSI devices are automatically recovered by the zfcp device driver. You can read the `zfcp_in_recovery` attribute to check whether recovery is under way.

Before you begin

The FCP device must be online.

Procedure

Perform the following steps to check the recovery status of a failed SCSI device:

1. Check the value of the `zfcp_in_recovery` attribute. Issue the **`lszfcp`** command:

   ```
   # lszfcp -l <LUN> -a
   ```

   where *<LUN>* is the LUN of the associated SCSI device.

   Alternatively, you can issue a command of this form:

   ```
   # cat /sys/class/scsi_device/<device_name>/device/zfcp_in_recovery
   ```

   The value is `1` if recovery is under way and `0` otherwise. If the value is `0` for a non-operational SCSI device, recovery might have failed. Alternatively, the device driver might have failed to detect that the SCSI device is malfunctioning.
2. To find out whether recovery failed, read the `zfcp_failed` attribute. Either use the **`lszfcp`** command again, or issue a command of this form:

   ```
   # cat /sys/class/scsi_device/<device_name>/device/zfcp_failed
   ```

   The value is `1` if recovery has failed, and `0` otherwise.
3. You can start or restart the recovery process for the SCSI device by writing `0` to the `zfcp_failed` attribute. Issue a command of this form:

   ```
   # echo 0 > /sys/class/scsi_device/<device_name>/device/zfcp_failed
   ```

Example

In the following example, SCSI device 0:0:0:0 is malfunctioning. The first command reveals that recovery is not already under way. The second command manually starts recovery for the SCSI device:

```
# cat /sys/class/scsi_device/0:0:0:0/device/zfcp_in_recovery
0
# echo 0 > /sys/class/scsi_device/0:0:0:0/device/zfcp_failed
```

What to do next

If you manually configured an FCP LUN (see “Manually configured FCP LUNs and their SCSI devices” on page 202), but did not get a corresponding SCSI device, you can also use the corresponding FCP LUN sysfs attributes, `in_recovery` and `failed`, to check on recovery. See Table 29 on page 209.

Updating the information about SCSI devices

Use the `rescan` attribute of the SCSI device to detect changes to a storage device on the storage server that are made after the device was discovered.

Before you begin

The FCP device must be online.

About this task

The initial information about the available SCSI devices is discovered automatically when LUNs first become available.

Procedure

To update the information about a SCSI device issue a command of this form:

```
# echo <string> > /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/rescan
```

where *<string>* is any alphanumeric string and the other variables have the same meaning as in "Mapping the representations of a SCSI device in sysfs" on page 204.

Example

In the following example, the information about a SCSI device 1:0:18:1086537744 is updated:

```
# echo 1 > /sys/bus/scsi/devices/1:0:18:1086537744/rescan
```

Setting the SCSI command timeout

You can change the timeout if the default is not suitable for your storage system.

Before you begin

The FCP device must be online.

About this task

There is a timeout for SCSI commands. If the timeout expires before a SCSI command completes, error recovery starts. The default timeout is 30 seconds.

To find out the current timeout, read the `timeout` attribute of the SCSI device:

```
# cat /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/timeout
```

where the variables have the same meaning as in "Mapping the representations of a SCSI device in sysfs" on page 204.

The attribute value specifies the timeout in seconds.

Procedure

To set a different timeout, enter a command of this form:

```
# echo <timeout> > /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/timeout
```

where *<timeout>* is the new timeout in seconds.

Example

In the following example, the timeout of a SCSI device 1:0:18:1086537744 is first read and then set to 45 seconds:

```
# cat /sys/bus/scsi/devices/1:0:18:1086537744/timeout
30
# echo 45 > /sys/bus/scsi/devices/1:0:18:1086537744/timeout
```

Controlling the SCSI device state

You can use the `state` attribute of the SCSI device to set a SCSI device back online if it was set offline by error recovery.

Before you begin

The FCP device must be online.

About this task

If the connection to a storage system is working but the storage system has a problem, the error recovery might set the SCSI device offline. This condition is indicated by a message like "Device offlined - not ready after error recovery".

To find out the current state of the device, read the `state` attribute:

```
# cat /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/state
```

where the variables have the same meaning as in “Mapping the representations of a SCSI device in sysfs” on page 204. The state can be:

running
The SCSI device can be used for running regular I/O requests.

cancel
The data structure for the device is being removed.

deleted
Follows the `cancel` state when the data structure for the device is being removed.

quiesce
No I/O requests are sent to the device, only special requests for managing the device. This state is used when the system is suspended.

offline
Error recovery for the SCSI device has failed.

blocked
Error recovery is in progress and the device cannot be used until the recovery process is completed.

Procedure

Issue a command of this form:

```
# echo running > /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/state
```

Example

In the following example, SCSI device 1:0:18:1086537744 is offline and is then set online again:

```
# cat /sys/bus/scsi/devices/1:0:18:1086537744/state
offline
# echo running > /sys/bus/scsi/devices/1:0:18:1086537744/state
```

Removing SCSI devices

How to remove a SCSI device depends on whether your environment is set up to use NPIV.

Important: Configuration changes can directly or indirectly affect information that is required to mount the root file system. Such changes require an update of the initial RAM disk, followed by a re-write of the boot record (see “Rebuilding the initial RAM disk image” on page 96).

Removing automatically attached SCSI devices

When running with NPIV and automatic LUN scan, you can temporarily delete a SCSI device by writing 1 to the `delete` attribute of the directory that represents the device in the sysfs SCSI branch.

About this task

See "Mapping the representations of a SCSI device in sysfs" on page 204 about how to find this directory.

Note: These steps delete the SCSI device only temporarily, until the next automatic or user triggered Linux SCSI target scan. The scan automatically adds the SCSI devices again, unless the LUNs were deconfigured on the storage target. To permanently delete SCSI devices, you must disable automatic LUN scannning and configure all LUNs manually, see "Manually configured FCP LUNs and their SCSI devices" on page 202.

Procedure

Issue a command of this form:

```
# echo 1 > /sys/bus/scsi/devices/<device>/delete
```

Example

In this example, an SCSI device with LUN `0x4010403700000000` is to be removed. Before the device is deleted, the corresponding device in the sysfs SCSI branch is found with an **lszfcp** command.

```
# lszfcp -l 0x4010403700000000
0.0.3d0f/0x500507630300c567/0x4010403700000000 0:0:3:1077362704
# echo 1 > /sys/bus/scsi/devices/0:0:3:1077362704/delete
```

Removing manually configured FCP LUNs and their SCSI device

Manually remove a SCSI device if your environment is not set up to use NPIV or if you disabled automatic LUN scan. For details about disabling automatic LUN scan, see "Setting up the zfcp device driver" on page 182.

Before you begin

Attention: Use this procedure only to dynamically test configuration settings.

To configure a persistent setting in a production system, remove the dracut boot parameter `rd.zfcp=` only for SCSI disks that are no longer required for the root file system. Use the configuration file `/etc/zfcp.conf` for all other SCSI devices, such as data volumes or tape libraries.

Procedure

Follow these steps to remove a manually configured FCP LUN and its SCSI device:

- Use the **chzdev** command. Issue a command of this form:

  ```
  chzdev -d -a zfcp-lun <device_bus_id>:<wwpn>:<fcp_lun>
  ```

- Alternatively, remove the SCSI device from the target port by writing the LUN of the device to the `unit_remove` attribute of the port. Issue a command of this form:

  ```
  # echo <fcp_lun> > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/unit_remove
  ```

  where the variables have the same meaning as in "Configuring SCSI devices" on page 202. Removing a LUN with `unit_remove` automatically unregisters the SCSI device first.

Example

The following example removes a SCSI device with LUN 0x4010403200000000, accessed through a target port with WWPN 0x500507630300c562 and is attached through an FCP device with bus ID 0.0.3d0c.

- Using **chzdev**, issue:

```
# chzdev -d zfcp-lun 0.0.3d0c:0x500507630300c562:0x4010403200000000
```

- Using sysfs, remove the device and the LUN:

```
# echo 0x4010403200000000 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/unit_remove
```

Confirming end-to-end configurations

You can confirm that specific integrity and security configurations are in place for your connections.

- "Confirming end-to-end data consistency checking" on page 216
- "Investigating IBM Fibre Channel Endpoint Security" on page 217

Confirming end-to-end data consistency checking

There are different types of end-to-end data consistency checking, with dependencies on hardware and software.

About this task

End-to-end data consistency checking is based on a data integrity field (DIF) that is added to transferred data blocks. DIF data is used to confirm that a data block originates from the expected source and was not modified during the transfer between the storage system and the FCP device. The SCSI Block Commands (T10 SBC) standard defines several types of DIF. Linux data integrity extension (DIX) builds on DIF to extend consistency checking, for example, to the operating system, middleware, or an application.

The zfcp device driver supports the following modes of end-to-end data consistency checking:

- The FCP device calculates and checks a DIF checksum (DIF type 1)

  Enable this mode with the `dif=` module parameter.

  Interface change: As of Red Hat Enterprise Linux 8.2, the dif= module parameter enables DIF only and no longer includes DIX. Use the dix= module parameter to enable end-to-end data consistency checking in extended mode, with both DIF and DIX.
- The Linux block integrity layer calculates and checks a TCP/IP checksum, which the FCP device then translates to a DIF checksum (DIX type 1 with DIF type 1)

  Enable this mode with the `dix=` module parameter.

  Note: End-to-end data consistency checking in extended mode is experimental. SCSI disks for which this mode is enabled must be accessed with direct I/O. Direct I/O requires direct access through the block device or through a file system that fully supports end-to-end data consistency checking in extended mode. For example, XFS provides this support. Expect error messages about invalid checksums when you use other access methods.

For more information about the module parameters that control the end-to-end data consistency checking, see "Setting up the zfcp device driver" on page 182.

For SCSI devices for which end-to-end data consistency checking is used, there is a sysfs directory

```
/sys/block/sd<x>/integrity
```

In the path, sd<*x*> is the standard name of the block device of the SCSI disk.

End-to-end data consistency checking is used only if all of the following components support it:

SCSI disk
Check your storage server documentation about T10 DIF support and any restrictions.

IBM Z hardware
IBM Z FCP adapter hardware supports end-to-end data consistency checking as of FICON Express8.

Hypervisor
For Linux on z/VM, you require a z/VM version with guest support for end-to-end data consistency checking.

FCP device
Check your FCP adapter hardware documentation about the support and any restrictions. For example, end-to-end data consistency checking might be supported only for disks with 512-byte block size.

Read the `prot_capabilities` sysfs attribute of the SCSI host that is associated with an FCP device to find out about its end-to-end data consistency checking support. The following values are possible:

0
The FCP device does not support end-to-end data consistency checking.

1
The FCP device supports DIF type 1.

17
The FCP device supports DIX type 1 with DIF type 1.

Procedure

Issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/host<n>/scsi_host/host<n>/prot_capabilities
```

where *<device_bus_id>* identifies the FCP device and *<n>* is an integer that identifies the corresponding SCSI host.

Example

```
# cat /sys/bus/ccw/devices/0.0.1940/host0/scsi_host/host0/prot_capabilities
1
```

Investigating IBM Fibre Channel Endpoint Security

You can check whether the connections between your FCP devices and remote ports use authentication and encryption.

About this task

You can investigate two aspects of IBM Fibre Channel Endpoint Security for your connections:

- The capabilities of your FCP device, which depend on your adapter hardware with its FCP channels.
- The status of your connections between your FCP devices and remote ports.

For information about configuring IBM Fibre Channel Endpoint Security, see the Redbooks® publication *IBM Fibre Channel Endpoint Security for IBM DS8900F and IBM Z*, SG24-8455.

Procedure

Display the IBM Fibre Channel Endpoint Security information for your environment by issuing an **lszdev** command. Use command options to read the `fc_security` attributes for your Fibre Channel hosts and LUNs as shown in the following example:

```
# lszdev zfcp -a -c TYPE,ID,ATTR:fc_security
TYPE      ID                                                 ATTR:fc_security
zfcp-host 0.0.5150                                           Authentication, Encryption
zfcp-lun  0.0.5150:0x500507630400120c:0x4081402000000000 Authentication
zfcp-lun  0.0.5150:0x500507630401120c:0x4081402000000000 Encryption
```

In the output, `zfcp-host` lines show information for your FCP devices:

Authentication
The FCP device supports authentication.

Encryption
The FCP device supports encryption.

unsupported
The FCP device does not support IBM Fibre Channel Endpoint Security.

none
The FCP device does not report any IBM Fibre Channel Endpoint Security capabilities.

unknown
The IBM Fibre Channel Endpoint Security capabilities of the FCP device are not known.

In the output, `zfcp-lun` lines show the current state of IBM Fibre Channel Endpoint Security of the connection between the FCP device and the FC remote port used to access the LUN:

Authentication
The connection was authenticated.

Encryption
The connection uses encryption.

unsupported
The connection does not support IBM Fibre Channel Endpoint Security because the FCP device does not support it.

none
The connection has no IBM Fibre Channel Endpoint Security.

unknown
The IBM Fibre Channel Endpoint Security state of the connection is not known.

Tip: If the output is lengthy, use the **lszdev** device selection filter to narrow the scope to the devices of interest (see “lszdev - Display IBM Z device configurations” on page 688).

Alternatively, you can use the **lszfcp** command with the `-a` option to display the IBM Fibre Channel Endpoint Security information for FCP devices. See “lszfcp - List zfcp devices” on page 693. For example, issue the following command:

```
# lszfcp -Ha
```

Instead of using commands, you can read the information directly from sysfs. For example, for an FCP channel that provides an FCP device with device-bus ID `0.0.5150`:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.5150/fc_security
Authentication, Encryption
```

For a remote port `0x500507630401120c` that is connected through this FCP device:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.5150/0x500507630401120c/fc_security
Encryption
```

Both sysfs attributes are read-only.

Scenario for finding available LUNs

There are several steps from setting an FCP device online to listing the available LUNs.

Procedure

1. Check for available FCP devices of type 1732/03:

```
# lscss -t 1732/03
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.3c02 0.0.0015  1732/03 1731/03      80  80  ff   36000000 00000000
```

   Another possible type would be, for example, 1732/04.
2. Set the FCP device online:

```
# chccwdev -e 0.0.3c02
```

   A port scan is performed automatically when the FCP device is set online.
3. Optional: Confirm that the FCP device is available and online:

```
# lszfcp -b 0.0.3c02 -a
0.0.3c02 host0
Bus = "ccw"
    availability        = "good"
...
    failed              = "0"
...
    in_recovery         = "0"
...
    online              = "1"
...
```

4. Optional: List the available ports:

```
# lszfcp -P
0.0.3c02/0x50050763030bc562 rport-0:0-0
0.0.3c02/0x500507630310c562 rport-0:0-1
0.0.3c02/0x500507630040727b rport-0:0-10
0.0.3c02/0x500507630e060521 rport-0:0-11
...
```

5. Scan for available LUNs on FCP device 0.0.3c02, port 0x50050763030bc562:

```
# lsluns -c 0.0.3c02 -p 0x50050763030bc562
Scanning for LUNs on adapter 0.0.3c02
        at port 0x50050763030bc562:
                0x4010400000000000
                0x4010400100000000
                0x4010400200000000
                0x4010400300000000
                0x4010400400000000
                0x4010400500000000
                0x4010400600000000
                ...
```

zfcp HBA API support

You require different libraries for developing and running HBA management client applications. To develop applications, you need the development version of the zFCP HBA API library. To run applications, you need the zFCP HBA API library.

Developing applications

To develop applications, you must install the development version of the zFCP HBA API provided by the `libzfcphbaapi-devel` RPM and link your application against the library.

Procedure

1. Install the development RPM for the zFCP HBA API.

Use, for example, yum:

```
# yum install libzfcphbaapi-devel
```

The development RPM `libzfcphbaapi-devel` provides the necessary header files and .so symbolic links needed to program against the zFCP HBA API library.

2. Add the command line option `-lzfcphbaapi` during the linker step of the build process to link your application against the zFCP HBA API library.
3. In the application, issue the **`HBA_LoadLibrary()`** call as the first call to load the library.

Functions provided

The zfcp HBA API implements Fibre Channel - HBA API (FC-HBA) functions as defined in the FC-HBA specification.

You can find the FC-HBA specification at www.t11.org. The following functions are available:

- HBA_GetVersion()
- HBA_LoadLibrary()
- HBA_FreeLibrary()
- HBA_RegisterLibrary()
- HBA_RegisterLibraryV2()
- HBA_GetNumberOfAdapters()
- HBA_GetAdapterName()
- HBA_OpenAdapter()
- HBA_CloseAdapter()
- HBA_RefreshInformation()
- HBA_RefreshAdapterConfiguration()
- HBA_GetAdapterAttributes()
- HBA_GetAdapterPortAttributes()
- HBA_GetDiscoveredPortAttributes()
- HBA_GetFcpTargetMapping()
- HBA_GetFcpTargetMappingV2()
- HBA_SendScsiInquiry()
- HBA_SendReadCapacity()
- HBA_SendReportLUNs()
- HBA_SendReportLUNsV2()
- HBA_SendCTPassThru()
- HBA_SendCTPassThruV2()
- HBA_SetRNIDMgmtInfo()
- HBA_GetRNIDMgmtInfo()
- HBA_SendRNID()
- HBA_SendRNIDV()
- HBA_SendRPL()
- HBA_SendRPS()
- HBA_SendSRL()
- HBA_SendLIRR()
- HBA_GetEventBuffer()

- HBA_RegisterForAdapterAddEvents()
- HBA_RegisterForAdapterEvents()
- HBA_RegisterForAdapterPortEvents()
- HBA_RegisterForAdapterPortStatEvents()
- HBA_RegisterForTargetEvents()
- HBA_RegisterForLinkEvents()
- HBA_RemoveCallback()

All other FC-HBA functions return status code HBA_STATUS_ERROR_NOT_SUPPORTED where possible.

Note: ZFCP HBA API can access only FCP devices, ports, and units that are configured in the operating system.

Getting ready to run applications

To run an application, you must install the zFCP HBA API library that is provided by the `libzfcphbaapi` RPM. You can set environment variables to log any errors in the library, and use tools to investigate the SAN configuration.

Before you begin

To use the HBA API support, you need the zFCP HBA API library, `libzfcphbaapi`. Installing `libzfcphbaapi` automatically installs all dependent packages.

The application must be developed to use the zFCP HBA API library, see "Developing applications" on page 219.

Procedure

Follow these steps to access the library from a client application:

1. Install the `libzfcphbaapi` RPM with **yum**.
   For example:

```
# yum install libzfcphbaapi
```

2. Optional: Set the environment variables for logging errors.

   The zfcp HBA API support uses the following environment variables to log errors in the zfcp HBA API library:

   LIB_ZFCP_HBAAPI_LOG_LEVEL
   specifies the log level. If not set or set to zero, there is no logging (default). If set to an integer value greater than 1, logging is enabled.

   LIB_ZFCP_HBAAPI_LOG_FILE
   specifies a file for the logging output. If not specified, stderr is used.

What to do next

You can use the **zfcp_ping** and **zfcp_show** commands to investigate your SAN configuration.

Tools for investigating your SAN configuration

As of version 2.1, the HBA API package includes tools that can help you to investigate your SAN configuration and to solve configuration problems.

zfcp_ping
to probe a port in the SAN.

zfcp_show
to retrieve information about the SAN topology and details about the SAN components.

See *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413 for details.

Chapter 13. Storage-class memory device driver

LPAR only: The Storage-class memory device driver applies to Linux in LPAR mode only.

Storage-class memory (SCM) is a class of data storage devices that combines properties of both storage and memory..

SCM can be implemented as Flash Express or as Virtual Flash Memory.

What you should know about storage-class memory

Storage-class memory (SCM) is accessed, in chunks called *increments*, through extended asynchronous data mover (EADM) subchannels.

The LPAR on which your Linux instance runs must be configured to provide SCM.

- At least one EADM subchannel must be available to the LPAR. Because SCM supports multiple concurrent I/O requests, it is advantageous to configure multiple EADM subchannels. A typical number of EADM subchannels is 64.
- One or more SCM increments must be added to the I/O configuration of the LPAR.

In Linux, each increment is represented as a block device. You can use the block device with standard Linux tools as you would use any other block device. Commonly used tools that work with block devices include: **`fdisk`**, **`mkfs`**, and **`mount`**.

Storage-class memory is useful for workloads with large write operations, that is, with a block size of 256 KB or more of data. Write operations with a block size of less than 256 KB of data might not perform optimally. Read operations can be of any size.

Storage-class memory device nodes

Applications access storage-class memory devices by device nodes. Red Hat Enterprise Linux creates a device node for each storage increment.

The device driver uses a device name of the form `/dev/scm<x>` for an entire block device. In the name, *<x>* is one or two lowercase letters.

You can partition a block device into up to seven partitions. If you use partitions, the device driver numbers them from 1 - 7. The partitions then have device nodes of the form `/dev/scm<x><n>`, where *<n>* is a number in the range 1 - 7, for example, `/dev/scma1`.

The following example shows two block devices, `scma` and `scmb`, where `scma` has one partition, `scma1`.

```
# lsblk
NAME     MAJ:MIN RM   SIZE RO MOUNTPOINT
scma     252:0    0    16G  0
`-scma1  252:1    0    16G  0
scmb     252:8    0    16G  0
```

Setting up the storage-class memory device driver

Configure the storage-class memory device driver by using the module parameters.

Storage-class memory module parameter syntax



where

nr_requests
specifies the number of parallel I/O requests. Set this number to the number of EADM subchannels. The default is 64.

write_cluster_size
specifies the number of pages that are used by the read-modify-write algorithm (available if CONFIG_SCM_BLOCK_CLUSTER_WRITE=y). The default is 64, resulting in all write requests smaller than 256 KiB being translated to 256 KiB writes. 1 KiB is 1024 bytes. Valid values are 2, 4, 8, 16, 32, 64, and 128. Do not change this value unless asked to do so by your service organization.

nr_request_per_io
submits more concurrent I/O requests than the current limit, which is based on the number of available EADM subchannels (64). Valid values are 1 - 64. Increasing the requests increases the number of I/O requests per second, especially for requests with a small block size. The default number of requests is 8. Depending on the workload, this setting might improve the throughput of the scm_block driver.

Working with storage-class memory increments

You can list storage-class memory increments and EADM devices.

- "Displaying EADM subchannels" on page 224
- "Listing storage-class memory increments" on page 225
- "Combining SCM devices with LVM" on page 225

Displaying EADM subchannels

Use the **lscss** command to list EADM subchannels.

About this task

The extended asynchronous data mover (EADM) subchannels are used to transfer data to and from the storage-class memory. At least one EADM subchannel must be available to the LPAR.

Procedure

To list EADM subchannels, issue:

```
# lscss --eadm
Device   Subchan.
----------------
n/a      0.0.ff00
n/a      0.0.ff01
n/a      0.0.ff02
n/a      0.0.ff03
n/a      0.0.ff04
n/a      0.0.ff05
n/a      0.0.ff06
n/a      0.0.ff07
```

For more information about the **lscss** command, see “lscss - List subchannels” on page 658.

Listing storage-class memory increments

Use the **lsscm** command to see the status and attributes of storage-class memory increments.

About this task

Each storage-class memory increment can be accessed as a block device through a device node `/dev/scm<x>`. Optionally, you can partition a storage-class memory increment in up to seven partitions.

You can also use the **lsblk** command to list all block devices.

Procedure

To list all storage-class memory increments, their status, and attributes, issue:

```
# lsscm
SCM Increment    Size    Name  Rank D_state O_state Pers ResID
------------------------------------------------------------
0000000000000000 16384MB scma     1       2       1    2     1
0000000400000000 16384MB scmb     1       2       1    2     1
```

See “lsscm - List storage-class memory increments” on page 673 for details about the **lsscm** command.

Combining SCM devices with LVM

You can use LVM to combine multiple SCM block devices into an arbitrary sized LVM device.

Example

Configure SCM as any other block devices in LVM. If your version of LVM does not accept SCM devices as valid LVM device types and issues an error message, add the SCM devices to the LVM configuration file `/etc/lvm/lvm.conf`. Add the following line to the section labeled "devices":

```
types = [ "scm", 8 ]
```

Chapter 14. Managing NVMe devices

LPAR and z/VM: The NVMe information applies to Linux in LPAR mode and to Linux on z/VM.

As of LinuxONE II, PCIe-attached NVMe devices are supported on IBM LinuxONE.

As of LinuxONE III, you can use NVMe devices as stand-alone dump devices for Linux in LPAR mode or in a DPM partition, see *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751.

The general information about PCIe and PCIe-attached devices applies to NVMe, see Chapter 33, "PCI Express support," on page 401.

To Linux, NVMe devices are block devices that can be partitioned and hold file systems. NVMe disk controllers use *name spaces* to divide a physical NVMe device into multiple logical devices. Booting from an HMC might require name space ID 1 for NVMe boot devices.

Device names and nodes

NVMe device names follow this pattern: `nvme<number>n<namespace>`, where:

<number>
: is an integer that is assigned by Linux during the boot process. The first NVMe device that is detected is assigned `0`. Devices that follow are assigned consecutive numbers.

<namespace>
: is an NVMe name space ID that is assigned by the NVMe disk controller.

For partitions, p*<n>* is appended to the device name of the whole device, where *<n>* denotes the *<n>*-th partition.

Table 31. Standard device names and nodes

Standard device names	Standard device nodes	Comment

Whole device:
`nvme0n1`
Partitions:
`nvme0n1p1`
`nvme0n1p2`
`...`
	Whole device:
	`/dev/nvme0n1`
	Partitions:
	`/dev/nvme0n1p1`
	`/dev/nvme0n1p2`
	`...`
		First device with name space 1

Whole device:
`nvme0n2`
Partitions:
`nvme0n2p1`
`nvme0n2p2`
`...`
	Whole device:
	`/dev/nvme0n2`
	Partitions:
	`/dev/nvme0n2p1`
	`/dev/nvme0n2p2`
	`...`
		First device with name space 2

Whole device:
`nvme1n1`
Partitions:
`nvme1n1p1`
`nvme1n1p2`
`...`
	Whole device:
	`/dev/nvme1n1`
	Partitions:
	`/dev/nvme1n1p1`
	`/dev/nvme1n1p2`
	`...`
		Second device with name space 1

The mapping between physical storage space and standard device names does not persist across reboots. Depending on the udev rules of your distribution, udev creates other device nodes for you.

Example: node based on a WWN

```
/dev/disk/by-id/nvme-eui.01000000010000005cd2e4f0bc174f51
```

The WWN is a unique, fixed hardware property. This type of device node maps to the same NVMe device, across reboots.

Example: node based on manufacturer specifications
`/dev/disk/by-id/nvme-INTEL_SSDPE2KX040T7_PHLF806200284P0IGN`

The manufacturer specification is a unique, fixed hardware property that includes the hardware model and serial number. This type of device node maps to the same NVMe device, across reboots.

Example: node based on the PCI function address
`/dev/disk/by-path/pci-0850:00:00.0-nvme-1`

In this example, the function address is `0850:00:00.0`. Whether the mapping of function addresses and NVMe device persists across reboots depends on your hardware setup.

The device nodes that udev creates for partitions depend on the udev rules. Commonly, the nodes names match the names of the whole device, with `-part1` appended for the first partition, `-part2` for the second partition, and `-part<x>` for the *<x>*th partition.

Function addresses

If your LinuxONE hardware is configured to support UIDs, NVMe function addresses follow the pattern *<UID>*`:00:00.0` and map to the same physical PCI slot of the NVMe device across reboots.

Without UID support, the pattern is *<hhhh>*`:00:00.0`, where the variable part, *<hhhh>*, is a 4-digit hexadecimal number. Linux sets this number to `0000` for the first PCIe device that it discovers and increments it by 1 for subsequent devices. So, according to this pattern, the function addresses for the first 3 PCIe devices are: `0000:00:00.0`, `0001:00:00.0`, and `0002:00:00.0`. This naming scheme does not persist across reboots. Because function addresses include all PCIe devices, this means that addresses that mapped to a specific NVMe device might not only map to a different NVMe device, but to a different type of PCIe device altogether.

To find the function address for a standard device node use the **`ls`** command to display details for the device's representation as a block device in sysfs.

```
# ls -l /sys/block/nvme0n1
lrwxrwxrwx. 1 root root 0 Oct 23 16:46 /sys/block/nvme0n1 -> ../devices/pci0850:00/0850:00:00.0/
nvme/nvme0/nvme0n1
```

In the example, `nvme0n1` maps to an NVMe device with a PCIe function address `0850:00:00.0`.

Tip: Issue `ls -l /sys/block/nvme*` for a complete mapping of function addresses and standard device nodes.

NVMe devices in sysfs

PCIe-attached NVMe devices have all generic PCIe sysfs attributes at `/sys/bus/pci/devices/<function_address>`, see Chapter 33, “PCI Express support,” on page 401.

You can find NVMe-specific attributes in the device representations at `/sys/bus/pci/drivers/nvme/<function_address>`.

NVMe devices with name space ID 1 can be prepared as boot devices. Consequently, they might be represented in `/sys/firmware`:

- As the IPL device of the current Linux instance at `/sys/firmware/ipl`, see “Further attributes for IPL type fcp” on page 116.
- As the currently configured re-IPL device at `/sys/firmware/reipl/nvme`, see “Attributes for nvme” on page 119.

Chapter 15. Channel-attached tape device driver

LPAR and z/VM: The channel-attached tape device driver applies to Linux in LPAR mode and to Linux on z/VM.

The tape device driver supports channel-attached tape devices on Red Hat Enterprise Linux 9.1 for IBM Z.

SCSI tape devices that are attached through an FCP channel are handled by the zfcp device driver (see Chapter 12, “SCSI-over-Fibre Channel device driver,” on page 177).

Features

The tape device driver supports a range of channel-attached tape devices and functions of these devices.

- The tape device driver supports channel-attached tape drives that are compatible with IBM 3480, 3490, 3590, and 3592 magnetic tape subsystems. Various models of these device types are handled (for example, the 3490/10).

  3592 devices that emulate 3590 devices are recognized and treated as 3590 devices.
- Logical character devices for non-rewinding and rewinding modes of operation (see “Tape device modes and logical devices” on page 229).
- Control operations through **mt** (see “Using the mt command” on page 231).
- Message display support (see “tape390_display - Display messages on tape devices and load tapes” on page 739).
- Encryption support (see “tape390_crypt - Manage tape encryption” on page 735).
- Up to 128 physical tape devices.

What you should know about channel-attached tape devices

A naming scheme helps you to keep track of your tape devices, their modes of operation, and the corresponding device nodes.

Tape device modes and logical devices

The tape device driver supports up to 128 physical tape devices. Each physical tape device can be used as a character device in non-rewinding or in rewinding mode.

In non-rewinding mode, the tape remains at the current position when the device is closed. In rewinding mode, the tape is rewound when the device is closed. The tape device driver treats each mode as a separate logical device.

Both modes provide sequential (traditional) tape access without any caching done in the kernel.

You can use a channel-attached tape device in the same way as any other Linux tape device. You can write to it and read from it using standard Linux facilities such as GNU **tar**. You can perform control operations (such as rewinding the tape or skipping a file) with the standard tool **mt**.

Tape naming scheme

The tape device driver assigns minor numbers along with an index number when a physical tape device comes online.

The naming scheme for tape devices is summarized in Table 32 on page 230:

Table 32. Tape device names and minor numbers

Device	Names	Minor numbers
Non-rewinding character devices	ntibm*<n>*	2×*<n>*
Rewinding character devices	rtibm*<n>*	2×*<n>*+1

where *<n>* is the index number that is assigned by the device driver. The index starts from 0 for the first physical tape device, 1 for the second, and so on. The name space is restricted to 128 physical tape devices, so the maximum index number is 127 for the 128th physical tape device.

The index number and corresponding minor numbers and device names are not permanently associated with a specific physical tape device. When a tape device goes offline, it surrenders its index number. The device driver assigns the lowest free index number when a physical tape device comes online. An index number with its corresponding device names and minor numbers can be reassigned to different physical tape devices as devices go offline and come online.

Tip: Use the **lstape** command (see "lstape - List tape devices" on page 676) to determine the current mapping of index numbers to physical tape devices.

When the tape device driver is loaded, it dynamically allocates a major number to channel-attached character tape devices. A different major number might be used when the device driver is reloaded, for example when Linux is rebooted.

For online tape devices directories provide information about the major/minor assignments. The directories have the form:

- `/sys/class/tape390/ntibm<n>`
- `/sys/class/tape390/rtibm<n>`

Each of these directories has a `dev` attribute. The value of the `dev` attribute has the form *<major>*:*<minor>*, where *<major>* is the major number of the device and *<minor>* is the minor number specific to the logical device.

Example

In this example, four physical tape devices are present, with three of them online. The TapeNo column shows the index number and the BusID indicates the associated physical tape device. In the example, no index number is allocated to the tape device in the first row. The device is offline and, currently, no names and minor numbers are assigned to it.

```
# lstape --ccw-only
TapeNo  BusID      CuType/Model DevType/DevMod  BlkSize State   Op      MedState
0       0.0.01a1   3490/10      3490/40         auto    UNUSED  ---     UNLOADED
1       0.0.01a0   3480/01      3480/04         auto    UNUSED  ---     UNLOADED
2       0.0.0172   3590/50      3590/11         auto    IN_USE  ---     LOADED
N/A     0.0.01ac   3490/10      3490/40         N/A     OFFLINE ---     N/A
```

Table 33 on page 230 summarizes the resulting names and minor numbers.

Table 33. Example names and minor numbers

Bus ID	Index (TapeNo)	Device	Device name	Minor number
0.0.01a1	0	non-rewind	ntibm0	0
		rewind	rtibm0	1
0.0.01a0	1	non-rewind	ntibm1	2
		rewind	rtibm1	3

Table 33. Example names and minor numbers (continued)

Bus ID	Index (TapeNo)	Device	Device name	Minor number
0.0.0172	2	non-rewind	ntibm2	4
		rewind	rtibm2	5
0.0.01ac	not assigned	n/a	n/a	not assigned

For the online character devices, the major/minor assignments can be read from their respective representations in `/sys/class`:

```
# cat /sys/class/tape390/ntibm0/dev
254:0
# cat /sys/class/tape390/rtibm0/dev
254:1
# cat /sys/class/tape390/ntibm1/dev
254:2
# cat /sys/class/tape390/rtibm1/dev
254:3
# cat /sys/class/tape390/ntibm2/dev
254:4
# cat /sys/class/tape390/rtibm2/dev
254:5
```

In the example, the major number that is used for character devices is 254. The minor numbers are as expected for the respective device names.

Tape device nodes

Applications access tape devices by device nodes. Red Hat Enterprise Linux 9.1 uses udev to create two device nodes for each tape device.

The device nodes have the form `/dev/<name>`, where *<name>* is the device name according to "Tape naming scheme" on page 229.

For example, if you have two tape devices, udev creates the device nodes that are shown in Table 34 on page 231:

Table 34. Tape device nodes

Node for	non-rewind device	rewind device
First tape device	`/dev/ntibm0`	`/dev/rtibm0`
Second tape device	`/dev/ntibm1`	`/dev/rtibm1`

Using the mt command

There are differences between the MTIO interface for channel-attached tapes and other tape drives. Correspondingly, some operations of the **mt** command are different for channel-attached tapes.

The **mt** command handles basic tape control in Linux. See the man page for general information about **mt**.

Basic Linux tape control is handled by the **mt** utility. See the man page for general information about **mt**.

setdensity
has no effect because the recording density is automatically detected on channel-attached tape hardware.

drvbuffer
has no effect because channel-attached tape hardware automatically switches to unbuffered mode if buffering is unavailable.

lock / unlock
have no effect because channel-attached tape hardware does not support media locking.

setpartition / mkpartition
have no effect because channel-attached tape hardware does not support partitioning.

status
returns a structure that, aside from the block number, contains mostly SCSI-related data that does not apply to the tape device driver.

load
does not automatically load a tape but waits for a tape to be loaded manually.

offline or rewoffl or eject
all include expelling the currently loaded tape. Depending on the stacker mode, it might attempt to load the next tape (see “Loading and unloading tapes” on page 236 for details).

Loading the tape device driver

You must load the appropriate tape device driver module before you can work with tape devices.

Use the **modprobe** command to ensure that any other required modules are loaded in the correct order.

Tape module syntax



See the **modprobe** man page for details about **modprobe**.

To load the tape device driver module automatically at boot time, see the section on persistent module loading in *Configuring basic system settings* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_basic_system_settings/)

Working with tape devices

Typical tasks for working with tape devices include displaying tape information, controlling compression, and loading and unloading tapes.

For information about working with the channel measurement facility, see Chapter 55, “Channel measurement facility,” on page 539.

For information about displaying messages on a tape device's display unit, see “tape390_display - Display messages on tape devices and load tapes” on page 739.

See “Working with newly available devices” on page 10 to avoid errors when working with devices that have become available to a running Linux instance.

- “Setting a tape device online or offline” on page 232
- “Displaying tape information” on page 234
- “Enabling compression” on page 236
- “Loading and unloading tapes” on page 236

Setting a tape device online or offline

Set a tape device online or offline with the **chccwdev** command or through the `online` sysfs attribute of the device.

About this task

Setting a physical tape device online makes both corresponding logical devices accessible:

- The non-rewind character device

- The rewind character device

At any time, the device can be online to a single Linux instance only. You must set the tape device offline to make it accessible to other Linux instances in a shared environment.

Procedure

Use the **chzdev** command (see "chzdev - Configure IBM Z devices" on page 590) to set a tape online or offline.

Alternatively, you can write `1` to the device's online attribute to set it online or `0` to set it offline.

Results

When a physical tape device is set online, the device driver assigns an index number to it. This index number is used in the standard device nodes (see "Tape device nodes" on page 231) to identify the corresponding logical devices. The index number is in the range 0 - 127. A maximum of 128 physical tape devices can be online concurrently.

If you are using the standard device nodes, you must find out which index number the tape device driver has assigned to your tape device. This index number, and consequently the associated standard device node, can change after a tape device was set offline and back online.

If you need to know the index number, issue a command of this form:

```
# lstape --ccw-only <device_bus_id>
```

where *<device_bus_id>* is the device bus-ID that corresponds to the physical tape device. The index number is the value in the TapeNo column of the command output. For more information about the **lstape** command, see "lstape - List tape devices" on page 676.

Examples

- To set a physical tape device with device bus-ID 0.0.015f online, first load the module if you have not already done so:

  ```
  # modprobe tape_3590
  ```

  Then issue:

  ```
  # chzdev -e -a tape 015f
  ```

  Alternatively, issue:

  ```
  # chccwdev -e 0.0.015f
  ```

  or

  ```
  # echo 1 > /sys/bus/ccw/devices/0.0.015f/online
  ```

  To find the index number that the tape device driver assigned to the device, issue:

  ```
  # lstape 0.0.015f --ccw-only
  TapeNo  BusID      CuType/Model DevType/Model   BlkSize State   Op      MedState
  2       0.0.015f   3480/01      3480/04         auto    UNUSED  ---     LOADED
  ```

  In the example, the assigned index number is 2. The standard device nodes for working with the device until it is set offline are then:

  – `/dev/ntibm2` for the non-rewinding device

  - /dev/rtibm2 for the rewinding device
- To set a physical tape device with device bus-ID 0.0.015f offline, issue:

```
# chzdev -d -a tape 015f
```

or

```
# chccwdev -d 0.0.015f
```

or

```
# echo 0 > /sys/bus/ccw/devices/0.0.015f/online
```

Displaying tape information

Use the **lstape** command to display summary information about your tape devices, or read tape information from sysfs.

Alternatively, you can read tape information from sysfs. Each physical tape device is represented in a sysfs directory of the form

```
/sys/bus/ccw/devices/<device_bus_id>
```

where *<device_bus_id>* is the device bus-ID that corresponds to the physical tape device. This directory contains a number of attributes with information about the physical device. The attributes: blocksize, state, operation, and medium_state, might not show the current values if the device is offline.

Table 35. Tape device attributes

Attribute	Explanation
online	1 if the device is online or 0 if it is offline (see “Setting a tape device online or offline” on page 232)
cmb_enable	1 if channel measurement block is enabled for the physical device or 0 if it is not enabled (see Chapter 55, “Channel measurement facility,” on page 539)
cutype	Type and model of the control unit
devtype	Type and model of the physical tape device
blocksize	Currently used block size in bytes or 0 for auto
state	State of the physical tape device, either of: **UNUSED** Device is not in use and is available to any operating system image in a shared environment **IN_USE** Device is being used as a character device by a process on this Linux image **OFFLINE** The device is offline. **NOT_OP** Device is not operational

Table 35. Tape device attributes (continued)

Attribute	Explanation
operation	The current tape operation, for example: **---** No operation **WRI** Write operation **RFO** Read operation **MSN** Medium sense Several other operation codes exist, for example, for rewind and seek.
medium_state	The current state of the tape cartridge: **1** Cartridge is loaded into the tape device **2** No cartridge is loaded **0** The tape device driver does not have information about the current cartridge state

Procedure

Issue a command of this form to read an attribute:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/<attribute>
```

where *<attribute>* is one of the attributes of Table 35 on page 234.

Example

The following **lstape** command displays information about a tape device with bus ID 0.0.015f:

```
# lstape 0.0.015f --ccw-only
TapeNo  BusID      CuType/Model DevType/Model   BlkSize State   Op      MedState
2       0.0.015f   3480/01      3480/04         auto    UNUSED  ---     LOADED
```

This sequence of commands reads the same information from sysfs:

```
# cat /sys/bus/ccw/devices/0.0.015f/online
1
# cat /sys/bus/ccw/devices/0.0.015f/cmb_enable
0
# cat /sys/bus/ccw/devices/0.0.015f/cutype
3480/01
# cat /sys/bus/ccw/devices/0.0.015f/devtype
3480/04
# cat /sys/bus/ccw/devices/0.0.015f/blocksize
0
# cat /sys/bus/ccw/devices/0.0.015f/state
UNUSED
# cat /sys/bus/ccw/devices/0.0.015f/operation
---
# cat /sys/bus/ccw/devices/0.0.015f/medium_state
1
```

Enabling compression

Control Improved Data Recording Capability (IDRC) compression with the **mt** command provided by the RPM mt-st.

About this task

Compression is off after the tape device driver is loaded.

Procedure

To enable compression, issue:

```
# mt -f <node> compression
```

or

```
# mt -f <node> compression 1
```

where *<node>* is the device node for a character device, for example, `/dev/ntibm0`.

To disable compression, issue:

```
# mt -f <tape> compression 0
```

Any other numeric value has no effect, and any other argument disables compression.

Example

To turn on compression for a tape device with a device node `/dev/ntibm0` issue:

```
# mt -f /dev/ntibm0 compression 1
```

Loading and unloading tapes

Unload tapes with the **mt** command. How to load tapes depends on the stacker mode of your tape hardware.

Procedure

Unload tapes by issuing a command of this form:

```
# mt -f <node> unload
```

where *<node>* is one of the character device nodes.

Whether or not you can load tapes from your Linux instance depends on the stacker mode of your tape hardware. There are three possible modes:

manual
: Tapes must always be loaded manually by an operator. You can use the **tape390_display** command (see “tape390_display - Display messages on tape devices and load tapes” on page 739) to display a short message on the tape device's display unit when a new tape is required.

automatic
: If there is another tape present in the stacker, the tape device automatically loads a new tape when the current tape is expelled. You can load a new tape from Linux by expelling the current tape with the **mt** command.

system
The tape device loads a tape when instructed from the operating system. From Linux, you can load a tape with the **tape390_display** command (see “tape390_display - Display messages on tape devices and load tapes” on page 739). You cannot use the **mt** command to load a tape.

Example

To expel a tape from a tape device that can be accessed through a device node `/dev/ntibm0`, issue:

```
# mt -f /dev/ntibm0 unload
```

Assuming that the stacker mode of the tape device is "system" and that a tape is present in the stacker, you can load a new tape by issuing:

```
# tape390_display -l "NEW TAPE" /dev/ntibm0
```

"NEW TAPE" is a message that is displayed on the tape devices display unit until the tape device receives the next tape movement command.

Part 4. Networking

Red Hat Enterprise Linux 9.1 includes several network device drivers that are specific to z/Architecture.

For information about high-performing, secure networking and connectivity, see www.ibm.com/it-infrastructure/z/capabilities/networking

Newest version

You can find the newest version of this book at ibm.com/docs/en/linux-on-systems?topic=linuxone-distributions

Restrictions

For prerequisites and restrictions see the IBM Z architecture specific information in the Red Hat Enterprise Linux 9.1 release notes at https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

Example

An example network setup that uses some available network setup types is shown in Figure 54 on page 239.



Figure 54. Networking example

In the example there are three Linux instances; two of them run as z/VM guests in one LPAR and a third Linux instance runs in another LPAR. Within z/VM, Linux instances can be connected through a guest LAN or VSWITCH. Within and between LPARs, you can connect Linux instances through HiperSockets. OSA-Express cards running in either non-QDIO mode (called LCS here) or in QDIO mode can connect the mainframe to an external network.

Table 36 on page 240 lists which control units and device type combinations are supported by the network device drivers.

Table 36. Supported device types, control units, and corresponding device drivers

Device type	Control unit	Device driver	Comment
1732/01	1731/01	qeth	OSA configured as OSD
1732/02	1731/02	qeth	OSA configured as OSX
1732/05	1731/05	qeth	HiperSockets
0000/00	3088/01	lcs	P/390
0000/00	3088/08	ctcm	Virtual CTC under z/VM
0000/00	3088/1e	ctcm	FICON channel
0000/00	3088/1f	lcs	2216 Nways Multiaccess Connector
0000/00	3088/1f	ctcm	ESCON channel
0000/00	3088/60	lcs	OSA configured as OSE (non-QDIO)

Chapter 16. qeth device driver for OSA-Express (QDIO) and HiperSockets

LPAR and z/VM: The qeth device driver applies to Linux in LPAR mode and to Linux on z/VM.

The qeth device driver supports a multitude of network connections, for example, connections through Open Systems Adapters (OSA), HiperSockets, guest LANs, and virtual switches.

Real connections that use OSA-Express

An IBM mainframe uses OSA-Express adapters, which are real LAN-adapter hardware, see Figure 55 on page 241. These adapters provide connections to the outside world, but can also connect virtual systems (between LPARs or between z/VM guest virtual machines) within the mainframe. The qeth driver supports these adapters if they are defined to run in queued direct I/O (QDIO) mode (defined as OSD or OSX in the hardware configuration). OSD-devices are the standard IBM Z LAN-adapters. For details about OSA-Express in QDIO mode, see *Open Systems Adapter-Express Customer's Guide and Reference*, SA22-7935.



Figure 55. OSA-Express adapters are real LAN-adapter hardware

The qeth device driver supports OSA-Express features for the IBM Z mainframes that are relevant to Red Hat Enterprise Linux 9.1 as shown in Table 37 on page 241:

Table 37. The qeth device driver support for OSA-Express features

Feature	IBM z16	IBM z15™	z14 and z14 ZR1	z13 and z13s
OSA-Express7S	Gigabit Ethernet 10 Gigabit Ethernet 25 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 25 Gigabit Ethernet 1000Base-T Ethernet	25 Gigabit Ethernet	Not supported

Table 37. The qeth device driver support for OSA-Express features (continued)				
Feature	**IBM z16**	**IBM z15™**	**z14 and z14 ZR1**	**z13 and z13s**
OSA-Express6S	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet
OSA-Express5S	Not supported	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet
OSA-Express4S	Not supported	Not supported	1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet

Note: Unless otherwise indicated, OSA-Express refers to the OSA-Express features as shown in Table 37 on page 241.

The qeth device driver supports CHPIDs of type OSD and OSX:

OSD
provides connectivity as the standard IBM Z LAN adapter type, running in either layer 3 or layer 2 mode. When running in layer 3 mode, only IP traffic is supported, using IP addresses. When running in layer 2 mode, the traffic is protocol-independent, using MAC addresses.

OSX
Supported up to z14, provides connectivity to and access control for the intraensemble data network (IEDN), which is managed by Unified Resource Manager functions. A zEnterprise® CPC and zBX within an ensemble are connected through the IEDN. See *zEnterprise System Introduction to Ensembles*, GC27-2609 and *zEnterprise System Ensemble Planning and Configuring Guide*, GC27-2608 for more details.

HiperSockets
An IBM mainframe uses internal connections that are called *HiperSockets*. These simulate QDIO network adapters and provide high-speed TCP/IP communication for operating system instances within and across LPARs. For details about HiperSockets, see *HiperSockets Implementation Guide*, SG24-6816.

HiperSockets Converged Interface (HSCI)
With HSCI, you can integrate HiperSockets connectivity with your external LAN, thus creating a single logical network interface. The single interface simplifies network management.

Virtual connections for Linux on z/VM
z/VM offers virtualized LAN-adapters that enable connections between z/VM guest virtual machines and the outside world. It allows definitions of simulated network interface cards (NICs) attached to certain z/VM guest virtual machines. The NICs can be connected to a simulated LAN segment called *guest LAN* for z/VM internal communication between z/VM guest virtual machines, or they can be connected to a virtual switch called *VSWITCH* for external LAN connectivity.

Guest LAN
Guest LANs represent a simulated LAN segment that can be connected to simulated network interface cards. There are three types of guest LANs:

- Simulated OSA-Express in layer 3 mode
- Simulated HiperSockets (layer 3) mode
- Simulated Ethernet in layer 2 mode

Each guest LAN is isolated from other guest LANs on the same system (unless some member of one LAN group acts as a router to other groups). See Figure 56 on page 243.



Figure 56. Guest LAN

Virtual switch

A virtual switch (VSWITCH) is a special-purpose guest LAN that provides external LAN connectivity through an additional OSA-Express device served by z/VM without the need for a routing virtual machine, see Figure 57 on page 243.



Figure 57. Virtual switch

A dedicated OSA adapter can be an option, but is not required for a VSWITCH.

HiperSockets bridge port

A HiperSockets bridge port connects a network defined by a virtual switch to a HiperSockets LAN. The two networks are combined into one logical network. If the VSWITCH is connected to an external Ethernet LAN, the HiperSockets LAN can then communicate outside the CEC as shown

in Figure 58 on page 244. You can thus connect a HiperSockets LAN to an external LAN without using a router.



Figure 58. HiperSockets bridge port in z/VM

For more information about bridge ports, see "Layer 2 promiscuous mode" on page 253, and Figure 62 on page 253.

From a Linux point of view there is no difference between guest LAN- and VSWITCH-devices; thus Linux talks about guest LAN-devices independently of their z/VM-attachment to a guest LAN or VSWITCH.

For information about guest LANs, virtual switches, and virtual HiperSockets, as well as about attaching OSD or HiperSockets to z/VM guests directly, see *z/VM: Connectivity*, SC24-6267.

Device driver functions

The qeth device driver supports many networking transport protocol functions, as well as offload functions and problem determination functions.

The qeth device driver supports functions listed in Table 38 on page 244 and Table 39 on page 246.

Table 38. Real connections

Function	OSA Layer 2	OSA Layer 3	HiperSockets Layer 2	HiperSockets Layer 3
Basic device or protocol functions				
IPv4/multicast/broadcast	Yes/Yes/Yes	Yes/Yes/Yes	Yes/Yes/Yes	Yes/Yes/Yes
IPv6/multicast	Yes/Yes	Yes/Yes	Yes/Yes	Yes/Yes
Non-IP traffic	Yes	Yes	Yes	No
VLAN IPv4/IPv6/non IP	sw/sw/sw	hw/sw/sw	sw/sw/sw	hw/hw/No
Linux ARP	Yes	No (hw ARP)	Yes	No
Linux neighbor solicitation	Yes	Yes	Yes	No
Unique MAC address	Yes (random for LPAR)	No	Yes	Yes
Change MAC address	Yes	No	Yes	No

Table 38. Real connections (continued)

Function	OSA Layer 2	OSA Layer 3	HiperSockets Layer 2	HiperSockets Layer 3
Promiscuous mode	Yes (bridgeport or VNIC characteristics: flooding)	No	Yes (bridgeport or VNIC characteristics: flooding)	• Yes (for sniffer=1) • No (for sniffer=0)
MAC headers send/receive	Yes/Yes	faked/faked	Yes/Yes	faked/faked
ethtool support	Yes	Yes	Yes	Yes
Bonding	Yes	No	Yes	No
Priority queueing	Yes	Yes	No	No
Secondary unicast MAC address	Yes	No	Yes	No
Bridge port	Yes	No	Yes	No
Offload features				
TCP segmentation offload (TSO)	Yes	Yes	No	No
Inbound (rx) checksum	Yes	Yes	No	No
Outbound (tx) checksum	Yes	Yes	No	No
OSA/QETH specific features				
Special device driver setup for VIPA	No	required	No	Yes
Special device driver setup for proxy ARP	No	required	No	Yes
Special device driver setup for IP takeover	No	required	No	Yes
Special device driver setup for routing IPv4/IPv6	No/No	required/ required	No/No	Yes/Yes
Receive buffer count	Yes	Yes	Yes	Yes
Direct connectivity to z/OS	Yes by HW	Yes	No	Yes
SNMP support	Yes	Yes	No	No
Multiport support	Yes	Yes	No	No
Data connection isolation	Yes	Yes	No	No
Problem determination				
Hardware trace	Yes	Yes	No	No

Table 38. Real connections (continued)

Function	OSA Layer 2	OSA Layer 3	HiperSockets Layer 2	HiperSockets Layer 3

Legend:

No - Function not supported or not required.
Yes - Function supported.
hw - Function performed by hardware.
sw - Function performed by software.
faked - Function will be simulated.
required - Function requires special setup.

Table 39. z/VM VSWITCH or Guest LAN connections

Function	Emulated OSA Layer 2	Emulated OSA Layer 3	Emulated HiperSockets Layer 3
Basic device or protocol features			
IPv4/multicast/broadcast	Yes/Yes/Yes	Yes/Yes/Yes	Yes/Yes/Yes
IPv6/multicast	Yes/Yes	Yes/Yes	No/No
Non-IP traffic	Yes	No	No
VLAN IPv4/IPv6/non IP	sw/sw/sw	hw/sw/No	hw/No/No
Linux ARP	Yes	No (hw ARP)	No
Linux neighbor solicitation	Yes	Yes	No
Unique MAC address	Yes	Yes	Yes
Change MAC address	Yes	No	No
Promiscuous mode	Yes	Yes	No
MAC headers send/receive	Yes/Yes	faked/faked	faked/faked
ethtool support	Yes	Yes	Yes
Bonding	Yes	No	No
Priority queueing	No	No	No
Secondary unicast MAC address	Yes	No	No
Offload features	No	No	No
OSA/QETH specific features			
Special device driver setup for VIPA	No	required	required
Special device driver setup for proxy ARP	No	required	required
Special device driver setup for IP takeover	No	required	required
Special device driver setup for routing IPv4/IPv6	No/No	required/required	required/required
Receive buffer count	Yes	Yes	Yes

Table 39. z/VM VSWITCH or Guest LAN connections (continued)			
Function	**Emulated OSA Layer 2**	**Emulated OSA Layer 3**	**Emulated HiperSockets Layer 3**
Direct connectivity to z/OS	No	Yes	Yes
SNMP support	No	No	No
Multiport support	No	No	No
Data connection isolation	No	No	No
Problem determination			
Hardware trace	No	No	No

Legend:

No - Function not supported or not required.
Yes - Function supported.
hw - Function performed by hardware.
sw - Function performed by software.
faked - Function will be simulated.
required - Function requires special setup.

What you should know about the qeth device driver

Interface names are assigned to qeth group devices, which map to subchannels and their corresponding device numbers and device bus-IDs. An OSA-Express adapter can handle both IPv4 and IPv6 packets.

Layer 2 and layer 3

The qeth device driver consists of a common core and two device disciplines: layer 2 and layer 3.

In layer 2 mode, OSA routing to the destination is based on MAC addresses. A local MAC address is assigned to each interface of a Linux instance and registered in the OSA Address Table. These MAC addresses are unique and different from the MAC address of the OSA adapter. See “MAC headers in layer 2 mode” on page 249 for details.

In layer 3 mode, all interfaces of all Linux instances share the MAC address of the OSA adapter. OSA routing to the destination Linux instance is based on IP addresses. See “MAC headers in layer 3 mode” on page 250 for details.

The layer 2 discipline (qeth_l2)

The layer 2 discipline supports:

- OSA devices and z/VM virtual NICs that couple to VSWITCHes or QDIO guest LANs running in layer 2 mode
- HiperSockets devices
- OSX (OSA-Express for zBX) devices for IEDN

The layer 2 discipline is the default setup for OSA. On HiperSockets the default continues to be layer 3. See “Setting the layer2 attribute” on page 260 for details.

For z/VM NICs that are coupled to a guest LAN or VSWITCH, the qeth device driver detects the required layer and configures it automatically. If a qeth device is created before the NIC is coupled, the qeth device driver defaults to layer 2.

The layer 3 discipline (qeth_l3)

The layer 3 discipline supports:

- OSA devices and z/VM virtual NICs that couple to VSWITCHes or QDIO guest LANs that are running in layer 3 mode (with faked link layer headers)
- HiperSockets and HiperSockets guest LAN devices that are running in layer 3 mode (with faked link layer headers)
- OSX (OSA-Express for zBX) devices for IEDN

This discipline supports those devices that are not capable of running in layer 2 mode. Not all Linux networking features are supported and others need special setup or configuration. See Table 45 on page 257. Some performance-critical applications might benefit from being layer 3.

Layer 2 and layer 3 interfaces cannot communicate within a HiperSockets LAN or within a VSWITCH or guest LAN. However, a shared OSA adapter can convert traffic between layer 2 and layer 3 networks.

qeth group devices

The qeth device driver requires three I/O subchannels for each HiperSockets CHPID or OSA-Express CHPID in QDIO mode. One subchannel is for control reads, one for control writes, and the third is for data.

The qeth device driver uses the QDIO protocol to communicate with the HiperSockets and OSA-Express adapter.



Figure 59. I/O subchannel interface

The three device bus-IDs that correspond to the subchannel triplet are grouped as one qeth group device. The following rules apply for the device bus-IDs:

read
no specific rules.

write
must be the device bus-ID of the read subchannel plus one.

data
can be any free device bus-ID on the same CHPID.

You can configure different triplets of device bus-IDs on the same CHPID differently. For example, if you have two triplets on the same CHPID they can have different attribute values for priority queueing.

Overview of the steps for setting up a qeth group device

You need to perform several steps before user-space applications on your Linux instance can use a qeth group device.

Before you begin

Find out how the hardware is configured and which qeth device bus-IDs are on which CHPID, for example by looking at the IOCDS. Identify the device bus-IDs that you want to group into a qeth group device. The three device bus-IDs must be on the same CHPID.

Procedure

Perform these steps to allow user-space applications on your Linux instance to use a qeth group device:

1. Create the qeth group device.

After booting Linux, each qeth device bus-ID is represented by a subdirectory in `/sys/bus/ccw/devices/`. These subdirectories are then named with the bus IDs of the devices. For example, a qeth device with bus IDs 0.0.fc00, 0.0.fc01, and 0.0.fc02 is represented as `/sys/bus/ccw/drivers/qeth/0.0.fc00`

2. Configure the device.
3. Set the device online.
4. Activate the device and assign an IP address to it.

What to do next

These tasks and the configuration options are described in detail in "Working with qeth devices" on page 255.

qeth interface names and device directories

Red Hat Enterprise Linux automatically assigns interfaces to the qeth group devices. The qeth device driver creates the corresponding sysfs structures.

An interface is represented in sysfs as:

```
/sys/class/net/<interface>
```

The mapping between interfaces and the device bus-ID that represents the qeth group device in sysfs is preserved when a device is set offline and back online.

Note: The interface is represented in sysfs even if the device is offline.

"Finding out the interface name of a qeth group device" on page 266 and "Finding out the bus ID of a qeth interface" on page 267 provide information about mapping device bus-IDs and interfaces.

Support for IP Version 6 (IPv6)

The qeth device driver supports IPv6 in many network setups.

IPv6 is supported on:

- Ethernet interfaces of the OSA-Express adapter that runs in QDIO mode.
- HiperSockets layer 2 and layer 3 interfaces.
- z/VM guest LAN interfaces running in QDIO or HiperSockets layer 3 mode.
- z/VM guest LAN and VSWITCH interfaces in layer 2.

There are noticeable differences between the IP stacks for versions 4 and 6. Some concepts in IPv6 are different from IPv4, such as neighbor discovery, broadcast, and Internet Protocol security (IPsec). IPv6 uses a 16-byte address field, while the addresses under IPv4 are 4 bytes in length.

Stateless autoconfiguration generates unique IP addresses for all Linux instances, even if they share an OSA-Express adapter with other operating systems.

Be aware of the IP version when you specify IP addresses and when you use commands that return IP version-specific output (such as **qetharp**).

MAC headers in layer 2 mode

In LAN environments, data packets find their destination through Media Access Control (MAC) addresses in their MAC header.



Figure 60. Standard IPv4 processing

MAC address handling as shown in Figure 60 on page 250) applies to non-mainframe environments and a mainframe environment with an OSA-Express adapter where the layer2 option is enabled.

The layer2 option keeps the MAC addresses on incoming packets. Incoming and outgoing packets are complete with a MAC header at all stages between the Linux network stack and the LAN as shown in Figure 60 on page 250. This layer2-based forwarding requires unique MAC addresses for all concerned Linux instances.

In layer 2 mode, the Linux TCP/IP stack has full control over the MAC headers and the neighbor lookup. The Linux TCP/IP stack does not configure IPv4 or IPv6 addresses into the hardware, but requires a unique MAC address for the card.

When Linux is running as a z/VM guest, the qeth device driver obtains a MAC address for each L2 device from the z/VM host. No configuration is necessary.

When Linux is running in an LPAR and you work with a directly attached OSA adapter in QDIO mode, you should assign a unique MAC address.

To assign a MAC address, add a line `MACADDR='<MAC address>'` to the configuration file `/etc/sysconfig/network-scripts/ifcfg-<if-name>`. Alternatively, you can set the MAC address by issuing the command:

```
ip link set addr <MAC address> dev <interface>
```

Note: Be sure not to assign the MAC address of the OSA-Express adapter to your Linux instance.

For OSX CHPIDs, you cannot set your own MAC addresses. Linux uses the MAC addresses defined by the Unified Resource Manager.

For HiperSockets connections, a MAC address is generated. For most purposes the generated address is adequate. However, you can change the address by using the **ip** command if you need to.

MAC headers in layer 3 mode

A qeth layer 3 mode device driver is an Ethernet offload engine for IPv4 and a partial Ethernet offload engine for IPv6. Hence, there are some special things to understand about the layer 3 mode.

To support IPv6 and protocols other than IPv4, the device driver registers a layer 3 card as an Ethernet device to the Linux TCP/IP stack.

In layer 3 mode, the OSA-Express adapter in QDIO mode removes the MAC header with the MAC address from incoming IPv4 packets. It uses the registered IP addresses to forward a packet to the recipient TCP/IP stack.See Figure 61 on page 251. Thus the OSA-Express adapter is able to deliver IPv4 packets

to the correct Linux images. Apart from broadcast packets, a Linux image can get only packets for IP addresses it configured in the stack and registered with the OSA-Express adapter.



Figure 61. MAC address handling in layer3 mode

The OSA-Express QDIO microcode builds MAC headers for outgoing IPv4 packets and removes them from incoming IPv4 packets. Hence, the operating systems' network stacks send and receive only IPv4 packets without MAC headers.

This lack of MAC headers can be a problem for applications that expect MAC headers. For examples of how such problems can be resolved, see "Setting up for DHCP with IPv4" on page 303.

Outgoing frames

The qeth device driver registers the layer 3 card as an Ethernet device. Therefore, the Linux TCP/IP stack will provide complete Ethernet frames to the device driver.

If the hardware does not require the Ethernet frame (for example, for IPv4) the driver removes the Ethernet header prior to sending the frame to the hardware. If necessary information like the Ethernet target address is not available (because of the offload functionality) the value is filled with the hardcoded address FAKELL.

Table 40. Ethernet addresses of outgoing frames

Frame	**Destination address**	**Source address**
IPv4	FAKELL	Real device address
IPv6	Real destination address	Real device address
Other packets	Real destination address	Real device address

Incoming frames

The device driver provides Ethernet headers for all incoming frames.

If necessary information like the Ethernet source address is not available (because of the offload functionality) the value is filled with the hardcoded address FAKELL.

Table 41. Ethernet addresses of incoming frames

Frame	**Destination address**	**Source address**
IPv4	Real device address	FAKELL
IPv6	Real device address	FAKELL

Table 41. Ethernet addresses of incoming frames (continued)		
Frame	**Destination address**	**Source address**
Other packets	Real device address	Real source address

Note that if a source or destination address is a multicast or broadcast address the device driver can provide the corresponding (real) Ethernet multicast or broadcast address even when the packet was delivered or sent through the offload engine. Always providing the link layer headers enables packet socket applications like **tcpdump** to work properly on a qeth layer 3 device without any changes in the application itself (the patch for libpcap is no longer required).

While the faked headers are syntactically correct, the addresses are not authentic, and hence applications requiring authentic addresses will not work. Some examples are given in Table 42 on page 252.

Table 42. Applications that react differently to faked headers		
Application	**Support**	**Reason**
tcpdump	Yes	Displays only frames, fake Ethernet information is displayed.
iptables	Partially	As long as the rule does not deal with Ethernet information of an IPv4 frame.
dhcp	Yes	Is non-IPv4 traffic.

IP addresses

The network stack of each operating system that shares an OSA-Express adapter in QDIO mode registers all its IP addresses with the adapter.

Whenever IP addresses are deleted from or added to a network stack, the device drivers download the resulting IP address list changes to the OSA-Express adapter.

For the registered IP addresses, the OSA-Express adapter off-loads various functions, in particular also:

- Handling MAC addresses and MAC headers
- ARP processing

ARP

The OSA-Express adapter in QDIO mode responds to Address Resolution Protocol (ARP) requests for all registered IPv4 addresses.

ARP is a TCP/IP protocol that translates 32-bit IPv4 addresses into the corresponding hardware addresses. For example, for an Ethernet device, the hardware addresses are 48-bit Ethernet Media Access Control (MAC) addresses. The mapping of IPv4 addresses to the corresponding hardware addresses is defined in the ARP cache. When it needs to send a packet, a host consults the ARP cache of its network adapter to find the MAC address of the target host.

If there is an entry for the destination IPv4 address, the corresponding MAC address is copied into the MAC header and the packet is added to the appropriate interface's output queue. If the entry is not found, the ARP functions retain the IPv4 packet, and broadcast an ARP request asking the destination host for its MAC address. When a reply is received, the packet is sent to its destination.

Notes:

1. On an OSA-Express adapter in QDIO mode, do not set the NO_ARP flag on the Linux Ethernet device. The device driver disables the ARP resolution for IPv4. Because the hardware requires no neighbor lookup for IPv4, but neighbor solicitation for IPv6, the NO_ARP flag is not allowed on the Linux Ethernet device.
2. On HiperSockets, which is a full Ethernet offload engine for IPv4 and IPv6 and supports no other traffic, the device driver sets the NO_ARP flag on the Linux Ethernet interface. Do not remove this flag from the interface.

Layer 2 promiscuous mode

OSA and HiperSockets ports that operate in layer 2 mode can be set up to receive all frames that are addressed to unknown MAC addresses.

On most architectures, traffic between operating systems and networks is handled by Ethernet Network Interface Controllers (NICs). NICs usually filter incoming traffic to admit only frames with destination MAC addresses that are registered with the NIC.

However, a NIC can also be configured to receive and pass to the operating system all Ethernet frames that reach it, regardless of the destination MAC address. This mode of operation is known as "promiscuous mode". For example, promiscuous mode is a prerequisite for configuring a NIC as a member of a Linux software bridge.

For more information about how to set up a software bridge, see the documentation that is provided by Red Hat Enterprise Linux, or the bridging how-to available at http://www.tldp.org/HOWTO/BRIDGE-STP-HOWTO

On IBM Z, you can realize a promiscuous mode for Ethernet traffic through a bridge port configuration or through Virtual Network Interface Controller (VNIC) characteristics.

Depending on the hardware level, OSA and HiperSockets devices can be configured as bridge ports or they can be configured with VNIC characteristics. The same OSA or HiperSockets device cannot simultaneously be configured as a bridge port and with VNIC characteristics.

VNIC characteristics

With (VNIC) characteristics, you can set and fine-tune a promiscuous mode for HiperSockets and OSA devices, see “Advanced packet-handling configuration” on page 287.

Bridge ports

Linux can assign a bridge port *role* to a logical port, and the HiperSockets or OSA adapter assigns an active *state* to one of the logical ports to which a role was assigned. A local port in active bridge port state receives all Ethernet frames with unknown destination MAC addresses.

Figure 62 on page 253 shows a setup with a HiperSockets bridge port and an OSA bridge port.



Figure 62. HiperSockets and OSA bridge port in Linux

HiperSockets only: On IQDX channels permission to configure ports as bridge ports must be granted in IBM zEnterprise Unified Resource Manager (zManager). On machines in PR/SM mode, bridge ports can only be configured on IQD channels that are defined as "external-bridged" in the IOCDS. On machines in DPM mode, bridge ports can be configured on any IQD channel.

Differences between promiscuous mode and bridge-port roles

Making a logical port of an OSA or HiperSockets adapter an active bridge port is similar to enabling promiscuous mode on a non-mainframe NIC that is connected to a real Ethernet switch. However, there are important differences:

Number of ports in promiscuous mode

- Real switches: Any number of interfaces that are connected to a real switch can be turned to promiscuous mode, and all of them then receive frames with unknown destination addresses.
- Bridge ports on IBM Z: Although you can assign the bridge-port role to multiple ports of a single OSA or HiperSockets adapter, only one port is active and receives traffic to unknown destinations.

Monitoring traffic to other systems

- Real switches: A port of a real switch can be configured to receive frames with both known and unknown destinations. If a NIC in promiscuous mode is connected to the port, the corresponding host receives a copy of all traffic that passes through the switch. This includes traffic that is destined to other hosts connected to this switch.
- Bridge ports on IBM Z or qeth devices with `vnicc/flooding` set: Only frames with unknown destinations are passed to the operating system. It is not possible to intercept traffic addressed to systems connected to other ports of the same OSA adapter.
- On IBM Z: The HiperSockets network traffic analyzer or z/VM guest LAN sniffer can be used to monitor traffic that is destined for other ports.

Limitation by the source of traffic (OSA bridge port only)

- Real switches and HiperSockets bridge-port LAN: Frames with unknown destination MAC addresses are delivered to the promiscuous interfaces regardless of the port through which the frames enter the switch or HiperSockets adapter.
- OSA bridge ports or OSA ports that are set to `vnicc/flooding` and `vnicc/learning`: Active bridge ports or ports with the flooding and learning VNIC characteristics *learn* which MAC addresses need to be routed to the owning system by analyzing ARP and other traffic. Incoming frames are routed to these ports if one of the following conditions applies:
  - The frame's destination MAC matches an address that is learned or registered with the port.
  - The frame's destination MAC is not learned or registered with any of the local ports of the OSA adapter, and it arrived from the physical Ethernet port.

Bridge port roles

Linux can assign a primary or secondary role to a logical port of an OSA or a HiperSockets adapter. Only one logical port of such an adapter can be assigned the primary role, but multiple other logical ports can be assigned secondary role. When one or more logical ports of an adapter are assigned primary or secondary role, the hardware ensures that exactly one of these ports is active. The active port receives frames with unknown destination. When a port with primary role is present, it always becomes active. When only ports with secondary role are present, the hardware decides which one becomes active. Changes in the ports' state are reported to Linux user space through udev events.

You can set a bridge port role either directly by using the **`bridge_role`** attribute or indirectly by using the **`bridge_reflect_promisc`** attribute. See “Configuring a network device as a member of a Linux bridge” on page 284.

Setting up the qeth device driver

No module parameters exist for the qeth device driver. qeth devices are set up using sysfs.

Loading the qeth device driver modules

You must load the qeth device driver before you can work with qeth devices.

Use the **modprobe** command to load the qeth device driver, and to automatically load all required additional modules in the correct order:

qeth module syntax



where:

qeth
is the core module that contains common functions that are used for both layer 2 and layer 3 disciplines.

qeth_l2
is the module that contains layer 2 discipline-specific code.

qeth_l3
is the module that contains layer 3 discipline-specific code.

When a qeth device is configured for a particular discipline, the driver tries to automatically load the corresponding discipline module.

Switching the discipline of a qeth device

To switch the discipline of a device, the network interface must be shut down and the device must be offline.

Some devices can only run in one discipline, see “Layer 2 and layer 3” on page 247. The device driver rejects any request to switch the discipline of these devices.

If the new discipline is accepted by the device driver, the old network interface is deleted. When the new discipline is set online the first time, the new network interface is created.

Removing the modules

Removing a module is not possible if there are cross dependencies between the discipline modules and the core module.

To release the dependencies from the core module to the discipline module, all devices of this discipline must be ungrouped. Now the discipline module can be removed. If all discipline modules are removed, the core module can be removed.

Working with qeth devices

Typical tasks that you need to perform when working with qeth devices include creating group devices, finding out the type of a network adapter, and setting a device online or offline.

About this task

Most of these tasks involve writing to and reading from attributes of qeth group devices in sysfs. This is useful on a running system where you want to make dynamic changes. If you want to make the

changes persistent across IPLs, use the interface configuration files. Network configuration parameters are defined in `/etc/sysconfig/network-scripts/ifcfg-<if_name>`. An example of how to define a qeth device persistently is in the installation documentation on the Red Hat website. For a general discussion of network configuration files, see configuration documentation on the Red Hat documentation website
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

Table 43 on page 256 and Table 45 on page 257 serve as both a task overview and a summary of the attributes and the possible values you can write to them. Underlined values are defaults.

Tip: Use the **chzdev** command to configure devices instead of using the attributes directly (see"chzdev - Configure IBM Z devices" on page 590). You can also use the **znetconf** command for network devices.

Not all attributes are applicable to each device. Some attributes apply only to HiperSockets or only to OSA-Express CHPIDs in QDIO mode, other attributes are applicable to IPv4 interfaces only. See the task descriptions for the applicability of each attribute.

Table 43. qeth tasks and attributes common to layer2 and layer3

Task	Corresponding attributes	Possible attribute values
"Creating a qeth group device" on page 259	group	n/a
"Removing a qeth group device" on page 260	ungroup	0 or 1
"Setting the layer2 attribute" on page 260	layer2	0 or 1, see "Layer 2 and layer 3" on page 247[1]
"Using priority queueing" on page 261	priority_queueing	prio_queueing_vlan prio_queueing_skb prio_queueing_prec prio_queueing_tos no_prio_queueing no_prio_queueing:0 no_prio_queueing:1 no_prio_queueing:2 no_prio_queueing:3
"Specifying the number of inbound buffers" on page 263	buffer_count	integer in the range 8 - 128. The default is 64 for OSA devices and 128 for HiperSockets devices
"Specifying the relative port number" on page 264	portno	integer, either 0 or 1, the default is 0
"Finding out the type of your network adapter" on page 265	card_type	n/a, read-only
"Setting a device online or offline" on page 265	online	0 or 1
"Finding out the interface name of a qeth group device" on page 266	if_name	n/a, read-only
"Finding out the bus ID of a qeth interface" on page 267	none	n/a
"Activating an interface" on page 267	none	n/a
"Deactivating an interface" on page 269	none	n/a
"Recovering a device" on page 269	recover	1

Table 43. qeth tasks and attributes common to layer2 and layer3 (continued)

Task	Corresponding attributes	Possible attribute values
"Enabling and disabling TCP segmentation offload" on page 271	none	n/a
"Configuring the receive checksum offload feature" on page 270	none	n/a
"Configuring the transmit checksum offload feature" on page 270	none	n/a
"Isolating data connections" on page 272	isolation	none, drop, forward
"Displaying and resetting QETH performance statistics" on page 274	performance_stats	0 or 1
"Capturing a hardware trace" on page 274	hw_trap	arm disarm

[1]A value of -1 means that the layer has not been set and that the default layer setting is used when the device is set online.

Table 44. qeth functions and attributes in layer 2 mode

Function	Corresponding attributes	Possible attribute values
"Configuring a network device as a member of a Linux bridge" on page 284	bridge_role bridge_state bridge_hostnotify	primary, secondary, none active, standby, inactive 0 or 1
"Advanced packet-handling configuration" on page 287	vnicc/bridge_invisible vnicc/flooding vnicc/learning vnicc/mcast_flooding vnicc/rx_bcast vnicc/takeover_learning vnicc/takeover_setvmac vnicc/learning_timeout	0 or 1 0 or 1 0 or 1 0 or 1 0 or 1 0 or 1 0 or 1 integer in the range 60 - 86400 the default is 600

Table 45. qeth tasks and attributes in layer 3 mode

Task	Corresponding attributes	Possible attribute values
"Setting up a Linux router in layer 3" on page 275	route4 route6	primary_router secondary_router primary_connector secondary_connector multicast_router no_router
"Faking broadcast capability" on page 278	fake_broadcast [1]	0 or 1

Table 45. qeth tasks and attributes in layer 3 mode (continued)

Task	Corresponding attributes	Possible attribute values
"Taking over IP addresses" on page 279	ipa_takeover/enable	0 or 1 or toggle
	ipa_takeover/add4 ipa_takeover/add6 ipa_takeover/del4 ipa_takeover/del6	IPv4 or IPv6 IP address and mask bits
	ipa_takeover/invert4 ipa_takeover/invert6	0 or 1 or toggle
"Configuring a device for proxy ARP" on page 282	rxip/add4 rxip/del4	IPv4 IP address
Configuring a device for NDP proxy	rxip/add6 rxip/del6	IPv6 IP address
"Configuring a device for virtual IP address (VIPA)" on page 283	vipa/add4 vipa/add6 vipa/del4 vipa/del6	IPv4 or IPv6 IP address
"Configuring a HiperSockets device for AF_IUCV addressing" on page 283	hsuid	1 to 8 characters
"Setting up a HiperSockets network traffic analyzer" on page 304	sniffer	0 or 1

[1] not valid for HiperSockets

Tip: Use the **qethconf** command instead of using the attributes for IPA, proxy ARP, and VIPA directly (see "qethconf - Configure qeth devices" on page 708).

sysfs provides multiple paths through which you can access the qeth group device attributes. For example, if a device with bus ID 0.0.a100 corresponds to interface enca100:

```
/sys/bus/ccwgroup/drivers/qeth/0.0.a100
/sys/bus/ccwgroup/devices/0.0.a100
/sys/devices/qeth/0.0.a100
/sys/class/net/enca100/device
```

all lead to the attributes for the same device. For example, the following commands are all equivalent and return the same value:

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a100/if_name
enca100
# cat /sys/bus/ccwgroup/devices/0.0.a100/if_name
enca100
# cat /sys/devices/qeth/0.0.a100/if_name
enca100
# cat /sys/class/net/enca100/device/if_name
enca100
```

The path through `/sys/class/net` becomes available when the device is first set online and the interface is created. The path persists until the device is ungrouped. Furthermore, it might lead to a different device if the assignment of interface names changes. A change can occur after rebooting or when devices are ungrouped and new group devices created.

Tip: Work through one of the paths that are based on the device bus-ID.

The following sections describe the tasks in detail.

Creating a qeth group device

Use the **znetconf** command to configure network devices. Alternatively, you can use the **chzdev** command or sysfs.

Before you begin

You must know the device bus-IDs that correspond to the read, write, and data subchannel of your OSA-Express CHPID in QDIO mode or HiperSockets CHPID as defined in the IOCDS of your mainframe.

Procedure

To create a qeth group device, either:

- Issue the **znetconf** command to create and configure a group device. The command groups the correct bus-IDs for you and sets the device online.

  For information about the **znetconf** command, see "znetconf - List and configure network devices" on page 767.
- Write the device numbers of the subchannel triplet to the sysfs `group` attribute to only define a group device.

  Issue a command of the form:

```
# echo <read_device_bus_id>,<write_device_bus_id>,<data_device_bus_id> > /sys/bus/ccwgroup/drivers/qeth/group
```

Results

The qeth device driver uses the device bus-ID of the read subchannel to create a directory for a group device:

```
/sys/bus/ccwgroup/drivers/qeth/<read_device_bus_id>
```

This directory contains a number of attributes that determine the settings of the qeth group device. The following sections describe how to use these attributes to configure a qeth group device.

Example

In this example (see Figure 63 on page 259), a single OSA-Express CHPID in QDIO mode is used to connect a Linux instance to a network.

Mainframe configuration:



Figure 63. Mainframe configuration

Linux configuration:

Assuming that 0.0.aa00 is the device bus-ID that corresponds to the read subchannel:

```
# echo 0.0.aa00,0.0.aa01,0.0.aa02 > /sys/bus/ccwgroup/drivers/qeth/group
```

This command results in the creation of the following directories in sysfs:

- `/sys/bus/ccwgroup/drivers/qeth/0.0.aa00`
- `/sys/bus/ccwgroup/devices/0.0.aa00`
- `/sys/devices/qeth/0.0.aa00`

Both the command and the resulting directories would be the same for a HiperSockets CHPID.

Removing a qeth group device

Use the `ungroup` sysfs attribute to remove a qeth group device.

Before you begin

The device must be set offline before you can remove it.

Procedure

To remove a qeth group device, write 1 to the `ungroup` attribute.

Issue a command of the form:

```
echo 1 > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/ungroup
```

Example

This command removes device 0.0.aa00:

```
echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.aa00/ungroup
```

Setting the layer2 attribute

If the detected hardware always runs in a specific discipline, the corresponding discipline module is automatically requested.

Before you begin

- To change a configured `layer2` attribute, the network interface must be shut down and the device must be set offline.
- If you are using the layer2 option within a QDIO-based guest LAN environment, you cannot define a VLAN with ID 1, because ID 1 is reserved for z/VM use.
- IQD channels that in the IOCDS are defined as "external-bridged" must be configured to use layer 2.

About this task

The qeth device driver attempts to load the layer 3 discipline for HiperSockets devices and layer 2 for non-HiperSockets devices.

You can use the layer 2 mode for almost all device types, however, note the following about layer 2 to layer 3 conversion:

real OSA-Express
: Hardware is able to convert layer 2 to layer 3 traffic and vice versa and thus there are no restrictions.

HiperSockets
: There is no support for layer 2 to layer 3 conversion and, thus, no communication is possible between HiperSockets layer 2 interfaces and HiperSockets layer 3 interfaces. Do not include HiperSockets layer 2 interfaces and HiperSockets layer 3 interfaces in the same LAN.

z/VM guest LAN
The qeth device driver detects the mode of the VSWITCH or LAN to which the NIC is coupled, and sets this mode on the device. The z/VM definition "Ethernet mode" is available for VSWITCHes and for guest LANs of type QDIO.

Procedure

The qeth device driver separates the configuration options in sysfs according to the device discipline. Hence the first configuration action after you group the device must be the configuration of the discipline.

To set the discipline, issue a command of the form:

```
chzdev -a <device_bus_id> layer2=<integer>
```

or, using sysfs:

```
echo <integer> > /sys/devices/qeth/<device_bus_id>/layer2
```

where *<integer>* is

- 0 to turn off the `layer2` attribute; this results in the layer 3 discipline (default for HiperSockets).
- 1 to turn on the `layer2` attribute; this results in the layer 2 discipline (default for connections other than HiperSockets).

If the `layer2` attribute has a value of -1, the layer was not set. The default layer setting is used when the device is set online.

Results

If you configured the discipline successfully, more configuration attributes are shown (for example route4 for the layer 3 discipline) and can be configured. If an OSA device is not configured for a discipline but is set online, the device driver assumes that it is a layer 2 device. It then tries to load the layer 2 discipline.

For more information about layer2, see:

- *Open Systems Adapter-Express Customer's Guide and Reference*, SA22-7935
- *OSA-Express Implementation Guide*, SG24-5948
- *Networking Overview for Linux on zSeries*, REDP-3901
- *z/VM: Connectivity*, SC24-6267

Using priority queueing

An OSA-Express CHPID in QDIO mode has up to four output queues (queues 0 - 3). The priority queueing feature gives these queues different priorities (queue 0 having the highest priority). The four output queues are available only if multiple priority is enabled for queues on the OSA-Express CHPID in QDIO mode.

Before you begin

- Priority queueing applies to OSA-Express CHPIDs in QDIO mode only.
- If more than 160 TCP/IP stacks per OSA-Express CHPID are defined in the IOCDS, priority queueing is disabled.
- The device must be offline while you set the queueing options.

About this task

Queueing is relevant mainly to high-traffic situations. When there is little traffic, queueing has no impact on processing. The qeth device driver can put data on one or more of the queues. By default, the driver uses queue 2 for all data.

Procedure

You can determine how outgoing IP packages are assigned to queues by setting a value for the priority_queueing attribute of your qeth device.

Issue a command of the form:

```
# chzdev -a <device_bus_id> priority_queueing=<method>
```

or, using sysfs:

```
# echo <method> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/priority_queueing
```

where *<method>* can be any of these values:

prio_queueing_vlan
to base the queue assignment on the two most significant bits in the priority code point in the IEEE 802.1Q header as used in VLANs. This value affects only traffic with VLAN headers, and hence works only with qeth devices in layer 2 mode.

You can set the priority code point in the IEEE 802.1Q headers of the traffic based on `skb->priority` by using a command of the form:

```
ip link add link <link> name <name> type vlan id <vlan-id> egress-qos-map <mapping>
```

Note: Enabling this option makes all traffic default to queue 3.

prio_queueing_skb
to base the queue assignment on the priority flag of the skbs. An skb, or socket buffer, is a Linux kernel-internal structure that represents network data. The mapping to the priority queues is as follows:

Table 46. Mapping of flag value to priority queues

Priority flag of the skb	Priority queue
0-1	3
2-3	2
4-5	1
≥6	0

You can use `prio_queueing_skb` for any network setups, including conventional LANs.

Use either `sockopt SO_PRIORITY` or an appropriate **`iptables`** command to adjust the priority flag of the skb (`skb->priority`).

Note: The priority flag of the skbs defaults to 0, hence enabling this option makes all traffic default to queue 3.

prio_queueing_prec
to base the queue assignment on the two most significant bits of each packet's IP header precedence field.

prio_queueing_tos
Deprecated; do not use for new setups.

no_prio_queueing
causes the qeth device driver to use queue 2 for all packets. This value is the default.

no_prio_queueing:0
causes the qeth device driver to use queue 0 for all packets.

no_prio_queueing:1
causes the qeth device driver to use queue 1 for all packets.

no_prio_queueing:2
causes the qeth device driver to use queue 2 for all packets. This value is equivalent to the default.

no_prio_queueing:3
causes the qeth device driver to use queue 3 for all packets.

Example

To read the current value of priority queueing for device 0.0.a110, issue:

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a110/priority_queueing
```

Possible results are:

by VLAN headers
if `prio_queueing_vlan` is set.

by skb-priority
if `prio_queueing_skb` is set.

by precedence
if `prio_queueing_prec` is set.

by type of service
if `prio_queuing_tos` is set.

always queue *<x>*
otherwise.

To configure queueing by `skb->priority` setting for device 0.0.a110, issue:

```
# chzdev -a 0.0.a110 priority_queueing=prio_queueing_skb
```

Alternatively, using sysfs:

```
# echo prio_queueing_skb > /sys/bus/ccwgroup/drivers/qeth/0.0.a110/priority_queueing
```

Specifying the number of inbound buffers

Depending on the amount of available storage and the amount of traffic, you can assign 8 - 128 inbound buffers for each qeth group device.

Before you begin

The device must be offline while you specify the number of inbound buffers.

About this task

By default, the qeth device driver assigns 64 inbound buffers to OSA devices and 128 to HiperSockets devices.

The Linux memory usage for inbound data buffers for the devices is `(number of buffers) × (buffer size)`.

The buffer size is equivalent to the frame size, which depends on the type of CHPID:

- For an OSA-Express CHPID in QDIO mode: 64 KB
- For HiperSockets: depending on the HiperSockets CHPID definition, 16 KB, 24 KB, 40 KB, or 64 KB

Procedure

Set the buffer_count attribute to the number of inbound buffers you want to assign.
Issue a command of the form:

```
# chzdev <device_type> <device_bus_id> buffer_count=<number>
```

or, using sysfs:

```
# echo <number> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/buffer_count
```

Example

In this example, an operational device 0.0.a000 is first set offline, 64 inbound buffers are assigned to the device, and then the device is set back online:

```
# chzdev -d -a 0.0.a000
# chzdev 0.0.a000 buffer_count=64
# chzdev -e -a 0.0.a000
```

or, using sysfs:

```
# echo 0 > /sys/bus/ccwgroup/drivers/qeth/0.0.a000/online
# echo 64 > /sys/bus/ccwgroup/drivers/qeth/0.0.a000/buffer_count
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a000/online
```

Specifying the relative port number

Use the portno sysfs attribute to specify the relative port number.

Before you begin

- This description applies to adapters that, per CHPID, show more than one port to Linux.
- The device must be offline while you specify the relative port number.

Procedure

By default, the qeth group device uses port 0.
To use a different port, issue a command of the form:

```
# chzdev -a <device_bus_id> portno=<integer>
```

or, using sysfs:

```
# echo <integer> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/portno
```

Where *<integer>* is either 0 or 1.

Example

In this example, port 1 is assigned to the qeth group device.

```
# chzdev -a 0.0.a000 portno=1
```

or, using sysfs:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a000/portno
```

Finding out the type of your network adapter

Use the card_type attribute to find out the type of the network adapter through which your device is connected.

Procedure

You can find out the type of the network adapter through which your device is connected. To find out the type, read the device's card_type attribute.

To list all attributes, issue a command of the form:

```
# lszdev -a <device_type> <device_bus_id> --info --info
```

or, using sysfs:

```
# cat /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/card_type
```

The card_type attribute gives information about both the type of network adapter and the type of network link (if applicable) available at the card's ports. See Table 47 on page 265 for details.

Table 47. Possible values of card_type and what they mean

Value of card_type	Adapter type	Link type
OSD_25GIG	OSA card in OSD mode	25 Gigabit Ethernet
OSD_10GIG		10 Gigabit Ethernet
OSD_1000		Gigabit Ethernet, 1000BASE-T
OSD_GbE_LANE		Gigabit Ethernet, LAN Emulation
OSD_FE_LANE		Fast Ethernet, LAN Emulation
OSD_Express		Unknown
OSX	OSA-Express for zBX	10 Gigabit Ethernet
HiperSockets	HiperSockets, CHPID type IQD	N/A
Virtual NIC QDIO	VSWITCH or guest LAN based on OSA	N/A
Virtual NIC Hiper	Guest LAN based on HiperSockets	N/A
Unknown	Other	

Example

To find the card_type of a device 0.0.a100 issue:

```
# lszdev -a qeth 0.0.a100 --info --info
...
 READONLY    ACTIVE
 card_type   "OSD_1000"
....
```

or, using sysfs:

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a100/card_type
OSD_1000
```

Setting a device online or offline

Use the online device group attribute to set a device online or offline.

Procedure

To set a qeth group device online, set the online device group attribute to 1. To set a qeth group device offline, set the online device group attribute to 0.

Issue a command of the form:

```
# chzdev -a <device_bus_id> online=<flag>
```

or, using sysfs:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/online
```

Setting a device online associates it with an interface name (see “Finding out the interface name of a qeth group device” on page 266).

Setting a device offline closes this network device. If IPv6 is active, you lose any IPv6 addresses set for this device. After you set the device online, you can restore lost IPv6 addresses only by issuing the **ip** or **ifconfig** commands again.

Example

To set a qeth device with bus ID 0.0.a100 online issue:

```
# chzdev -a 0.0.a100 online=1
```

or, using sysfs:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a100/online
```

To set the same device offline issue:

```
# chzdev -a 0.0.a100 online=0
```

or, using sysfs:

```
# echo 0 > /sys/bus/ccwgroup/drivers/qeth/0.0.a100/online
```

Finding out the interface name of a qeth group device

When a qeth group device is set online, an interface name is assigned to it.

Procedure

To find the interface name of a qeth group device, either:

- Obtain a list of all attributes for a device by issuing the **lszdev** command for the device.

  Issue a command of the form:

  ```
  # lszdev -a qeth <device_bus_id> --info --info
  ```

- Obtain a mapping for all qeth interfaces and devices by issuing the **lsqeth -p** command.
- Find out the interface name of a qeth group device for which you know the device bus-ID by reading the group device's if_name attribute.

  Issue a command of the form:

  ```
  # cat /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/if_name
  ```

Example

```
# lszdev -a qeth 0.0.a100 --info --info
 ...
if_name   "enca100"
....
```

or, using sysfs:

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a100/if_name
enca100
```

Finding out the bus ID of a qeth interface

Use the **lsqeth -p** command to obtain a mapping for all qeth interfaces and devices. Alternatively, you can use sysfs.

Procedure

To find the device bus-ID that corresponds to an interface, either:

- Use the **lsqeth -p** command.
- Use the readlink command.

  For each network interface, there is a directory in sysfs under `/sys/class/net/`, for example, `/sys/class/net/encf500` for interface encf500. This directory contains a symbolic link "device" to the corresponding device in `/sys/devices`. Read this link to find the device bus-ID of the device that corresponds to the interface.

Example

To find out which device bus-ID corresponds to an interface enca100 issue, for example:

```
# readlink /sys/class/net/enca100/device
../../../0.0.a100
```

In this example, enca100 corresponds to the device bus-ID 0.0.a100.

Activating an interface

Use the **ip** command or equivalent to activate an interface.

Before you begin

- You must know the interface name of the qeth group device (see “Finding out the interface name of a qeth group device” on page 266).
- You must know the IP address that you want to assign to the device.

About this task

The MTU size defaults to the correct settings for HiperSockets devices. For OSA-Express CHPIDs in QDIO mode, the default MTU size depends on the device mode, layer 2 or layer 3.

- For layer 2, the default MTU is 1500 bytes.
- For layer 3, the default MTU is 1492 bytes.

In most cases, the default MTU sizes are well suited for OSA-Express CHPIDs in QDIO mode. If your network is laid out for jumbo frames, increase the MTU size to a maximum of 9000 bytes for layer 2, or to 8992 bytes for layer 3. Exceeding the defaults for regular frames or the maximum frame sizes for jumbo frames might cause performance degradation. See *Open Systems Adapter-Express Customer's Guide and Reference*, SA22-7935 for more details about MTU size.

For HiperSockets, the maximum MTU size is restricted by the maximum frame size as announced by the Licensed Internal Code (LIC). The maximum MTU is equal to the frame size minus 8 KB. Hence, the possible frame sizes of 16 KB, 24 KB, 40 KB, or 64 KB result in maximum corresponding MTU sizes of 8 KB, 16 KB, 32 KB, or 56 KB.

The MTU size defaults to the correct settings for both HiperSockets and OSA-Express CHPIDs in QDIO mode. As a result, you do not need to specify the MTU size when you activate the interface.

Procedure

You activate or deactivate network devices with **ip** or an equivalent command. For details of the **ip** command, see the **ip** man page.

Examples

- This example activates a HiperSockets CHPID with broadcast address 192.168.100.255:

```
# ip addr add 192.168.100.10/24 dev enca1c0
# ip link set dev enca1c0 up
```

- This example activates an OSA-Express CHPID in QDIO mode with broadcast address 192.168.100.255:

```
# ip addr add 192.168.100.11/24 dev encf500
# ip link set dev encf500 up
```

- This example reactivates an interface that was already activated and subsequently deactivated:

```
# ip link set dev encf500 up
```

Confirming that an IP address has been set under layer 3

There may be circumstances that prevent an IP address from being set, most commonly if another system in the network has set that IP address already.

About this task

The Linux network stack design does not allow feedback about IP address changes. If **ip** or an equivalent command fails to set an IP address on an OSA-Express network CHPID, a query with **ip** shows the address as being set on the interface although the address is not actually set on the CHPID.

There are usually failure messages about not being able to set the IP address or duplicate IP addresses in the kernel messages. You can find these messages in the output of the **dmesg** command. In Red Hat Enterprise Linux 9.1, you can also find the messages if you issue **journalctl**.

If you are not sure whether an IP address was set properly or experience a networking problem, check the messages or logs to see if an error was encountered when setting the address. This also applies in the context of HiperSockets and to both IPv4 and IPv6 addresses. It also applies to whether an IP address has been set for IP takeover, for VIPA, or for proxy ARP.

Duplicate IP addresses

The OSA-Express adapter in QDIO mode recognizes duplicate IP addresses on the same OSA-Express adapter or in the network using ARP and prevents duplicates.

About this task

Several setups require duplicate addresses:

- To perform IP takeover you need to be able to set the IP address to be taken over. This address exists prior to the takeover. See “Taking over IP addresses” on page 279 for details.

- For proxy ARP you need to register an IP address for ARP that belongs to another Linux instance. See “Configuring a device for proxy ARP” on page 282 for details.
- For VIPA you need to assign the same virtual IP address to multiple devices. See “Configuring a device for virtual IP address (VIPA)” on page 283 for details.

You can use the **qethconf** command (see “qethconf - Configure qeth devices” on page 708) to maintain a list of IP addresses that your device can take over, a list of IP addresses for which your device can handle ARP, and a list of IP addresses that can be used as virtual IP addresses, regardless of any duplicates on the same OSA-Express adapter or in the LAN.

Deactivating an interface

You can deactivate an interface with **ip** or an equivalent command or by setting the network device offline.

About this task

Setting a device offline involves actions on the attached device, but deactivating an interface only stops the interface logically within Linux.

Procedure

To deactivate an interface with **ip**, issue a command of the form:

```
# ip link set dev <interface_name> down
```

Example

To deactivate encf500 issue:

```
# ip link set dev encf500 down
```

Recovering a device

You can use the recover attribute of a qeth group device to recover it in case of failure.

About this task

issue **journalctl** to find messages that might inform you of a malfunctioning device.

Procedure

Issue a command of the form:

```
# chzdev -a <device_bus_id> -a recover=1
```

or, using sysfs:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/recover
```

Example

```
# chzdev 0.0.a100 -a recover=1
```

Alternatively, using sysfs:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a100/recover
```

Configuring hardware offload operations

Some CPU-intensive operations can be offloaded to the OSA adapter, thus reducing the load on the host CPU.

The qeth device driver supports offloading for the following operations on both layer 2 and layer 3:

- Inbound (receive) and outbound (transmit) checksum calculations for TCP and UDP network packets
- TCP segmentation, see "Enabling and disabling TCP segmentation offload" on page 271.

VLAN interfaces inherit offload settings from their base interface.

You can set the offload operations with the Linux **ethtool** command. See the **ethtool** man page for details. The following abbreviated example shows shows some offload settings:

```
# ethtool -k encf500
Features for encf500:
rx-checksumming: on
tx-checksumming: on
    tx-checksum-ipv4: on
    tx-checksum-ip-generic: off [fixed]
    tx-checksum-ipv6: on
    tx-checksum-fcoe-crc: off [fixed]
    tx-checksum-sctp: off [fixed]
scatter-gather: on
    tx-scatter-gather: on
    tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: on
    tx-tcp-segmentation: on
    tx-tcp-ecn-segmentation: off [fixed]
    tx-tcp6-segmentation: on
udp-fragmentation-offload: off [fixed]
generic-segmentation-offload: off [requested on]
generic-receive-offload: on
large-receive-offload: off [fixed]
...
```

Configuring the receive checksum offload feature

A checksum calculation is a form of redundancy check to protect the integrity of data.

Procedure

The qeth device driver supports offloading checksum calculations on inbound packets (receive) to the OSA feature.

To enable or disable checksum calculations by the OSA feature, issue a command of this form:

```
# ethtool -K <interface_name> rx <value>
```

where *<value>* is on or off.

Examples

- To let the OSA feature calculate the receive checksum for network device encf500, issue

```
# ethtool -K encf500 rx on
```

- To let the host CPU calculate the receive checksum for network device encf500, issue

```
# ethtool -K encf500 rx off
```

Configuring the transmit checksum offload feature

The qeth device driver supports offloading outbound (transmit) checksum calculations to the OSA feature.

About this task

You can enable or disable the OSA feature that calculates the transmit checksums by using the **ethtool** command.

Procedure

Issue a command of the form:

```
# ethtool -K <interface_name> tx <value>
```

where *<value>* is on or off.

Examples

- To let the OSA feature calculate the transmit checksum for network device encf500, issue

```
# ethtool -K encf500 tx on
```

- To let the host CPU calculate the transmit checksum for network device encf500, issue

```
# ethtool -K encf500 tx off
```

Enabling and disabling TCP segmentation offload

Offloading the TCP segmentation operation from the Linux network stack to the adapter can lead to enhanced performance for interfaces with predominately large outgoing packets.

About this task

TCP segmentation offload is supported for OSA connections on layer 3. On layer 2 it is available as of z14 for OSA Express6S and newer adapters. Use the **ethtool -k** (see example in "Configuring hardware offload operations" on page 270) to check whether your system supports it.

Procedure

Outbound (TX) checksumming and scatter gather are prerequisites for TCP segmentation offload (TSO). You must turn on scatter gather and outbound checksumming before configuring TSO.

All three options can be turned on or off with a single **ethtool** command of the form:

```
# ethtool -K <interface_name> tx <value> sg <value> tso <value>
```

where *<value>* is either on or off. For more information about TX checksumming, see "Configuring the transmit checksum offload feature" on page 270.

Attention: When TCP segmentation is offloaded, the OSA feature performs the calculations. Offloaded calculations are supported only for packets that go out to the LAN.

Examples

- To offload the TCP segmentation operation for a network device encf500 issue:

```
# ethtool -K encf500 tx on sg on tso on
```

- To disable offloading the TCP segmentation operation for a network device encf500 issue:

```
# ethtool -K encf500 tx off sg off tso off
```

Isolating data connections

You can restrict communications between operating system instances that share an OSA port on an OSA adapter.

About this task

A Linux instance can configure the OSA adapter to prevent any direct package exchange between itself and other operating system instances that share an OSA adapter. This configuration ensures a higher degree of isolation than VLANs.

QDIO data connection isolation is configured as a policy. The policy is implemented as a sysfs attribute called `isolation`. The attribute appears in sysfs regardless of whether the hardware supports the feature. The policy can take the following values:

none
No isolation. This is the default.

drop
Specifies the ISOLATION_DROP policy. All packets from guests sharing the same OSA adapter to the guest having this policy configured are dropped automatically. The same holds for all packets sent by the guest having this policy configured to guests on the same OSA card. All packets to or from the isolated guest need to have a target that is not hosted on the OSA card. You can accomplish this by a router hosted on a separate machine or a separate OSA adapter.

For example, assume that three Linux instances share an OSA adapter, but only one instance (Linux A) needs to be isolated. Then Linux A declares its OSA adapter (QDIO Data Connection to the OSA adapter) to be isolated. Any packet being sent to or from Linux A must pass at least the physical switch to which the shared OSA adapter is connected. Linux A cannot communicate with other instances that share the OSA adapter, here B or C. The two other instances could still communicate directly through the OSA adapter without the external switch in the network path (see Figure 64 on page 272).



Figure 64. Linux instance A is isolated from instances B and C

forward
Specifies the ISOLATION_FORWARD policy. All packets are passed through a switch. The ISOLATION_FORWARD policy requires a network adapter in Virtual Ethernet Port Aggregator (VEPA) mode with an adjacent switch port configured for reflective relay mode.

To check whether the switch of the adapter is in reflective relay mode, read the sysfs attribute `switch_attrs`. The attribute lists all supported forwarding modes, with the currently active mode enclosed in square brackets. For example:

```
lszdev -a qeth 0.0.f5f0 --info --info
...
READONLY         ACTIVE
...
switch_attrs:    "802.1 [rr]"
```

Or, using sysfs to query the attribute directly:

```
cat /sys/devices/qeth/0.0.f5f0/switch_attrs
802.1 [rr]
```

The example indicates that the adapter supports both 802.1 forwarding mode and reflective relay mode, and reflective relay mode is active.

Using a network adapter in VEPA mode achieves further isolation. VEPA mode forces traffic from the Linux guests to be handled by the external switch. For example, Figure 65 on page 273 shows instances A and B with ISOLATION_FORWARD specified for the policy. All traffic between A and B goes through the external switch. The rule set of the switch now determines which connections are possible. The graphic assumes that A can communicate with B, but not with C.



Figure 65. Traffic from Linux instance A and B is forced through an external switch

If the ISOLATION_FORWARD policy was enforced successfully, but the switch port later loses the reflective-relay capability, the device is set offline to prevent damage.

You can configure the policy regardless of whether the device is online. If the device is online, the policy is configured immediately. If the device is offline, the policy is configured when the device comes online.

Examples

- To check the current isolation policy:

```
# cat /sys/devices/qeth/0.0.f5f0/isolation
```

- To set the isolation policy to ISOLATION_DROP:

```
# chzdev -a qeth 0.0.f5f0 isolation=drop
```

Or, using sysfs:

```
# echo "drop" > /sys/devices/qeth/0.0.f5f0/isolation
```

- To set the isolation policy to ISOLATION_FORWARD:

```
# chzdev -a qeth 0.0.f5f0 isolation=forward
```

Or, using sysfs:

```
# echo "forward" > /sys/devices/qeth/0.0.f5f0/isolation
```

If the switch is not capable of VEPA support, or VEPA support is not configured on the switch, then you cannot set the isolation attribute value to 'forward' while the device is online. If the switch does not support VEPA and you set the isolation value 'forward' while the device is offline, then the device cannot be set online until the isolation value is set back to 'drop' or 'none'.

- To set the isolation policy to none:

```
# chzdev -a qeth 0.0.f5f0 isolation=none
```

Or, using sysfs:

```
# echo "none" > /sys/devices/qeth/0.0.f5f0/isolation
```

When using vNICs, VEPA mode needs to be enabled on the respective VSWITCH. See *z/VM: Connectivity*, SC24-6267 for information about setting up data connection isolation on a VSWITCH.

Displaying and resetting QETH performance statistics

Use the **ethtool** to display the QETH performance statistics and the `performance_stats` sysfs attribute to reset the statistic values.

About this task

Red Hat Enterprise Linux 9.1 continuously gathers QETH performance data.

Procedure

1. Use the **ethtool** command to display the statistics. For details, see the **ethtool** man page.
2. Optional: Reset the statistic values to 0 by writing 1 to the `performance_stats` sysfs attribute of the QETH device.
   For example:

```
# chzdev -a <device_bus_id> performance_stats=1
```

or, using sysfs:

```
echo 1 > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/performance_stats
```

Capturing a hardware trace

Hardware traces are intended for use by the IBM Support organization. Hardware tracing is turned off by default. Turn on the hardware-tracing feature only when instructed to do so by IBM Support.

Before you begin

- The OSA-Express adapter must support the hardware-tracing feature.
- The qeth device must be online to return valid values of the `hw_trap` attribute.

About this task

When errors occur on an OSA-Express adapter, both software and hardware traces must be collected. Instructions for software traces depend on a case-by-case basis and are communicated as part of the service process. The hardware-tracing feature requests a hardware trace if an error is detected. This feature makes it possible to correlate the hardware trace with the device driver trace. If the hardware-tracing feature is activated, traces are captured automatically, but you can also start the capturing yourself.

Procedure

To activate or deactivate the hardware-tracing feature, issue a command of the form:

```
# chzdev -a <device_bus_id> hw_trap=<value>
```

or, using sysfs:

```
# echo <value> > /sys/devices/qeth/<device_bus_id>/hw_trap
```

Where *<value>* can be:

arm
: If the hardware-tracing feature is supported, write `arm` to the `hw_trap` sysfs attribute to activate it. If the hardware-tracing feature is present and activated, the `hw_trap` sysfs attribute has the value `arm`.

disarm
: Write `disarm` to the `hw_trap` sysfs attribute to turn off the hardware-tracing feature. If the hardware-tracing feature is not present or is turned off, the `hw_trap` sysfs attribute has the value `disarm`. This setting is the default.

trap
: (Write only) Capture a hardware trace. Hardware traces are captured automatically, but if asked to do so by IBM Support, you can start the capturing yourself by writing `trap` to the `hw_trap` sysfs attribute. The hardware trap function must be set to arm.

Examples

In this example the hardware-tracing feature is activated and then started for qeth device 0.0.a000:

```
# chzdev -a 0.0.a000 hw_trap=arm
# chzdev 0.0.a000 -a hw_trap=trap
```

or, using sysfs:

```
# echo arm > /sys/devices/qeth/0.0.a000/hw_trap
# echo trap > /sys/devices/qeth/0.0.a000/hw_trap
```

Working with qeth devices in layer 3 mode

Tasks you can perform on qeth devices in layer 3 mode include setting up a router and taking over IP addresses. Use the layer 2 attribute to set the mode. See “Setting the layer2 attribute” on page 260 about setting the mode. See “Layer 2 and layer 3” on page 247 for general information about the layer 2 and layer 3 disciplines.

Setting up a Linux router in layer 3

By default, your Linux instance is not a router. Depending on your IP version, IPv4 or IPv6 you can use the route4 or route6 attribute of your qeth device to define it as a router.

Before you begin

- A suitable hardware setup must be in place that enables your Linux instance to act as a router.
- The Linux instance is set up as a router. To configure Linux running in a z/VM guest virtual machine or in an LPAR as a router, IP forwarding must be enabled in addition to setting the route4 or route6 attribute.

  For IPv4, enable IP forwarding by issuing:

```
# sysctl -w net.ipv4.conf.all.forwarding=1
```

  For IPv6, enable IP forwarding by issuing:

```
# sysctl -w net.ipv6.conf.all.forwarding=1
```

About this task

You can set the route4 or route6 attribute dynamically, while the qeth device is online.

The same values are possible for route4 and route6 but depend on the type of CHPID:

Table 48. Summary of router setup values

Router specification	OSA-Express CHPID in QDIO mode	HiperSockets CHPID
primary_router	Yes	No
secondary_router	Yes	No
primary_connector	No	Yes
secondary_connector	No	Yes
multicast_router	Yes	Yes
no_router	Yes	Yes

Both types of CHPIDs accept:

multicast_router
causes the qeth driver to receive all multicast packets of the CHPID. For a unicast function for HiperSockets see “HiperSockets Network Concentrator” on page 299.

no_router
is the default. You can use this value to reset a router setting to the default.

An OSA-Express CHPID in QDIO mode accepts the following values:

primary_router
to make your Linux instance the principal connection between two networks.

secondary_router
to make your Linux instance a backup connection between two networks.

A HiperSockets CHPID accepts the following values, if the microcode level supports the feature:

primary_connector
to make your Linux instance the principal connection between a HiperSockets network and an external network (see “HiperSockets Network Concentrator” on page 299).

secondary_connector
to make your Linux instance a backup connection between a HiperSockets network and an external network (see “HiperSockets Network Concentrator” on page 299).

Example

In this example, two Linux instances, "Linux P" and "Linux S", running on an IBM mainframe use OSA-Express to act as primary and secondary routers between two networks. IP forwarding must be enabled for Linux in an LPAR or as a z/VM guest to act as a router. In Red Hat Enterprise Linux 9.1 you can set IP forwarding permanently in `/etc/sysctl.conf` or dynamically with the sysctl command.

Mainframe configuration:



Figure 66. Mainframe configuration

It is assumed that both Linux instances are configured as routers in their LPARs or in z/VM.

Linux P configuration:

To create the qeth group devices:

```
# chzdev --active --enable qeth 0.0.0400,0.0.0401,0.0.0402
# chzdev --active --enable qeth 0.0.0200,0.0.0201,0.0.0202
```

Alternatively, use the sysfs attribute `group`:

```
# echo 0.0.0400,0.0.0401,0.0.0402 > /sys/bus/ccwgroup/drivers/qeth/group
# echo 0.0.0200,0.0.0201,0.0.0202 > /sys/bus/ccwgroup/drivers/qeth/group
```

To make Linux P a primary router for IPv4:

```
# chzdev -a qeth 0.0.0400 route4=primary_router
# chzdev -a qeth 0.0.0200 route4=primary_router
```

or, using sysfs:

```
# echo primary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0400/route4
# echo primary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0200/route4
```

Linux S configuration:

To create the qeth group devices:

```
# chzdev --active --enable qeth 0.0.0404,0.0.0405,0.0.0406
# chzdev --active --enable qeth 0.0.0204,0.0.0205,0.0.0206
```

or, using sysfs:

```
# echo 0.0.0404,0.0.0405,0.0.0406 > /sys/bus/ccwgroup/drivers/qeth/group
# echo 0.0.0204,0.0.0205,0.0.0206 > /sys/bus/ccwgroup/drivers/qeth/group
```

To make Linux S a secondary router for IPv4:

```
# chzdev -a qeth 0.0.0400 route4=secondary_router
# chzdev -a qeth 0.0.0200 route4=secondary_router
```

or, using sysfs:

```
# echo secondary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0404/route4
# echo secondary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0204/route4
```

In this example, qeth device 0.01510 is defined as a primary router for IPv6:

```
# chzdev --enable qeth 0.0.1510,0.0.1511,0.0.1512
# chzdev qeth 0.0.1510 route6=primary_router
# lszdev 1510 -i | grep route6
primary router
```

Alternatively, using sysfs attributes:

```
/sys/bus/ccwgroup/drivers/qeth # cd 0.0.1510
# echo 1 > online
# echo primary_router > route6
# cat route6
primary router
```

See “HiperSockets Network Concentrator” on page 299 for further examples.

Faking broadcast capability

It is possible to fake the broadcast capability for devices that do not support broadcasting.

Before you begin

- You can fake the broadcast capability only on devices that do not support broadcast.
- The device must be offline while you enable faking broadcasts.

About this task

For devices that support broadcast, the broadcast capability is enabled automatically.

To find out whether a device supports broadcasting, use the **ip** command. If the resulting list shows the BROADCAST flag, the device supports broadcast. This example shows that the device encf500 supports broadcast:

```
# ip -s link show dev encf500
3: encf500: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1492 qdisc pfifo_fast qlen 1000
    link/ether 00:11:25:bd:da:66 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    236350     2974     0       0       0       9
    TX: bytes  packets  errors  dropped carrier collsns
    374443     1791     0       0       0       0
```

Some processes, for example, the *gated* routing daemon, require the devices' broadcast capable flag to be set in the Linux network stack.

Procedure

To set the broadcast capable flag for devices that do not support broadcast, set the fake_broadcast attribute of the qeth group device to 1. To reset the flag, set it to 0.

Issue a command of the form:

```
# chzdev -a <device_bus_id> fake_broadcast=<flag>
```

or, using sysfs:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/fake_broadcast
```

Example

In this example, a device 0.0.a100 is instructed to pretend that it can broadcast.

```
# chzdev -a 0.0.a100 fake_broadcast=1
```

or, using sysfs:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a100/fake_broadcast
```

Taking over IP addresses

You can configure IP takeover if the layer2 option is not enabled. If you enabled the layer2 option, you can configure for IP takeover as you would in a distributed server environment.

About this task

For information about the layer2 option, see “MAC headers in layer 2 mode” on page 249.

Taking over an IP address overrides any previous allocation of this address to another LPAR. If another LPAR on the same CHPID already registered for that IP address, this association is removed.

An OSA-Express CHPID in QDIO mode can take over IP addresses from any IBM Z operating system. IP takeover for HiperSockets CHPIDs is restricted to taking over addresses from other Linux instances in the same Central Electronics Complex (CEC).

IP address takeover between multiple CHPIDs requires ARP for IPv4 and Neighbor Discovery for IPv6. OSA-Express handles ARP transparently, but not Neighbor Discovery.

There are three stages to taking over an IP address:

Stage 1: Ensure that your qeth group device is enabled for IP takeover
Stage 2: Activate the address to be taken over for IP takeover
Stage 3: Issue a command to take over the address

Stage 1: Enabling a qeth group device for IP takeover

For OSA-Express and HiperSockets CHPIDs, both the qeth group device that is to take over an IP address and the device that surrenders the address must be enabled for IP takeover.

Procedure

By default, qeth devices are not enabled for IP takeover. To enable a qeth group device for IP address takeover set the enable device group attribute to 1. To switch off the takeover capability set the enable device group attribute to 0.

In sysfs, the enable attribute is located in a subdirectory `ipa_takeover`. Issue a command of the form:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/ipa_takeover/enable
```

Example

In this example, a device 0.0.a500 is enabled for IP takeover:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a500/ipa_takeover/enable
```

Stage 2: Activating and deactivating IP addresses for takeover

The qeth device driver maintains a list of IP addresses that qeth group devices can take over or surrender. To enable Linux to take over an IP-address or to surrender an address, the address must be added to this list.

Procedure

Use the **qethconf** command to add IP addresses to the list.

- To display the list of IP addresses that are activated for IP takeover issue:

```
# qethconf ipa list
```

- To activate an IP address for IP takeover, add it to the list.
  Issue a command of the form:

```
# qethconf ipa add <ip_address>/<mask_bits> <interface_name>
```

- To deactivate an IP address delete it from the list.
  Issue a command of the form:

```
# qethconf ipa del <ip_address>/<mask_bits> <interface_name>
```

  In these commands, *<ip_address>/<mask_bits>* is the range of IP addresses to be activated or deactivated. See “qethconf - Configure qeth devices” on page 708 for more details about the **qethconf** command.

IPv4 example

In this example, there is only one range of IP addresses (192.168.10.0 to 192.168.10.255) that can be taken over by HiperSockets device enca1c0.

List the range of IP addresses (192.168.10.0 to 192.168.10.255) that can be taken over by HiperSockets device enca1c0.

```
# qethconf ipa list
ipa add 192.168.10.0/24 enca1c0
```

The following command adds a range of IP addresses that can be taken over by OSA device encf500.

```
# qethconf ipa add 192.168.11.0/24 encf500
qethconf: Added 192.168.11.0/24 to /sys/class/net/encf500/device/ipa_takeover/add4.
qethconf: For verification please use "qethconf ipa list"
```

Listing the activated IP addresses now shows both ranges of addresses.

```
# qethconf ipa list
ipa add 192.168.10.0/24 enca1c0
ipa add 192.168.11.0/24 encf500
```

The following command deletes the range of IP addresses that can be taken over by OSA device encf500.

```
# qethconf ipa del 192.168.11.0/24 encf500
qethconf: Deleted 192.168.11.0/24 from
          sysfs entry /sys/class/net/encf500/device/ipa_takeover/del4.
qethconf: For verification please use "qethconf ipa list"
```

IPv6 example

The following command adds one range of IPv6 addresses, fec0:0000:0000:0000:0000:0000:0000:0000 to fec0:0000:0000:0000:FFFF:FFFF:FFFF:FFFF, that can be taken over by device encd300.
Add a range of IP addresses:

```
qethconf ipa add fec0::/64 encd300
qethconf: Added fec0:0000:0000:0000:0000:0000:0000:0000/64 to
          sysfs entry /sys/class/net/encd300/device/ipa_takeover/add6.
qethconf: For verification please use "qethconf ipa list"
```

Listing the activated IP addresses now shows the range of addresses:

```
qethconf ipa list
...
ipa add fec0:0000:0000:0000:0000:0000:0000:0000/64 encd300
```

The following command deletes the IPv6 address range that can be taken over by encd300:

```
qethconf ipa del fec0:0000:0000:0000:0000:0000:0000:0000/64 encd300:
qethconf: Deleted fec0:0000:0000:0000:0000:0000:0000:0000/64 from
          sysfs entry /sys/class/net/encd300/device/ipa_takeover/del6.
qethconf: For verification please use "qethconf ipa list"
```

Stage 3: Issuing a command to take over the address

To complete taking over a specific IP address and remove it from the CHPID or LPAR that previously held it, issue an **ip addr** or equivalent command.

Before you begin

- Both the device that is to take over the IP address and the device that is to surrender the IP address must be enabled for IP takeover. This rule applies to the devices on both OSA-Express and HiperSockets CHPIDs. (See “Stage 1: Enabling a qeth group device for IP takeover” on page 279).
- The IP address to be taken over must have been activated for IP takeover (see “Stage 2: Activating and deactivating IP addresses for takeover” on page 280).

About this task

Be aware of the information in “Confirming that an IP address has been set under layer 3” on page 268 when using IP takeover.

Examples

IPv4 example:

To make a HiperSockets device enca1c0 take over IP address 192.168.10.22 issue:

```
# ip addr add 192.168.10.22/24 dev enca1c0
```

For IPv4, the IP address you are taking over must be different from the one that is already set for your device. If your device already has the IP address it is to take over, you must issue two commands: First remove the address to be taken over if it is already there. Then add the IP address to be taken over.

For example, to make a HiperSockets device enca1c0 take over IP address 192.168.10.22 if enca1c0 is already configured to have IP address 192.168.10.22 issue:

```
# ip addr del 192.168.10.22/24 dev enca1c0
# ip addr add 192.168.10.22/24 dev enca1c0
```

IPv6 example:

To make a device encd300 take over fec0::111:25ff:febd:d9da/64 issue:

```
ip addr add fec0::111:25ff:febd:d9da/64 nodad dev encd300
```

For IPv6, setting the **nodad** (no duplicate address detection) option ensures that the encd300 interface uses the IP address fec0::111:25ff:febd:d9da/64. Without the **nodad** option, the previous owner of the IP address might prevent the takeover by responding to a duplicate address detection test.

The IP address you are taking over must be different from the one that is already set for your device. If your device already has the IP address it is to take over you must issue two commands: First remove the address to be taken over if it is already there. Then add the IP address to be taken over.

For example, to make a device encd300 take over IP address fec0::111:25ff:febd:d9da/64 when encd300 is already configured to have that particular IP address issue:

```
ip addr del fec0::111:25ff:febd:d9da/64 nodad dev encd300
ip addr add fec0::111:25ff:febd:d9da/64 nodad dev encd300
```

Configuring a device for proxy ARP

You can configure a device for proxy ARP if the layer2 option is not enabled. If you enabled the layer2 option, you can configure for proxy ARP as you would in a distributed server environment.

Before you begin

Configure only qeth group devices that are set up as routers for proxy ARP.

About this task

For information about the layer2 option, see "MAC headers in layer 2 mode" on page 249.

The qeth device driver maintains a list of IP addresses for which a qeth group device handles ARP and issues gratuitous ARP packets. For more information about proxy ARP, see

```
http://www.cisco.com/c/en/us/support/docs/ip/dynamic-address-allocation-resolution/13718-5.html
```

Use the **qethconf** command to display this list or to change the list by adding and removing IP addresses (see "qethconf - Configure qeth devices" on page 708).

Be aware of the information in "Confirming that an IP address has been set under layer 3" on page 268 when you work with proxy ARP.

Example

Figure 67 on page 282 shows an environment where proxy ARP is used.



Figure 67. Example of proxy ARP usage

G1, G2, and G3 are instances of Linux on z/VM (connected, for example, through a guest LAN to a Linux router R), reached from GW (or the outside world) through R. R is the ARP proxy for G1, G2, and G3. That is, R agrees to take care of packets that are destined for G1, G2, and G3. The advantage of using proxy ARP is that GW does not need to know that G1, G2, and G3 are behind a router.

To receive packets for 1.2.3.4, so that it can forward them to G1 1.2.3.4, R would add 1.2.3.4 to its list of IP addresses for proxy ARP for the interface that connects it to the OSA adapter.

```
# qethconf parp add 1.2.3.4 encf500
qethconf: Added 1.2.3.4 to /sys/class/net/encf500/device/rxip/add4.
qethconf: Use "qethconf parp list" to check for the result
```

After issuing similar commands for the IP addresses 1.2.3.5 and 1.2.3.6 the proxy ARP configuration of R would be:

```
# qethconf parp list
parp add 1.2.3.4 encf500
parp add 1.2.3.5 encf500
parp add 1.2.3.6 encf500
```

Configuring a device for virtual IP address (VIPA)

You can configure a device for VIPA if the layer2 option is not enabled. If you enabled the layer2 option, you can configure for VIPA as you would in a distributed server environment.

About this task

For information about the layer2 option, see “MAC headers in layer 2 mode” on page 249.

IBM Z uses VIPAs to protect against certain types of hardware connection failure. You can assign VIPAs that are independent from particular adapter. VIPAs can be built under Linux using *dummy* devices (for example, "dummy0" or "dummy1").

The qeth device driver maintains a list of VIPAs that the OSA-Express adapter accepts for each qeth group device. Use the **qethconf** utility to add or remove VIPAs (see “qethconf - Configure qeth devices” on page 708).

For an example of how to use VIPA, see “Scenario: VIPA – minimize outage due to adapter failure” on page 294.

Be aware of “Confirming that an IP address has been set under layer 3” on page 268 when you work with VIPAs.

Configuring a HiperSockets device for AF_IUCV addressing

Use the `hsuid` attribute of a HiperSockets device in layer 3 mode to identify it to the AF_IUCV addressing family support.

Before you begin

- Support for AF_IUCV based connections through real HiperSockets requires Completion Queue Support.
- The device must be set up for AF_IUCV addressing (see “Setting up HiperSockets devices for AF_IUCV addressing” on page 334).

Procedure

To set an identifier, issue a command of this form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/qeth/0.0.a007/hsuid
```

The identifier is case-sensitive and must adhere to these rules:

- It must be 1 - 8 characters.
- It must be unique across your environment.
- It must not match any z/VM user ID in your environment. The AF_IUCV addressing family support also supports z/VM IUCV connections.

Example

In this example, MYHOST01 is set as the identifier for a HiperSockets device with bus ID `0.0.a007`.

```
# echo MYHOST01 > /sys/bus/ccwgroup/drivers/qeth/0.0.a007/hsuid
```

Working with qeth devices in layer 2 mode

Tasks that you can perform on qeth devices in layer 2 mode include setting up a OSA or HiperSockets bridge port and tuning packet handling for a HiperSockets device with VNIC characteristics..

VNIC characteristics and the bridge port role are mutually exclusive.

Use the layer2 attribute to set the mode. See “Setting the layer2 attribute” on page 260 about setting the mode. See “Layer 2 and layer 3” on page 247 for general information about the layer 2 and layer 3 disciplines.

Configuring a network device as a member of a Linux bridge

You can define an OSA or HiperSockets device to be a bridge port, which allows it to act as a member of a Linux software bridge. Use the `bridge_role` attribute of a network device in layer 2 to make it receive all traffic with unknown destination MAC addresses.

Alternatively, use VNIC characteristics to configure a layer 2 network device to receive all unknown traffic (see “Advanced packet-handling configuration” on page 287).

Before you begin

To use the bridging support, you need OSA or HiperSockets hardware that supports layer 2 SETBRIDGEPORT functionality.

You can have one active bridge port per Internal Queued Direct Communication (IQD) or OSA channel. You can have either only secondary bridge ports, or one primary together with up to four secondary bridge ports per OSA or HiperSockets channel. If the primary, or currently active secondary, bridge port fails, one of the available secondary bridge ports takes over. For each secondary bridge port, set `bridge_role` to `secondary`.

If you configure your Linux instance to have a bridge port, consider other bridge ports that might be configured that share the channel. Bridge ports can be distributed, for example, over five LPARs with one bridge port each.

Devices for which VNIC characteristics are configured cannot also be configured as bridge ports.

On z13 and earlier mainframes: HiperSockets bridge ports only bridge traffic to and from HiperSockets ports in z/VM guests. On z14 and later HiperSockets bridge ports bridge traffic to and from all layer 2 HiperSockets ports that are not configured as "bridge_invisible", see “Advanced packet-handling configuration” on page 287.

HiperSockets only: On IQDX channels permission to configure ports as bridge ports must be granted in IBM zEnterprise Unified Resource Manager (zManager). On machines in PR/SM mode, bridge ports can only be configured on IQD channels that are defined as "external-bridged" in the IOCDS. On machines in DPM mode, bridge ports can be configured on any IQD channel.

For more information about the bridge port concept, see “Layer 2 promiscuous mode” on page 253.

About this task

The following sysfs attributes control the bridge port functions. The attributes can be found in the `/sys/bus/ccwgroup/drivers/qeth/<device_bus_id>` directory.

bridge_role
Read-write attribute that controls the role of the port. Valid values are:

primary
Assigns the port the primary bridge port role.

secondary
Assigns the port a secondary bridge port role.

none
Revokes existing bridge port roles and indicates that no role is assigned.

Assigning a role directly to a port prevents use of the **bridge_reflect_promisc** attribute.

bridge_state
Read-only attribute that shows the state of the port. Valid values are:

active
The port is assigned a bridge port role and is switched into active state by the adapter. The device receives frames that are addressed to unknown MAC addresses.

standby
The port is assigned a bridge port role, but is not currently switched into active state by the adapter. The device does not receive frames that are destined to unknown MAC addresses.

inactive
The port is not assigned a bridge port role.

bridge_hostnotify
HiperSockets only: Read-write attribute that controls the sending of notifications for the port. When you enable notifications (even if notifications were already enabled), udev events are emitted for all currently connected communication peers in quick succession. After that, a udev event is emitted every time a communication peer is connected, or a previously connected peer is disconnected. Any user space program that monitors these events must repopulate its list of registered peers every time the status of the bridge port device changes to enable notifications.

Valid values are:

1
The port is set to send notifications.

0
Notifications are turned off.

Notifications about the change of the state of bridge ports, and (if enabled) about registration and deregistration of communication peers on the LAN are delivered as udev events. The events are described in the file `Documentation/s390/qeth.txt` in the Linux kernel source tree.

bridge_reflect_promisc
Read-write attribute that, when set, makes the bridge-port role of the port follow ("reflect") the promiscuity flag (IFF_PROMISC) of the corresponding Linux network interface. You can specify the following values:

none
Setting and resetting the promiscuous mode on the network interface has no effect on the bridge-port role of the underlying port.

primary
Setting or resetting the promiscuous mode on the network interface that is served by this device causes the driver to attempt assigning (or resetting) the primary role to the port. If a port with the primary role exists, assignment fails.

secondary
Setting or resetting the promiscuous mode on the network interface that is served by this device causes the driver to attempt assigning (or resetting) the secondary role to the port.

Setting **bridge_reflect_promisc** to anything but **none** causes the **bridge_role** attribute to become read-only. The role of a port changes as a result of setting or unsetting the promiscuity flag (IFF_PROMISC) of the corresponding network interface. You can check the currently assigned role by reading the **bridge_role** attribute.

Procedure

1. To configure a network device as a bridge, issue a command of this form:

```
# chzdev -a <device_bus_id> bridge_role=<value>
```

or, using sysfs:

```
# echo <value> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/bridge_role
```

Setting the **bridge_role** attribute requires the **bridge_reflect_promisc** attribute to be **none**. Alternatively, to make the bridge-port role of the port follow the promiscuity flag (IFF_PROMISC) of the corresponding Linux network interface, issue a command of the following form:

```
# chzdev -a <device_bus_id> bridge_reflect_promisc=<value>
```

or, using sysfs:

```
# echo <value> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/bridge_reflect_promisc
```

where valid values are:

- primary
- secondary
- none

2. Check the state of the bridge port by reading the `bridge_state` attribute. Issue a command of this form:

```
# lszdev -a qeth <device_bus_id> --info --info
```

Alternatively, use the sysfs attribute `bridge_state` directly:

```
# cat /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/bridge_state
```

where displayed values could be:

- active
- standby
- inactive

Example

In this example, a network device with bus ID `0.0.a007` is defined as a primary bridge port.

```
# chzdev -a 0.0.a007 bridge_role=primary
```

Equivalently, using sysfs:

```
# echo primary > /sys/bus/ccwgroup/drivers/qeth/0.0.a007/bridge_role
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a007/bridge_state
active
```

Advanced packet-handling configuration

Use VNIC characteristics to control how OSA or HiperSockets devices in layer 2 mode handle special scenarios, for example, packets with unknown MAC addresses, address takeover, or traffic with bridge ports.

Before you begin

- See your IBM Z hardware documentation about support for VNIC characteristics. Support might differ for OSA and HiperSockets devices.
- VNIC characteristics are supported for layer 2 mode only.
- VNIC characteristics cannot be configured on devices that are configured as bridge ports.

About this task

You can configure and fine-tune a promiscuous mode for incoming packets. You can configure the device to receive all packets regardless of the MAC address, or you can reject incoming multicast packets, or broadcast packets, or both.

For Linux instances that host multiple guest operating systems with different MAC addresses, you can configure the device to learn and handle these MAC addresses. The device then provides functions similar to a switch or to a software bridge.

The VNIC characteristics also include settings that can protect the MAC address of the device from being taken over by another device. You can deny takeover, or you can explicitly permit takeover to facilitate migration, for example in a recovery situation.

The VNIC characteristics of a qeth device are represented by sysfs attributes in `/sys/devices/qeth/<device_bus_id>/vnicc`:

flooding
With flooding enabled, the device receives packets to any unknown destination MAC address. Valid values are `0` for disabled and `1` for enabled. By default, flooding is disabled.

For a shared OSA adapter, flooding applies to traffic between the physical port and the OSA interfaces, but not to communication between the interfaces that share the adapter. Enable `learning` to configure bridge-like behavior of shared OSA adapters.

An OSA Express adapter can support a maximum of 64 devices with flooding enabled.

mcast_flooding
With multicast flooding enabled, the device receives packets to any multicast MAC addresses. Valid values are `0` for disabled and `1` for enabled. By default, multicast flooding is disabled and the device receives only packets to multicast MAC addresses to which it has previously registered.

rx_bcast
With broadcast receiving enabled, the device receives packets with the broadcast destination MAC address. Valid values are `0` for disabled and `1` for enabled. By default, the device is enabled to receive broadcast packets.

learning
With learning enabled, the device assembles a list of source MAC addresses of outgoing packets. The device then receives incoming packets to any MAC address in the list. Valid values are `0` for disabled and `1` for enabled. By default, learning is disabled.

A MAC address is added to the list unless it has been explicitly assigned to another device on the same channel. An exception are addresses of devices on which the takeover_learning characteristic is set. Such devices surrender their address to a learning device. If an address is already listed by

a different learning device on the same channel, the address is removed from that learning device's list. Explicitly configuring a MAC address on a different device removes the address from the learning device list.

A learned MAC address is dropped from the list of learned MAC addresses unless packets with this MAC address are sent within a specific timeout period. The default timeout period is 600 s. You can specify a different timeout period with the `learning_timeout` attribute.

takeover_setvmac
With this option enabled, the device's MAC address can be configured on a different device, without notification. Valid values are `0` for disabled and `1` for enabled. By default, this option is disabled and the MAC address cannot be configured on a different device.

takeover_learning
With takeover by learning enabled, the MAC address of this device can be learned on a different device and, thus, taken over by this other device, without notification. Valid values are `0` for disabled and `1` for enabled. By default, takeover by learning is disabled.

bridge_invisible
With bridge-port invisible enabled, packets are not transferred between the device and any other device that is configured as a bridge port. Valid values are `0` for disabled and `1` for enabled. By default, this option is disabled and, thus, traffic to and from bridge ports is permitted.

This characteristic applies to HiperSockets devices only.

learning_timeout
With learning enabled, this attribute specifies a timeout period, in seconds. A MAC address is dropped from the list of learned MAC addresses if this timeout period expires without any packets with this MAC address being received.

You can set this timeout period by writing a value in the range 60 - 86400 to the attribute. The default is 600. The timeout must be set before learning is enabled on the device.

Procedure

1. Optional: To read a VNIC characteristic setting from sysfs, issue a command of this form:

```
# cat /sys/devices/qeth/<device_bus_id>/vnicc/<attribute>
```

where *<device_bus_id>* is the device-bus ID of the qeth device and *<attribute>* is one of the attributes that represent the VNIC characteristics.

Example:

```
# cat /sys/devices/qeth/0.0.a016/vnicc/learning
0
```

Tip: For an overview of all VNIC characteristics of the device, find the interface name of the device, then use the **lsqeth** command, or **lszdev -i** *<dev_bus_ID>*.

Example:

```
# cat /sys/devices/qeth/0.0.a016/if_name
enca160
# lsqeth enca160 | grep vnicc
        vnicc/bridge_invisible   : 0
        vnicc/flooding           : 0
        vnicc/learning           : 0
        vnicc/learning_timeout   : 600
        vnicc/mcast_flooding     : 1
        vnicc/rx_bcast           : 1
        vnicc/takeover_learning  : 0
        vnicc/takeover_setvmac   : 0
```

2. To set a VNIC characteristic issue a command of this form:

```
# chzdev <device_bus_id> vnicc/<attribute>=<value>
```

where *<device_bus_id>* is the device-bus ID of the qeth device, *<attribute>* is one of the attributes that represent the VNIC characteristics, and *<value>* is the value to be set.

This setting persists across re-boots. To apply this setting to the running system only, use the **chzdev** command with the `-a` option or use the corresponding sysfs attribute.

Example: In this example, learning is enabled for a device with bus-ID `0.0.a016`.

```
# chzdev 0.0.a016 vnicc/learning=1
```

or, using sysfs:

```
# echo 1 > /sys/devices/qeth/0.0.a016/vnicc/learning
```

Example

This example shows a typical configuration for a bridge-like behavior of the device.

```
# lsqeth enca160 | grep vnicc
        vnicc/bridge_invisible   : 0
        vnicc/flooding           : 1
        vnicc/learning           : 1
        vnicc/learning_timeout   : 600
        vnicc/mcast_flooding     : 1
        vnicc/rx_bcast           : 1
        vnicc/takeover_learning  : 1
        vnicc/takeover_setvmac   : 1
```

Working with HiperSockets Converged Interfaces

Using HiperSockets Converged Interface (HSCI) connections, a HiperSockets network interface can be combined with an external OSA- or RoCE port, thus creating a single network interface.

About this task

The HSCI function is available as of IBM z15 or IBM LinuxONE III.

With this function, you can connect an instance of Linux that runs in LPAR mode to z/OS through layer 2 HiperSockets. The z/OS version must support HSCI.

A converged network can span multiple IBM Z servers.

Example: Consolidating subnets

Between LPARs, you can connect Linux instances through HiperSockets. To connect to an external network, you need an OSA-Express adapter in QDIO mode, or a RoCE Express adapter.

To connect Linux and z/OS LPARs with each other and an external network, you can use OSA Express adapters, for example, as shown in Figure 68 on page 290. All traffic between the operating system instances go through the OSA adapters, which puts load on the OSA adapters, and might not perform as well as HiperSockets.



Figure 68. A network using OSA Express adapters that connects Linux and z/OS LPARS

You might add a no-charge HiperSockets for the internal communication, which allows for faster communication inside the hardware system, and reduces load on the OSA adapters.

The performance gain comes at the cost of managing twice the number of interfaces and a second IP subnet.

Communication networks at enterprise level can easily grow in complexity and become a burden for the network administrator, as shown in Figure 69 on page 290



Figure 69. A complex network with two subnets and two IP addresses for each operating system instance

With HSCI interfaces, you can create a converged network that includes both direct HiperSockets connections for traffic within the server hardware and external connectivity through OSA Express or RoCE Express adapters. The HSCI interface is managed as a single interface.

In the sample network, there is now only one subnet, one IP address per operating system instance, and HiperSockets is still used for fast internal communication. This setup is shown in Figure 70 on page 291.



Figure 70. A converged HSCI network with one subnet and one IP address for each operating system instance

All HiperSockets interfaces of the HiperSockets channel must participate in the HSCI network. A HiperSockets interface on its own cannot communicate correctly with its network neighbors.

Working with HSCI connections comprises the following tasks:

- "Creating an HSCI interface" on page 291
- "Using an HSCI interface as a base device for MacVTap or OpenVSwitch" on page 292

Creating an HSCI interface

Combine a HiperSockets network interface with an external OSA- or RoCE port to create a single network interface.

Before you begin

- It is useful to assign the participating adapters and HiperSockets channels to the same PNET ID in the IOCDS.

Procedure

1. Ensure that the HiperSockets interface and the OSA or RoCE interface that you want to work with are up.
2. Merge the HiperSockets interface and the OSA or RoCE interface by issuing a command of the form:

```
# hsci add <HipSock_if> <Ext_if>
```

   The resulting HSCI interface name is based on the device-bus ID of the HiperSockets interface.

   As a simple example, Figure 71 on page 292 illustrates how a Linux instance running in an LPAR is changed to use one HSCI interface instead of one OSA interface and one HiperSockets interface.



Figure 71. A Linux instance where interfaces are merged to a single HSCI interface

Assume you want to set up an HSCI interface by converging a HiperSockets interface named `encb112`, and an OSA-Express interface named `enca100`. Connect the two by issuing:

```
# hsci add encb112 enca100
...
Successfully added HSCI interface hscib112
```

In the example, the device-bus ID of 0.0.b112 results in HiperSockets interface encb112, and then in the HSCI interface name `hscib112`. The HSCI interface name is predictable.

For more information about the **hsci** command, see "hsci - Manage HSCI interfaces" on page 642.

3. Assign an IP address to the new HSCI interface, for example, with the **ip** command.

   Issue a command of the form:

   ```
   # ip addr add <IP_address> dev <interface>
   ```

   For example, assuming that the new HSCI interface is called `hscib112`:

   ```
   # ip addr add 10.1.0.1/16 dev hscib112
   ```

4. Optional: You can list the new HSCI interface with the **hsci show** command:

   ```
   # hsci show
   HSCI      PNET_ID  HiperSockets  External
   ------------------------------------------------------------
   hsci8410  NET1     enca100       hscib112
   ```

What to do next

You can delete an HSCI interface with the **hsci del** command, for example:

```
hsci del hscib112
Deleting HSCI interface hsci8410 with the HiperSockets enc8410 and the
external enca100
Deleting MAC fe:c2:f4:35:00:12 on enca100
Successfully deleted device hscib112
```

Using an HSCI interface as a base device for MacVTap or OpenVSwitch

You can use an HSCI network device as the base device for a MacVTap or OpenVSwitch connection. You can, for example, attach KVM virtual servers to the converged network.

Before you begin

It is useful to define the PNET ID for the HiperSockets channel and the OSA or RoCE adapters to mark them as part of the same network segment.

About this task

To attach KVM virtual servers to a converged network, you define the HSCI device as a source device in the domain XML of the virtual server.

The following example assumes that there is an HiperSockets interface enc8410, an OSA interface encb040, and you want to create an HSCI interface hsci8410. Then you can use the HSCI interface to set up a MacVTap connection with two KVM virtual servers.

Procedure

1. On the KVM host, define the HiperSockets interface as layer 2. Issue a command of the form:

```
# chzdev -e <device_ID> layer2=1
```

   For example, if the device ID of the HiperSockets device is 8410:

```
# chzdev -e 8410 layer2=1
QETH device 0.0.8410:0.0.8411:0.0.8412 configured
```

2. Define the OSA interface with flooding and mcast_flooding enabled. Issue a command of the form:

```
# chzdev -e <device_ID> vnicc/flooding=1 vnicc/mcast_flooding=1
```

   For example, if the OSA device ID is b040:

```
# chzdev -e b040 vnicc/flooding=1 vnicc/mcast_flooding=1
QETH device 0.0.b040:0.0.b041:0.0.b042 configured
   Adding layer2=1 to active configuration (required by vnicc/mcast_flooding)
   Adding layer2=1 to persistent configuration (required by vnicc/mcast_flooding)
```

3. Create the converged HSCI interface. Issue a command of the form:

```
# hsci add <HipSock_if> <OSA_if>
```

   For example, if the HiperSockets interface is enc8410 and the OSA interface is encb040:

```
# hsci add enc8410 encb040
Verifying net dev encb040 and HiperSockets dev enc8410
Adding hsci8410 with a HiperSockets dev enc8410 and an external dev encb040
Added HSCI interface hsci8410
```

4. Optional: Check that the HSCI interface was created. Use the **`hsci show`** command.
   For example:

```
hsci show
HSCI       PNET_ID        HiperSockets            External
-------------------------------------------------------------
hsci8410   NET1           enc8410                 encb040
```

What to do next

You can use the HSCI interface as the base for a MacVTap or an OpenVSwitch connection on a KVM virtual server, as illustrated in Figure 72 on page 294.



Figure 72. MacVTAp connection with two KVM virtual servers

See *KVM Virtual Server Management*, SC34-2752 for how to configure a network interface in the domain configuration XML of the KVM virtual servers.

Scenario: VIPA – minimize outage due to adapter failure

Using VIPA you can assign IP addresses that are not associated with a particular adapter. VIPA thus minimizes outage that is caused by adapter failure.

VIPA is a facility for assigning an IP address to a system, instead of to individual adapters. It is supported by the Linux kernel. The addresses can be in IPv4 or IPv6 format.

This scenario describes how to use standard VIPA. Standard VIPA is sufficient for applications, such as web servers, that do *not* open connections to other nodes.

Note: See the information in "Confirming that an IP address has been set under layer 3" on page 268 concerning possible failure when you set IP addresses for OSA-Express features in QDIO mode (qeth driver).

Setting up standard VIPA

To set up VIPA you must create a dummy device, ensure that your service listens to the IP address, and set up routing to it.

Procedure

Follow these main steps to set up VIPA in Linux:

1. Create a dummy device with a virtual IP address.
2. Ensure that your service (for example, the Apache web server) listens to the virtual IP address assigned in step "1" on page 294.
3. Set up routes to the virtual IP address, on clients or gateways. To do so, you can use either:
   - Static routing (shown in "Example of how to set up standard VIPA" on page 295).
   - Dynamic routing. For details of how to configure routes, you must see the documentation that is delivered with your routing daemon (for example, `zebra` or `gated`).

Adapter outage

If outage of an adapter occurs, you must switch adapters.

Procedure

- Under static routing:
  a) Delete the route that was set previously.

b) Create an alternative route to the virtual IP address.

- Under dynamic routing, see the documentation that is delivered with your routing daemon for details.

Example of how to set up standard VIPA

This example shows you how to configure VIPA under static routing, and how to switch adapters when an adapter outage occurs.

About this task

Figure 73 on page 295 shows the network adapter configuration that is used in the example.



Figure 73. Example of using Virtual IP Address (VIPA)

Procedure

1. Define the real interfaces.

```
[server]# ip addr add 10.1.0.2/16 dev encf500
[server]# ip link set dev encf500 up
[server]# ip addr add 10.2.0.2/16 dev ence400
[server]# ip link set dev ence400 up
```

2. If the dummy component was not compiled into the kernel, ensure that the dummy module was loaded.

   If necessary, load it by issuing:

```
[server]# modprobe dummy
```

3. Create a dummy interface with a virtual IP address 198.51.100.100 and a netmask 255.255.255.0:

```
 [server]# ip addr add 198.51.100.100/24 dev dummy0
 [server]# ip link set dev dummy0 up
```

4. Enable the network devices for this VIPA so that it accepts packets for this IP address.

   - IPv4 example:

```
 [server]# qethconf vipa add 198.51.100.100 encf500
qethconf: Added 198.51.100.100 to /sys/class/net/encf500/device/vipa/add4.
qethconf: For verification please use "qethconf ipa list"
 [server]# qethconf vipa add 198.51.100.100 ence400
qethconf: Added 198.51.100.100 to /sys/class/net/ence400/device/vipa/add4.
qethconf: For verification please use "qethconf ipa list"
```

   - For IPv6, the address is specified in IPv6 format:

```
[server]# qethconf vipa add 2002::1235:5678 encf500
qethconf: Added 2002:0000:0000:0000:0000:0000:1235:5678 to
          /sys/class/net/encf500/device/vipa/add6.
qethconf: For verification please use "qethconf ipa list"
[server]# qethconf vipa add 2002::1235:5678 ence400
qethconf: Added 2002:0000:0000:0000:0000:0000:1235:5678 to
          /sys/class/net/ence400/device/vipa/add6.
qethconf: For verification please use "qethconf ipa list"
```

5. Ensure that the addresses are set:

```
[server]# qethconf vipa list
vipa add 198.51.100.100 encf500
vipa add 198.51.100.100 ence400
```

6. Ensure that your service (such as the Apache web server) listens to the virtual IP address.
7. Set up a route to the virtual IP address (static routing) so that VIPA can be reached through the gateway with address `10.1.0.2`.

```
[router]# ip route add 198.51.100.100 via 10.1.0.2
```

What to do next

Now assume that an adapter outage occurs. You must then:

1. Delete the previously created route.

```
[router]# ip route del 198.51.100.100
```

2. Create the alternative route to the virtual IP address.

```
[router]# ip route add 198.51.100.100 via 10.2.0.2
```

Scenario: Virtual LAN (VLAN) support

VLAN technology works according to IEEE Standard 802.1Q by logically segmenting the network into different broadcast domains. Thus packets are switched only between ports that are designated for the same VLAN.

By containing traffic that originates on a particular LAN to other LANs within the same VLAN, switched virtual networks avoid wasting bandwidth. Wasted bandwidth is a drawback inherent in traditional bridged/switched networks where packets are often forwarded to LANs that do not require them.

The qeth device driver for OSA-Express (QDIO) and HiperSockets supports priority tags as specified by IEEE Standard 802.1Q for both layer 2 and layer 3.

Introduction to VLANs

Use VLANs to increase traffic flow and reduce latency. With VLANs, you can organize your network by traffic patterns rather than by physical location.

In a conventional network topology, such as that shown in the following figure, devices communicate across LAN segments in different broadcast domains by using routers. Although routers add latency by delaying transmission of data while they are using more of the data packet to determine destinations, they are preferable to building a single broadcast domain. A single domain can easily be flooded with traffic.



Figure 74. Conventional routed network

By organizing the network into VLANs by using Ethernet switches, distinct broadcast domains can be maintained without the latency that is introduced by multiple routers. As the following figure shows, a single router can provide the interfaces for all VLANs that appeared as separate LAN segments in the previous figure.



Figure 75. Switched VLAN network

The following figure shows how VLANs can be organized logically, according to traffic flow, rather than being restricted by physical location. If workstations 1-3 communicate mainly with the small server, VLANs can be used to organize only these devices in a single broadcast domain that keeps broadcast traffic within the group. This setup reduces traffic both inside the domain and outside, on the rest of the network.



Figure 76. VLAN network organized for traffic flow

Configuring VLAN devices

Configure VLANs with the **ip link add** command. See the **ip-link** man page for details.

About this task

Information about the current VLAN configuration is available by listing the files in `/proc/net/vlan/*` with **cat** or **more**. For example:

```
# cat /proc/net/vlan/config
VLAN Dev name    | VLAN ID
Name-Type: VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD  bad_proto_recvd: 0
encd300.100       | 100  | encd300
encd300.200       | 200  | encd300
encd300.300       | 300  | encd300

# cat /proc/net/vlan/encd300.300
encd300.300  VID: 300       REORDER_HDR: 1  dev->priv_flags: 1
         total frames received:     10914061
          total bytes received:   1291041929
      Broadcast/Multicast Rcvd:            6

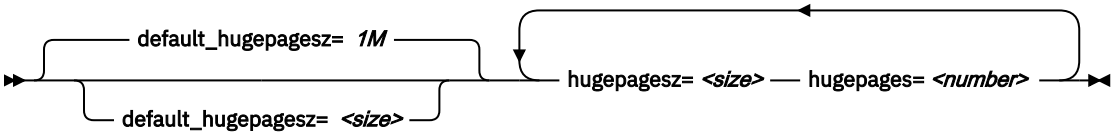
      total frames transmitted:     10471684
       total bytes transmitted:   4170258240
            total headroom inc:            0
           total encap on xmit:     10471684
Device: encd300
INGRESS priority mappings: 0:0  1:0  2:0  3:0  4:0  5:0  6:0 7:0
EGRESS  priority Mappings:
#
```

Example: Creating two VLANs

VLANs are allocated in an existing interface that represents a physical Ethernet LAN.

The following example creates two VLANs, one with ID 3 and one with ID 5.

```
    ip addr add 198.51.160.23/19 dev ence400
    ip link set dev ence400 up
    ip link add dev ence400.3 link ence400 type vlan id 3
    ip link add dev ence400.5 link ence400 type vlan id 5
```

The **ip link add** commands added interfaces "ence400.3" and "ence400.5", which you can then configure:

```
ip addr add  1.2.3.4/24    dev ence400.3
ip link set dev ence400.3 up
ip addr add  10.100.2.3/16 dev ence400.5
ip link set dev ence400.5 up
```

The traffic that flows out of ence400.3 is in the VLAN with ID=3. This traffic is not received by other stacks that listen to VLANs with ID=4.

The internal routing table ensures that every packet to 1.2.3.x goes out through ence400.3 and everything to 10.100.x.x through ence400.5. Traffic to 198.51.1xx.x flows through ence400 (without a VLAN tag).

To remove one of the VLAN interfaces:

```
ip link set dev ence400.3 down
ip link delete ence400.3 type vlan
```

HiperSockets Network Concentrator

You can configure a HiperSockets Network Concentrator on a QETH device in layer 3 mode.

Before you begin: The instructions that are given apply to IPv4 only. The HiperSockets Network Concentrator connector settings are available in layer 3 mode only.

The HiperSockets Network Concentrator connects systems to an external LAN within one IP subnet using HiperSockets. HiperSockets Network Concentrator connected systems look as if they were directly connected to the LAN. This simplification helps to reduce the complexity of network topologies that result from server consolidation.

Without changing the network setup, you can use HiperSockets Network Concentrator to port systems:

- From the LAN into an IBM Z Server environment
- From systems that are connected by a different HiperSockets Network Concentrator into an IBM Z Server environment

Thus, HiperSockets Network Concentrator helps to simplify network configuration and administration.

Design

A connector Linux system forwards traffic between the external OSA interface and one or more internal HiperSockets interfaces. The forwarding is done via IPv4 forwarding for unicast traffic and via a particular bridging code (xcec_bridge) for multicast traffic.

A script named ip_watcher.pl observes all IP addresses registered in the HiperSockets network and configures them as proxy ARP entries (see “Configuring a device for proxy ARP” on page 282) on the OSA interfaces. The script also establishes routes for all internal systems to enable IP forwarding between the interfaces.

All unicast packets that cannot be delivered in the HiperSockets network are handed over to the connector by HiperSockets. The connector also receives all multicast packets to bridge them.

Setup

The setup principles for configuring the HiperSockets Network Concentrator are as follows:

leaf nodes
The leaf nodes do not require a special setup. To attach them to the HiperSockets network, their setup should be as if they were directly attached to the LAN. They do not have to be Linux systems.

connector systems
In the following, HiperSockets Network Concentrator IP refers to the subnet of the LAN that is extended into the HiperSockets net.

- If you want to support forwarding of all packet types, define the OSA interface for traffic into the LAN as a multicast router (see “Setting up a Linux router in layer 3” on page 275).

- All HiperSockets interfaces that are involved must be set up as connectors: set the route4 attributes of the corresponding devices to "primary_connector" or to "secondary_connector". Alternatively, you can add the OSA interface name to the start script as a parameter. This option results in HiperSockets Network Concentrator ignoring multicast packets, which are then not forwarded to the HiperSockets interfaces.
- IP forwarding must be enabled for the connector partition. Enable the forwarding either manually with the command

  ```
  sysctl -w net.ipv4.ip_forward=1
  ```

  Alternatively, you can enable IP forwarding in the `/etc/sysctl.conf` configuration file to activate IP forwarding for the connector partition automatically after booting.
- The network routes for the HiperSockets interface must be removed. A network route for the HiperSockets Network Concentrator IP subnet must be established through the OSA interface. To establish a route, assign the IP address 0.0.0.0 to the HiperSockets interface. At the same time, assign an address used in the HiperSockets Network Concentrator IP subnet to the OSA interface. These assignments set up the network routes correctly for HiperSockets Network Concentrator.
- To start HiperSockets Network Concentrator, run the script `start_hsnc.sh`. You can specify an interface name as optional parameter. The interface name makes HiperSockets Network Concentrator use the specified interface to access the LAN. There is no multicast forwarding in that case.
- To stop HiperSockets Network Concentrator, use the command **`killall ip_watcher.pl`** to remove changes that are caused by running HiperSockets Network Concentrator.

Availability setups

If a connector system fails during operation, it can simply be restarted. If all the startup commands are run automatically, it will instantaneously be operational again after booting. Two common availability setups are mentioned here:

One connector partition and one monitoring system
: As soon as the monitoring system cannot reach the connector for a specific timeout (for example, 5 seconds), it restarts the connector. The connector itself monitors the monitoring system. If it detects (with a longer timeout than the monitoring system, for example, 15 seconds) a monitor system failure, it restarts the monitoring system.

Two connector systems monitoring each other
: In this setup, there is an active and a passive system. As soon as the passive system detects a failure of the active connector, it takes over operation. To take over operation, it must reset the other system to release all OSA resources for the multicast_router operation. The failed system can then be restarted manually or automatically, depending on the configuration. The passive backup HiperSockets interface can either switch into primary_connector mode during the failover, or it can be set up as secondary_connector. A secondary_connector takes over the connecting function, as soon as there is no active primary_connector. This setup has a faster failover time than the first one.

Hints

- The MTU of the OSA and HiperSockets link should be of the same size. Otherwise, multicast packets that do not fit in the link's MTU are discarded as there is no IP fragmentation for multicast bridging. Warnings are printed to a corresponding syslog destination.
- The script `ip_watcher.pl` prints error messages to the standard error descriptor of the process.
- `xcec-bridge` logs messages and errors to syslog. On Red Hat Enterprise Linux 9.1, issue **`journalctl`** to find these messages.
- Registering all internal addresses with the OSA adapter can take several seconds for each address.
- To shut down the HiperSockets Network Concentrator function, issue `killall ip_watcher.pl`. This script removes all routing table and Proxy ARP entries added during the use of HiperSockets Network Concentrator.

Note:

1. Broadcast bridging is active only on OSA or HiperSockets hardware that can handle broadcast traffic without causing a bridge loop. If you see the message "`Setting up broadcast echo filtering for ... failed`" in the message log when you set the qeth device online, broadcast bridging is not available.
2. Unicast packets are routed by the common Linux IPv4 forwarding mechanisms. As bridging and forwarding are done at the IP Level, the IEEE 802.1q VLAN and the IPv6 protocol are not supported.

Examples for setting up a network concentrator

An example of a network environment with a network concentrator.

Figure 77 on page 301 shows a network environment where a Linux instance C acts as a network concentrator that connects other operating system instances on a HiperSockets LAN to an external LAN.



Figure 77. HiperSockets network concentrator setup

Setup for the network concentrator C:

The HiperSockets interface enca1c0 (you can infer the interface name from the device bus-ID 0.0.a1c0) has IP address 10.20.30.51/24. The default gateway is 10.20.30.1.

Issue:

```
# echo primary_connector > /sys/bus/ccwgroup/drivers/qeth/0.0.a1c0/route4
```

The OSA-Express CHPID in QDIO mode interface enca1c4 has IP address 10.20.30.11/24. The default gateway is 10.20.30.1.

Issue:

```
# echo multicast_router > /sys/bus/ccwgroup/drivers/qeth/0.0.a1c4/route4
```

To enable IP forwarding issue:

```
# sysctl -w net.ipv4.ip_forward=1
```

To remove the network routes for the HiperSockets interface issue:

```
# ip route del 10.20.30/24
```

To start the HiperSockets network concentrator, run the script `start_hsnc.sh`. Issue:

```
# start_hsnc.sh &
```

Setup for G:
No special setup required. The HiperSockets interface has IP address 10.20.30.54/24. The default gateway is 10.20.30.1.

Setup for workstation:
No special setup required. The network interface IP address is 10.20.30.120/24. The default gateway is 10.20.30.1.

Figure 78 on page 302 shows the example of Figure 77 on page 301 with an additional mainframe. On the second mainframe a Linux instance D acts as a HiperSockets network concentrator.



Figure 78. Expanded HiperSockets network concentrator setup

The configuration of C, G, and the workstation remain the same as for Figure 77 on page 301.

Setup for the network concentrator D:
The HiperSockets interface enca1d0 has the corresponding device bus-ID 0.0.a1d0 and IP address 0.0.0.0. Issue:

```
# echo primary_connector > /sys/bus/ccwgroup/drivers/qeth/0.0.a1d0/route4
```

The OSA-Express CHPID in QDIO mode interface enca1d1 has IP address 10.20.30.50/24. The default gateway is 10.20.30.1.

D is not configured as a multicast router, it therefore only forwards unicast packets.

To enable IP forwarding issue:

```
# sysctl -w net.ipv4.ip_forward=1
```

Tip: See the Red Hat documentation website for information about using configuration files to automatically enable IP forwarding when Linux boots:
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

To start the HiperSockets network concentrator, run the script `start_hsnc.sh`. Issue:

```
# start_hsnc.sh &
```

Setup for H:
No special setup required. The HiperSockets interface has IP address 10.20.30.55/24. The default gateway is 10.20.30.1.

Setting up for DHCP with IPv4

For connections through an OSA-Express adapter in QDIO mode, the OSA-Express adapter offloads ARP, MAC header, and MAC address handling.

For information about MAC headers, see "MAC headers in layer 3 mode" on page 250.

Because a HiperSockets connection does not go out on a physical network, there are no ARP, MAC headers, and MAC addresses for packets in a HiperSockets LAN. The resulting problems for DHCP are the same in both cases and the fixes for connections through the OSA-Express adapter also apply to HiperSockets.

Dynamic Host Configuration Protocol (DHCP) is a TCP/IP protocol that allows clients to obtain IP network configuration information (including an IP address) from a central DHCP server. The DHCP server controls whether the address it provides to a client is allocated permanently or is leased temporarily. DHCP specifications are described by RFC 2131"Dynamic Host Configuration Protocol" and RFC 2132 "DHCP options and BOOTP Vendor Extensions", which are available at

```
www.ietf.org
```

Two types of DHCP environments must be taken into account:

- DHCP that uses OSA-Express adapters in QDIO mode
- DHCP in a z/VM VSWITCH or guest LAN

For information about setting up DHCP for a Linux instance in a z/VM guest LAN environment, see Redpaper *Linux on IBM eServer™ zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP-3596 at

```
www.ibm.com/redbooks
```

The programs *dhclient* and *dhcp* are examples of a DHCP client and a DHCP server you can use.

Required options for using dhclient with layer3

You must configure the DHCP client program `dhclient` to use it on Linux on IBM Z with layer3.

- Run the DHCP client with an option that instructs the DHCP server to broadcast its response to the client.

  Because the OSA-Express adapter in QDIO mode forwards packets to Linux based on IP addresses, a DHCP client that requests an IP address cannot receive the response from the DHCP server without this option.
- Run the DHCP client with an option that specifies the client identifier string.

  By default, the client uses the MAC address of the network interface. Hence, without this option, all Linux instances that share the OSA-Express adapter in QDIO mode would also have the same client identifier.

See the documentation for `dhclient` about how to select these options.

You need no special options for the DHCP server program, `dhcp`. You need no special options for using `dhcp`.

Setting up Linux as a LAN sniffer

You can set up a Linux instance to act as a LAN sniffer, for example, to make data on LAN traffic available to tools like **`tcpdump`** or Wireshark.

The LAN sniffer can be:

- A HiperSockets Network Traffic Analyzer for LAN traffic between LPARs
- A LAN sniffer for LAN traffic between z/VM guest virtual machines, for example, through a z/VM virtual switch (VSWITCH)

Setting up a HiperSockets network traffic analyzer

A HiperSockets network traffic analyzer (NTA) runs in an LPAR and monitors LAN traffic between LPARs.

Before you begin

- Your Linux instance must run in LPAR mode.
- On the SE, the LPARs must be authorized for analyzing and being analyzed.

  Tip: SE authorization changes for the HiperSockets network traffic analyzer require re-creating the device by ungrouping and regrouping (see “Removing a qeth group device” on page 260 and “Creating a qeth group device” on page 259). Do any authorization changes before you configure the NTA device.
- You need a traffic-dumping tool such as **tcpdump**.

About this task

HiperSockets NTA is available to trace both layer 3 and layer 2 network traffic, but the analyzing device itself must be configured as a layer 3 device. The analyzing device is a dedicated NTA device and cannot be used as a regular network interface.

Linux setup:

Ensure that the qeth device driver module was loaded.

Procedure

Perform the following steps:

1. Configure a HiperSockets interface dedicated to analyzing with the `layer2` sysfs attribute set to `0` and the `sniffer` sysfs attribute set to `1`.

   For example, assuming the HiperSockets interface is enca1c0 with device bus-ID 0.0.a1c0:

   ```
   # znetconf -a a1c0 -o layer2=0 -o sniffer=1
   ```

   The **znetconf** command also sets the device online. For more information about **znetconf**, see “znetconf - List and configure network devices” on page 767. The qeth device driver automatically sets the buffer_count attribute to 128 for the analyzing device.

2. Activate the device (no IP address is needed):

   ```
   # ip link set enca1c0 up
   ```

3. Switch the interface into promiscuous mode:

   ```
   # tcpdump -i enca1c0
   ```

Results

The device is now set up as a HiperSockets network traffic analyzer.

Hint: A HiperSockets network traffic analyzer with no free empty inbound buffers might have to drop packets. Dropped packets are reflected in the "dropped counter" of the HiperSockets network traffic analyzer interface and reported by **tcpdump**.

Example:

```
# ip -s link show dev enca1c0
...
    RX: bytes  packets  errors  dropped overrun mcast
    223242     6789     0       5       0       176
...
# tcpdump -i enca1c0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enca1c0, link-type EN10MB (Ethernet), capture size 96 bytes
...
5 packets dropped by kernel
```

Setting up a z/VM guest LAN sniffer

You can set up a guest LAN sniffer on a virtual NIC that is coupled to a z/VM VSWITCH or guest LAN.

Before you begin

- You need class B authorization on z/VM.
- The Linux instance to be set up as a guest LAN sniffer must run as a guest of the same z/VM system as the guest LAN you want to investigate.

About this task

If a virtual switch connects to a VLAN that includes nodes outside the z/VM system, these external nodes are beyond the scope of the sniffer.

For information about VLANs and z/VM virtual switches, see *z/VM: Connectivity*, SC24-6267.

Procedure

- Set up Linux.

  Ensure that the qeth device driver is loaded.
- Set up z/VM.

  Ensure that the z/VM guest virtual machine on which you want to set up the guest LAN sniffer is authorized for the switch or guest LAN and for promiscuous mode.

  For example, if your virtual NIC is coupled to a z/VM virtual switch, perform the following steps on your z/VM system:

  a) Check whether the z/VM guest virtual machine already has the requisite authorizations. Enter a CP command of this form:

  ```
  q vswitch <switchname> promisc
  ```

  where *<switchname>* is the name of the virtual switch. If the output lists the z/VM guest virtual machine as authorized for promiscuous mode, no further setup is needed.

  b) If the output from step “1” on page 305 does not list the guest virtual machine, check if the guest is authorized for the virtual switch. Enter a CP command of this form:

  ```
  q vswitch <switchname> acc
  ```

  where *<switchname>* is the name of the virtual switch.

  If the output lists the z/VM guest virtual machine as authorized, you must temporarily revoke the authorization for the switch before you can grant authorization for promiscuous mode. Enter a CP command of this form:

  ```
  set vswitch <switchname> revoke <userid>
  ```

where *<switchname>* is the name of the virtual switch and *<userid>* identifies the z/VM guest virtual machine.

c) Authorize the Linux instance for the switch and for promiscuous mode. Enter a CP command of this form:

```
set vswitch <switchname> grant <userid> promisc
```

where *<switchname>* is the name of the virtual switch and *<userid>* identifies the z/VM guest virtual machine.

For details about the CP commands that are used here and for commands you can use to check and assign authorizations for other types of guest LANs, see *z/VM: CP Commands and Utilities Reference*, SC24-6268.

Chapter 17. OSA-Express SNMP subagent support

LPAR and z/VM: The SNMP subagent support applies to Linux in LPAR mode and to Linux on z/VM.

The OSA-Express Simple Network Management Protocol (SNMP) subagent (osasnmpd) supports management information bases (MIBs) for the OSA-Express features.

The subagent supports OSA-Express features as shown in Table 37 on page 241.

This subagent capability through the OSA-Express features is also called *Direct SNMP* to distinguish it from another method of accessing OSA SNMP data through OSA/SF, a package for monitoring and managing OSA features that does not run on Linux.

To use the osasnmpd subagent, you need:

- An OSA-Express feature running in QDIO mode with the latest textual MIB file for the appropriate LIC level (recommended)
- The qeth device driver for OSA-Express (QDIO)
- The osasnmpd subagent from the s390utils RPM
- The net-snmp package delivered with Red Hat Enterprise Linux 6

What you should know about osasnmpd

The osasnmpd subagent requires a master agent to be installed on a Linux system.

You get the master agent from the net-snmp package. The subagent uses the Agent eXtensibility (AgentX) protocol to communicate with the master agent.

net-snmp is an open source project that is owned by the Open Source Development Network, Inc. (OSDN). For more information about net-snmp visit:

```
net-snmp.sourceforge.net
```

When the master agent (snmpd) is started on a Linux system, it binds to a port (default 161) and awaits requests from SNMP management software. Subagents can connect to the master agent to support MIBs of special interest (for example, OSA-Express MIB). When the osasnmpd subagent is started, it retrieves the MIB objects of the OSA-Express features currently present on the Linux system. It then registers with the master agent the object IDs (OIDs) for which it can provide information.

An OID is a unique sequence of dot-separated numbers (for example, .1.3.6.1.4.1.2) that represents a particular information. OIDs form a hierarchical structure. The longer the OID, that is the more numbers it is made up of, the more specific is the information that is represented by the OID. For example, .1.3.6.1.4.1.2 represents all IBM-related network information while ..1.3.6.1.4.1.2.6.188 represents all OSA-Express-related information.

A MIB corresponds to a number of OIDs. MIBs provide information about their OIDs including textual representations the OIDs. For example, the textual representation of .1.3.6.1.4.1.2 is .iso.org.dod.internet.private.enterprises.ibm.

The structure of the MIBs might change when updating the OSA-Express Licensed Internal Code (LIC) to a newer level. If MIB changes are introduced by a new LIC level, you must download the appropriate MIB file for the LIC level (see “Downloading the IBM OSA-Express MIB” on page 308). You do not need to update the subagent. Place the updated MIB file in a directory that is searched by the master agent.



Figure 79. OSA-Express SNMP agent flow

Figure 79 on page 308 illustrates the interaction between the snmpd master agent and the osasnmpd subagent.

Example: This example shows the processes that run after the snmpd master agent and the osasnmpd subagent are started. When you start osasnmpd, a daemon called osasnmpd starts. In the example, PID 687 is the SNMP master agent and PID 729 is the OSA-Express SNMP subagent process:

```
ps -ef | grep snmp

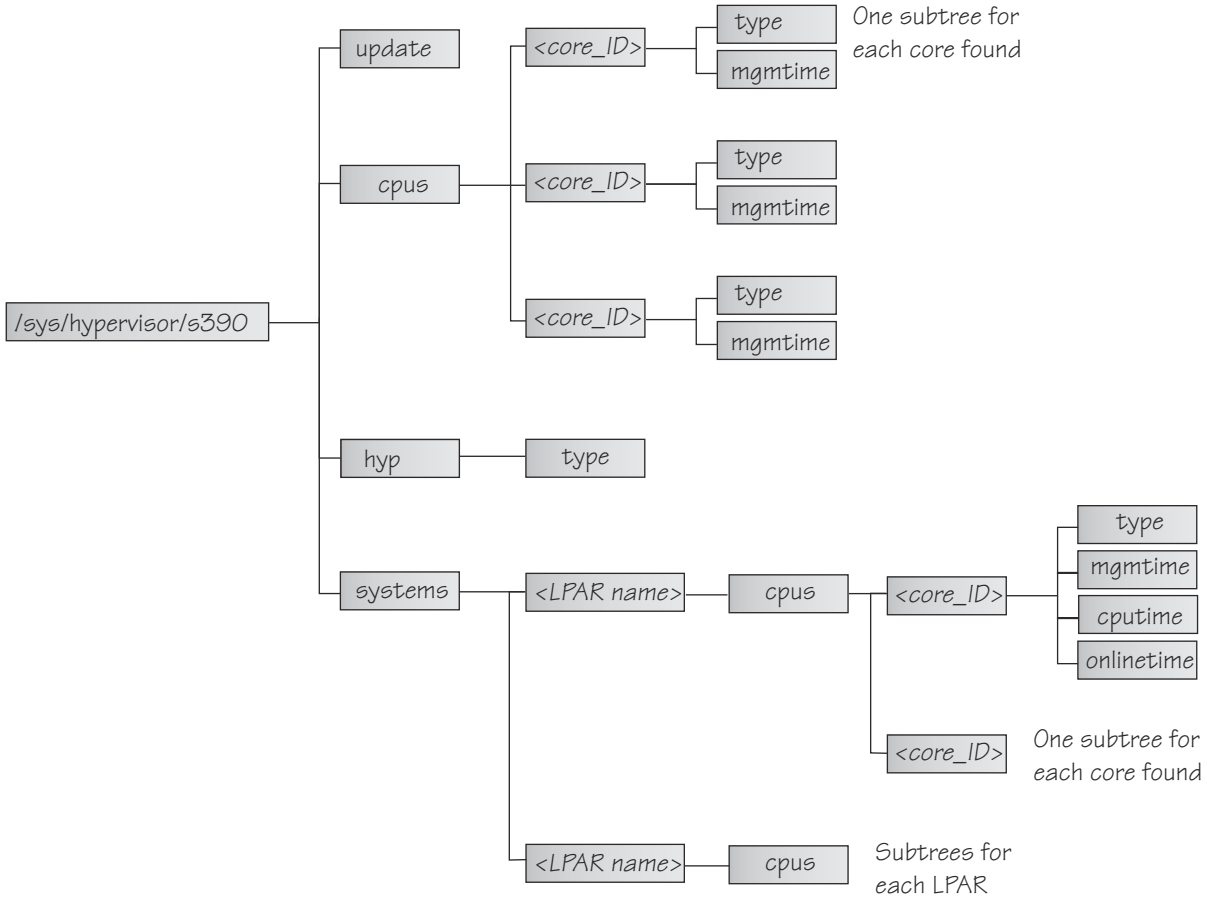
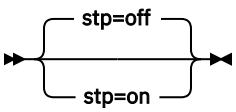
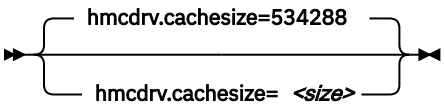
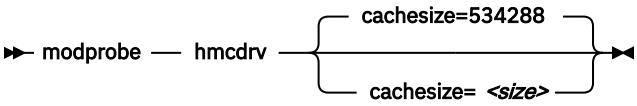
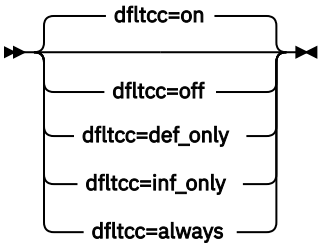
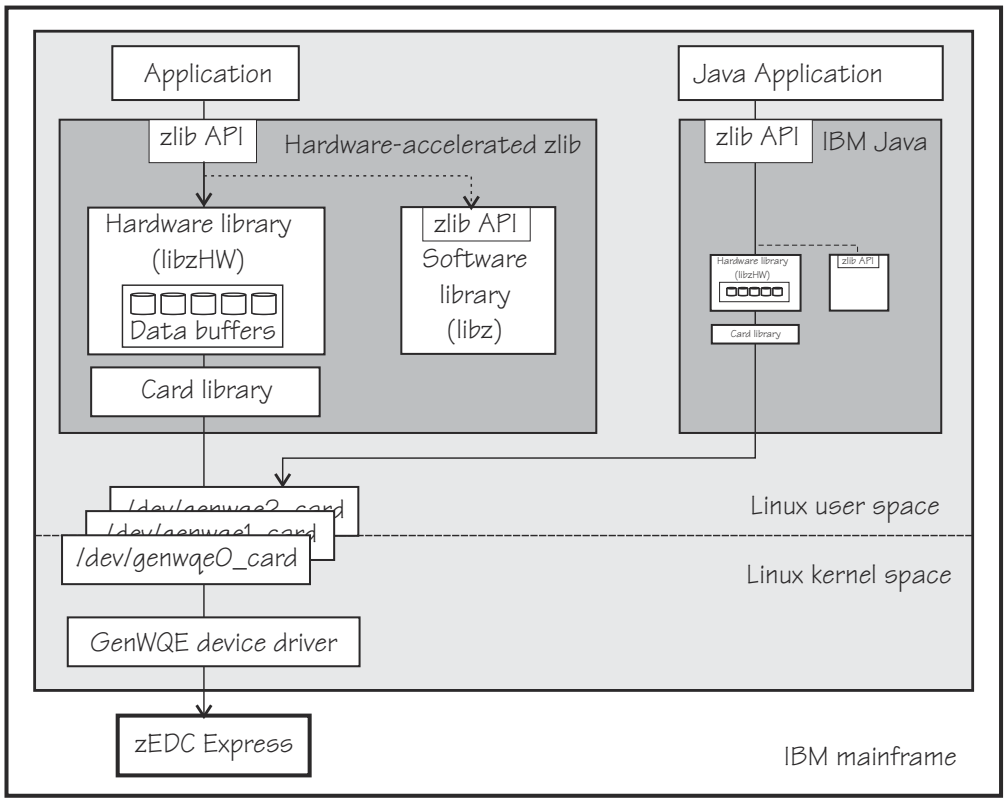
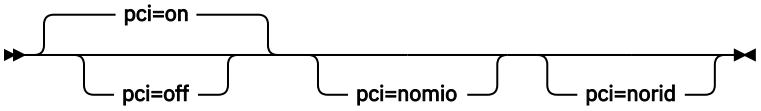
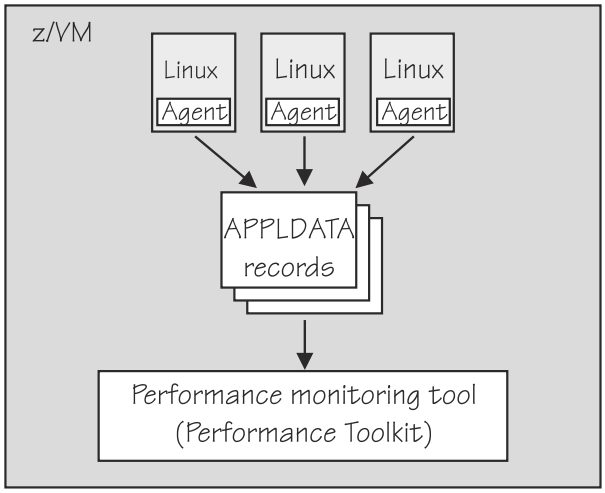
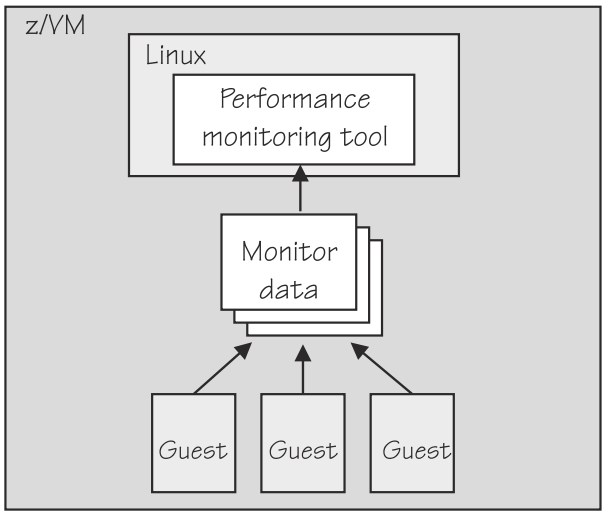
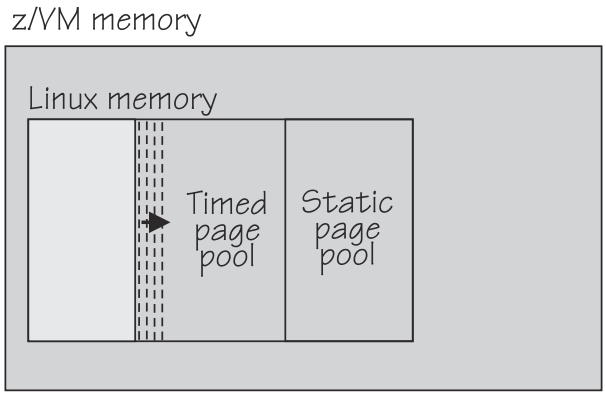
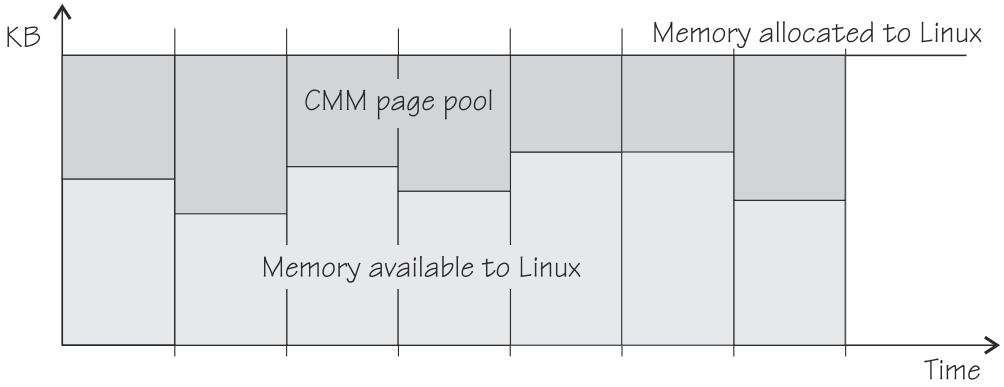
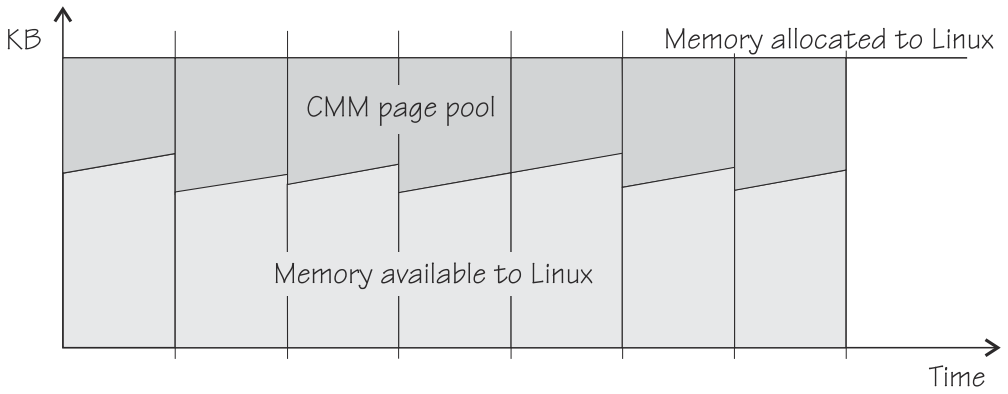
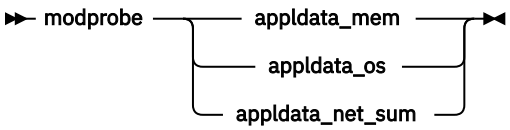
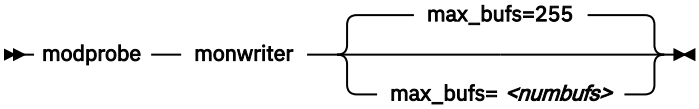
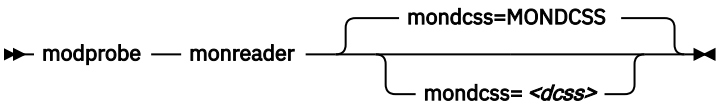
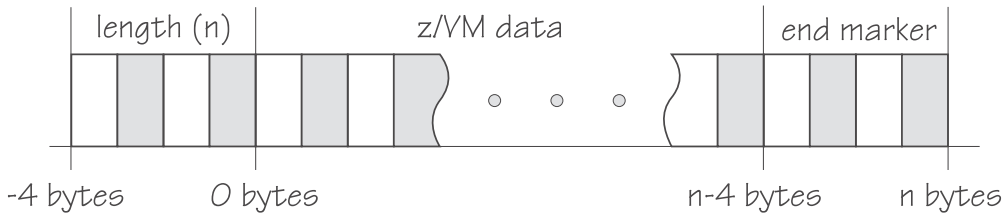
USER       PID
root       687     1  0 11:57 pts/1    00:00:00 snmpd
root       729   659  0 13:22 pts/1    00:00:00 osasnmpd
```

When the master agent receives an SNMP request for an OID that is registered by a subagent, the master agent uses the subagent to collect any requested information and to perform any requested operations. The subagent returns any requested information to the master agent. Finally, the master agent returns the information to the originator of the request.

Setting up osasnmpd

You must download the IBM OSA-Express MIB and configure access control before you use can the osasnmpd subagent.

Downloading the IBM OSA-Express MIB

Keep your MIB file up to date by downloading the latest version.

About this task

Perform the following steps to download the IBM OSA-Express MIB. The MIB file is valid only for hardware that supports the OSA-Express adapter.

Procedure

1. Go to www.ibm.com/servers/resourcelink

   A user ID and password are required. If you do not yet have one, you can apply for a user ID.
2. Sign in.
3. Select **Library** from the navigation area.
4. Under **Library shortcuts**, select **Open Systems Adapter (OSA) Library**.

5. Follow the link for **OSA-Express Direct SNMP MIB module**.
6. Select and download the MIB for your LIC level.
7. Rename the MIB file to the name specified in the MIBs definition line and use the extension `.txt`.

   Example: If the definition line in the MIB looks like this:

   ```
   ==>IBM-OSA-MIB DEFINITIONS ::= BEGIN
   ```

   Rename the MIB to `IBM-OSA-MIB.txt`.
8. Place the MIB into `/usr/share/snmp/mibs`.

   If you want to use a different directory, be sure to specify the directory in the snmp.conf configuration file (see step "10" on page 311).

Results

You can now make the OID information from the MIB file available to the master agent. You can then use textual OIDs instead of numeric OIDs when you use master agent commands.

See also the FAQ (How do I add a MIB to the tools?) for the master agent package at

```
net-snmp.sourceforge.net/FAQ.html
```

Configuring access control

To start successfully, the subagent requires at least read access to the standard MIB-II on the local node.

About this task

During subagent startup or when network interfaces are added or removed, the subagent must query OIDs from the interfaces group of the standard MIB-II.

Given here is an example of how you can use the `snmpd.conf` and `snmp.conf` configuration files to assign access rights with the View-Based Access Control Mechanism (VACM). The following access rights are assigned on the local node:

- General read access for the scope of the standard MIB-II
- Write access for the scope of the OSA-Express MIB
- Public local read access for the scope of the interfaces MIB

The example is intended for illustration purposes only. Depending on the security requirements of your installation, you might need to define your access differently. See the snmpd man page for a more information about assigning access rights to snmpd.

Procedure

1. See the Red Hat Enterprise Linux 9.1 documentation to find out where you must place the `snmpd.conf` file.

   Some of the possible locations are:

   - `/etc`
   - `/etc/snmp`
2. Open snmpd.conf with your preferred text editor. There might be a sample in `usr/share/doc/packages/net-snmp/EXAMPLE.conf`
3. Find the security name section and include a line of this form to map a community name to a security name:

   ```
   com2sec <security-name> <source> <community-name>
   ```

   where:

<security-name>
is given access rights through further specifications within `snmpd.conf`.

<source>
is the IP-address or DNS-name of the accessing system, typically a Network Management Station.

<community-name>
is the community string used for basic SNMP password protection.

Example:

```
#       sec.name    source       community
com2sec osasec      default      osacom
com2sec pubsec      localhost    public
```

4. Find the group section.

   Use the security name to define a group with different versions of the master agent for which you want to grant access rights. Include a line of this form for each master agent version:

   ```
   group <group-name> <security-model> <security-name>
   ```

   where:

   <group-name>
   is a group name of your choice.

   <security-model>
   is the security model of the SNMP version.

   <security-name>
   is the same as in step “3” on page 309.

   Example:

   ```
   #        groupName    securityModel    securityName
   group    osagroup     v1               osasec
   group    osagroup     v2c              osasec
   group    osagroup     usm              osasec
   group    osasnmpd     v2c              pubsec
   ```

   Group "osasnmpd" with community "public" is required by osasnmpd to determine the number of network interfaces.

5. Find the view section and define your views.

   A view is a subset of all OIDs. Include lines of this form:

   ```
   view  <view-name>  <included|excluded>   <scope>
   ```

   where:

   <view-name>
   is a view name of your choice.

   <included|excluded>
   indicates whether the following scope is an inclusion or an exclusion statement.

   <scope>
   specifies a subtree in the OID tree.

   Example:

   ```
   #    name       incl/excl    subtree                 mask(optional)
   view allview    included     .1
   view osaview    included     .1.3.6.1.4.1.2
   view ifmibview  included     interfaces
   view ifmibview  included     system
   ```

   View "allview" encompasses all OIDs while "osaview" is limited to IBM OIDs.
   The numeric OID provided for the subtree is equivalent to the textual OID

".iso.org.dod.internet.private.enterprises.ibm" View "ifmibview" is required by osasnmpd to determine the number of network interfaces.

Tip: Specifying the subtree with a numeric OID leads to better performance than using the corresponding textual OID.

6. Find the access section and define access rights. Include lines of this form:

```
access <group-name> "" any noauth exact <read-view> <write-view> none
```

where:

<group-name>
is the group you defined in step “4” on page 310.

<read-view>
is a view for which you want to assign read-only rights.

<write-view>
is a view for which you want to assign read-write rights.

Example:

```
#      group    context sec.model sec.level prefix read      write   notif
access osagroup ""      any       noauth    exact  allview   osaview none
access osasnmpd ""      v2c       noauth    exact  ifmibview none    none
```

The access line of the example gives read access to the "allview" view and write access to the "osaview". The second access line gives read access to the "ifmibview".

7. Also include the following line to enable the AgentX support:

```
master agentx
```

AgentX support is compiled into the net-snmp master agent.

8. Save and close `snmpd.conf`.
Example of an `snmpd.conf` file:

```
#       sec.name       source        community
com2sec osasec         default       osacom
com2sec pubsec         localhost     public
#       groupName      securityModel securityName
group   osagroup       v1            osasec
group   osagroup       v2c           osasec
group   osagroup       usm           osasec
group   osasnmpd       v2c           pubsec
#       name          incl/excl  subtree        mask(optional)
view    allview       included   .1
view    osaview       included   .1.3.6.1.4.1.2
view    ifmibview     included   interfaces
view    ifmibview     included      system
#       group          context sec.model sec.level prefix read       write    notif
access  osagroup       ""      any       noauth    exact  allview    osaview  none
access  osasnmpd       ""      v2c       noauth    exact  ifmibview  none     none
master  agentx
```

9. Open `~/.snmp/snmp.conf` with your preferred text editor.

Tip: See **`man snmp.conf`** for possible locations of `snmp.conf`.

10. Include a line of this form to specify the directory to be searched for MIBs:

```
mibdirs +<mib-path>
```

Example:

```
mibdirs +/usr/share/snmp/mibs
```

11. Include a line of this form to make the OSA-Express MIB available to the master agent:

```
mibs +<mib-name>
```

where *<mib-name>* is the stem of the MIB file name you assigned in "Downloading the IBM OSA-Express MIB" on page 308.

Example: `mibs +IBM-OSA-MIB`

12. Define defaults for the version and community to be used by the snmp commands. Add lines of this form:

```
defVersion   <version>
defCommunity <community-name>
```

where *<version>* is the SNMP protocol version and *<community-name>* is the community you defined in step "3" on page 309.

Example:

```
defVersion   2c
defCommunity osacom
```

These default specifications simplify issuing master agent commands.

13. Save and close `~/.snmp/snmp.conf`.

Working with the osasnmpd subagent

Working with the osasnmpd subagent includes starting it, checking the log file, issuing queries, and stopping the subagent.

Working with osasnmpd comprises the following tasks:

- "Starting the osasnmpd subagent" on page 312
- "Checking the log file" on page 312
- "Issuing queries" on page 313
- "Stopping osasnmpd" on page 314

Starting the osasnmpd subagent

Use the **osasnmpd** command to start the osasnmpd subagent.

Procedure

After you download the s390utils-osasnmpd package and set up the osasnmpd subagent, start the subagent with the command:

```
# osasnmpd
```

The osasnmpd subagent, in turn, starts a daemon that is called osasnmpd.

For command options see the **osasnmpd** command man page.

If you restart the master agent, you must also restart the subagent. When the master agent is started, it does not look for already running subagents. Any running subagents must also be restarted to be register with the master agent.

Checking the log file

Warnings and messages are written to the log file of either the master agent or the OSA-Express subagent. It is good practice to check these files at regular intervals.

Example

This example assumes that the default subagent log file is used. The lines in the log file show the messages after a successful OSA-Express subagent initialization.

```
# cat /var/log/osasnmpd.log
IBM OSA-E NET-SNMP 5.1.x subagent version  1.3.0
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.2.1.10.7.2.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.1.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.3.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.4.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.8.
OSA-E microcode level is 611 for interface encf500
Initialization of OSA-E subagent successful...
```

Issuing queries

You can issue queries against your SNMP setup.

About this task

Examples of what SNMP queries might look like are given here. For more comprehensive information about the master agent commands, see the **snmpcmd** man page.

The commands can use either numeric or textual OIDs. While the numeric OIDs might provide better performance, the textual OIDs are more meaningful and give a hint about which information is requested.

Examples

The query examples assume an interface, encf500, for which the CHPID is 6B. You can use the **lsqeth** command to find the mapping of interface names to CHPIDs.

- To list the ifIndex and interface description relation (on one line):

```
# snmpget -v 2c -c osacom localhost interfaces.ifTable.ifEntry.ifDescr.6
interfaces.ifTable.ifEntry.ifDescr.6 = encf500
```

  Using this GET request you can see that encf500 has the ifIndex 6 assigned.

- To find the CHPID numbers for your OSA devices:

```
# snmpwalk -OS -v 2c -c osacom localhost .1.3.6.1.4.1.2.6.188.1.1.1.1
IBM-OSA-MIB::ibmOSAExpChannelNumber.6 = Hex-STRING: 00 6B
IBM-OSA-MIB::ibmOSAExpChannelNumber.7 = Hex-STRING: 00 7A
IBM-OSA-MIB::ibmOSAExpChannelNumber.8 = Hex-STRING: 00 7D
```

  The first line of the command output, with index number 6, corresponds to CHPID 0x6B of the encf500 example. The example assumes that the community osacom is authorized as described in “Configuring access control” on page 309.

  If you provided defaults for the SNMP version and the community (see step “12” on page 312), you can omit the `-v` and `-c` options:

```
# snmpwalk -OS localhost .1.3.6.1.4.1.2.6.188.1.1.1.1
IBM-OSA-MIB::ibmOSAExpChannelNumber.6 = Hex-STRING: 00 6B
IBM-OSA-MIB::ibmOSAExpChannelNumber.7 = Hex-STRING: 00 7A
IBM-OSA-MIB::ibmOSAExpChannelNumber.8 = Hex-STRING: 00 7D
```

  You can obtain the same output by substituting the numeric OID .1.3.6.1.4.1.2.6.188.1.1.1.1 with its textual equivalent:

```
.iso.org.dod.internet.private.enterprises.ibm.ibmProd.ibmOSAMib.ibmOSAMibObjects.ibmOSAExpChannelTable.ibmOSAExpChannelEntry.ibmOSAExpChannelNumber
```

  You can shorten this unwieldy OID to the last element, ibmOsaExpChannelNumber:

```
# snmpwalk -OS localhost ibmOsaExpChannelNumber
IBM-OSA-MIB::ibmOSAExpChannelNumber.6 = Hex-STRING: 00 6B
IBM-OSA-MIB::ibmOSAExpChannelNumber.7 = Hex-STRING: 00 7A
IBM-OSA-MIB::ibmOSAExpChannelNumber.8 = Hex-STRING: 00 7D
```

- To find the port type for the interface with index number 6:

```
# snmpwalk -OS localhost .1.3.6.1.4.1.2.6.188.1.4.1.2.6
IBM-OSA-MIB::ibmOsaExpEthPortType.6 = INTEGER: fastEthernet(81)
```

fastEthernet(81) corresponds to card type OSD_100.

Using the short form of the textual OID:

```
# snmpwalk -OS localhost ibmOsaExpEthPortType.6
IBM-OSA-MIB::ibmOsaExpEthPortType.6 = INTEGER: fastEthernet(81)
```

Specifying the index, 6 in the example, limits the output to the interface of interest.

Stopping osasnmpd

The subagent can be stopped by sending either a SIGINT or SIGTERM signal to the thread.

About this task

Avoid stopping the subagent with **kill -9** or with **kill -SIGKILL**. These commands do not allow the subagent to unregister the OSA-Express MIB objects from the SNMP master agent. This can cause problems when restarting the subagent.

If you saved the subagent PID to a file when you started it, you can consult this file for the PID. Otherwise, you can issue a **ps** command to find it out.

Example

The osasnmpd subagent starts a daemon that is called osasnmpd. To stop osasnmpd, issue the **kill** command for either the daemon or its PID:

```
# ps -ef | grep snmp

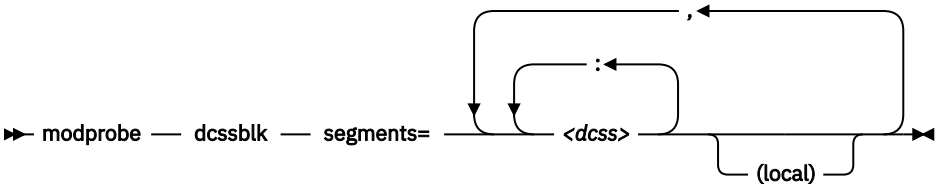
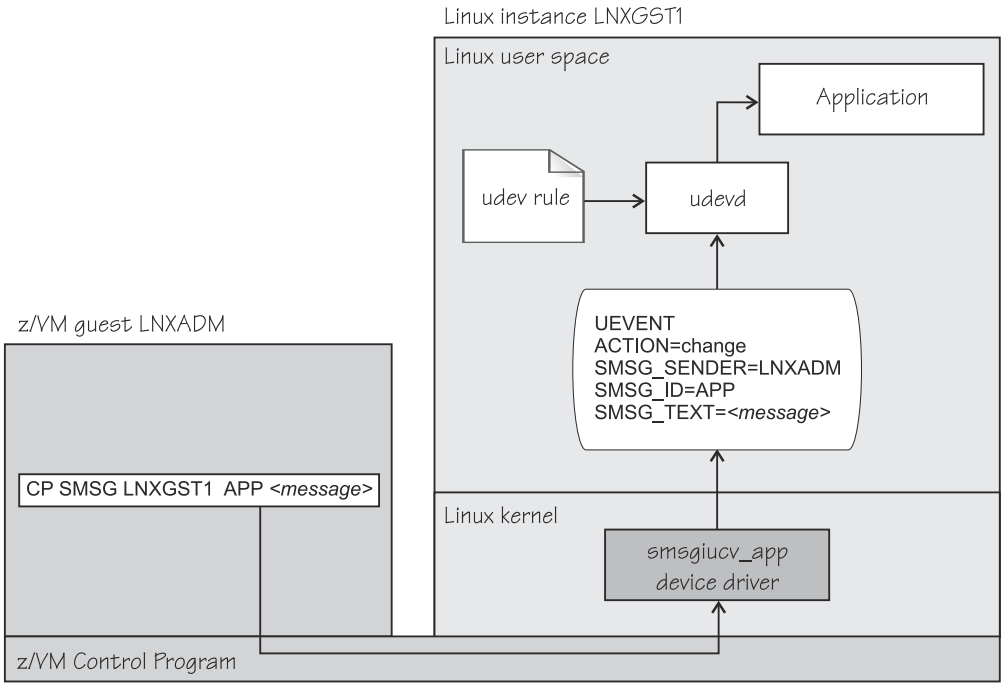
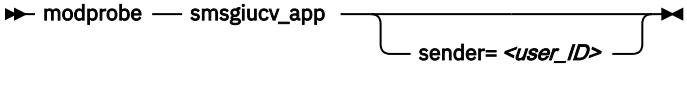
USER       PID
root       687     1  0 11:57 pts/1    00:00:00 snmpd
root       729   659  0 13:22 pts/1    00:00:00 osasnmpd
# killall osasnmpd
```

Chapter 18. LAN channel station device driver

LPAR and z/VM: The LCS device driver applies to Linux in LPAR mode and to Linux on z/VM.

The LAN channel station device driver (LCS device driver) supports Open Systems Adapters (OSA) features in non-QDIO mode.

Table 49 on page 315 shows the supported OSA-Express features.

Table 49. The LCS device driver supported OSA features

Feature	z15	z14 and z14 ZR1	z13 and z13s
OSA-Express7S	1000Base-T Ethernet	Not supported	Not supported
OSA-Express6S	1000Base-T Ethernet	1000Base-T Ethernet	Not supported
OSA-Express5S	1000Base-T Ethernet	1000Base-T Ethernet	1000Base-T Ethernet
OSA-Express4S	1000Base-T Ethernet	1000Base-T Ethernet	1000Base-T Ethernet

The LCS device driver supports automatic detection of Ethernet connections. The LCS device driver can be used for Internet Protocol, version 4 (IPv4) only.

What you should know about LCS

Interface names are assigned to LCS group devices, which map to subchannels and their corresponding device numbers and device bus-IDs.

LCS group devices

The LCS device driver requires two I/O subchannels for each LCS interface, a read subchannel and a write subchannel. The corresponding bus IDs must be configured for control unit type 3088.



Figure 80. I/O subchannel interface

The device bus-IDs that correspond to the subchannel pair are grouped as one LCS group device. The following rules apply for the device bus-IDs:

read
must be even.

write
must be the device bus-ID of the read subchannel plus one.

Setting up the LCS device driver

There are no module parameters for the LCS device driver.

You must load the lcs module before you can work with LCS devices. Load the lcs module with the **modprobe** command to ensure that any other required modules are loaded in the correct order:

```
# modprobe lcs
```

Working with LCS devices

Working with LCS devices includes tasks such as creating an LCS group device, specifying a timeout, or activating an interface.

- "Creating an LCS group device" on page 316
- "Removing an LCS group device" on page 317
- "Specifying a timeout for LCS LAN commands" on page 317
- "Setting an LCS group device online or offline" on page 318
- "Activating and deactivating an interface" on page 318
- "Recovering an LCS group device" on page 319

Most of these tasks involve writing to and reading from device attributes in sysfs. Using attributes is useful on a running system where you want to make dynamic changes. If you want to make persistent changes across IPLs, use the interface configuration files. Network configuration parameters are defined in `/etc/sysconfig/network-scripts/ifcfg-<if_name>`. See the Red Hat documentation website for an example of how to define an LCS device persistently. The website also contains a general discussion of network configuration files, see https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux.

Creating an LCS group device

Use the `group` attribute to create an LCS group device.

Before you begin

You must know the device bus-IDs that correspond to the read and write subchannel of your OSA card. The subchannel is defined in the IOCDS of your mainframe.

Procedure

To define an LCS group device, write the device bus-IDs of the subchannel pair to `/sys/bus/ccwgroup/drivers/lcs/group`.

Issue a command of this form:

```
# echo <read_device_bus_id>,<write_device_bus_id> > /sys/bus/ccwgroup/drivers/lcs/group
```

Results

The lcs device driver uses the device bus-ID of the read subchannel to create a directory for a group device:

```
/sys/bus/ccwgroup/drivers/lcs/<read_device_bus_id>
```

This directory contains a number of attributes that determine the settings of the LCS group device. The following sections describe how to use these attributes to configure an LCS group device.

Example

Assuming that 0.0.d000 is the device bus-ID that corresponds to a read subchannel:

```
# echo 0.0.d000,0.0.d001 > /sys/bus/ccwgroup/drivers/lcs/group
```

This command results in the creation of the following directories in sysfs:

- `/sys/bus/ccwgroup/drivers/lcs/0.0.d000`

- `/sys/bus/ccwgroup/devices/0.0.d000`
- `/sys/devices/lcs/0.0.d000`

Note: When the device subchannels are added, device types 3088/08 and 3088/1f can be assigned to either the CTCM or the LCS device driver.

To check which devices are assigned to which device driver, issue the following commands:

```
# ls -l /sys/bus/ccw/drivers/ctcm
# ls -l /sys/bus/ccw/drivers/lcs
```

To change a faulty assignment, use the unbind and bind attributes of the device. For example, to change the assignment for device bus-IDs `0.0.2000` and `0.0.2001` issue the following commands:

```
# echo 0.0.2000 > /sys/bus/ccw/drivers/ctcm/unbind
# echo 0.0.2000 > /sys/bus/ccw/drivers/lcs/bind
# echo 0.0.2001 > /sys/bus/ccw/drivers/ctcm/unbind
# echo 0.0.2001 > /sys/bus/ccw/drivers/lcs/bind
```

Removing an LCS group device

Use the `ungroup` attribute to remove an LCS group device.

Before you begin

The device must be set offline before you can remove it.

Procedure

To remove an LCS group device, write 1 to the `ungroup` attribute.

Issue a command of the form:

```
echo 1 > /sys/bus/ccwgroup/drivers/lcs/<device_bus_id>/ungroup
```

Example

This command removes device 0.0.d000:

```
echo 1 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/ungroup
```

Specifying a timeout for LCS LAN commands

Use the `lancmd_timeout` attribute to set a timeout for an LCS LAN command.

About this task

You can specify a timeout for the interval that the LCS device driver waits for a reply after issuing a LAN command to the LAN adapter. For older hardware, the replies can take a longer time. The default is 5 s.

Procedure

To set a timeout, issue a command of this form:

```
# echo <timeout> > /sys/bus/ccwgroup/drivers/lcs/<device_bus_id>/lancmd_timeout
```

where *<timeout>* is the timeout interval in seconds in the range 1 - 60.

Example

In this example, the timeout for a device 0.0.d000 is set to 10 s.

```
# echo 10 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/lancmd_timeout
```

Setting an LCS group device online or offline

Use the online device group attribute to set an LCS device online or offline.

About this task

Setting a device online associates it with an interface name. Setting the device offline preserves the interface name.

You must know the interface name to activate the network interface. To determine the assigned interface name, use the **lszdev --existing** command. For each online interface, the interface name is shown in the Name column.

Alternatively, for each online interface, there is a symbolic link of the form /sys/class/net/<interface_name>/device in sysfs. You can confirm that you found the correct interface name by reading the link.

Procedure

To set an LCS group device online, set the online device group attribute to 1. To set an LCS group device offline, set the online device group attribute to 0.

Issue a command of this form:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/lcs/<device_bus_id>/online
```

Example

To set an LCS device with bus ID 0.0.d000 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/online
# lszdev --existing
TYPE         ID                         ON   PERS  NAMES
...
lcs          0.0.d000:0.0.d001          yes  no    encd000
...
```

The interface name that was assigned to the LCS group device in the example is encd000. To confirm that this name is correct for the group device issue:

```
# readlink /sys/class/net/encd000/device
../../../devices/lcs/0.0.d000
```

If an error occurs when you set the device online, ensure that the physical connection from the port to the network is in place. If the error persists, note the return code from the error message and contact IBM support.

To set the device offline issue:

```
# echo 0 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/online
```

Activating and deactivating an interface

Use the **ip** command or equivalent to activate or deactivate an interface.

About this task

Before you can activate an interface, you must set the group device online and find out the interface name that is assigned by the LCS device driver. See “Setting an LCS group device online or offline” on page 318.

You activate or deactivate network devices with **ip** or an equivalent command. For details of the **ip** command, see the **ip** man page.

Examples

- This example activates an Ethernet interface:

```
# ip addr add 192.168.100.10/24 dev encf500
# ip link set dev encf500 up
```

- This example deactivates the Ethernet interface:

```
# ip link set dev encf500 down
```

- This example reactivates an interface that was already activated and subsequently deactivated:

```
# ip link set dev encf500 up
```

Recovering an LCS group device

You can use the recover attribute of an LCS group device to recover it in case of failure. For example, issue **journalctl** to find error messages that inform you of a malfunctioning device.

Procedure

Issue a command of the form:

```
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/<device_bus_id>/recover
```

Example

```
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/0.0.d100/recover
```

Chapter 19. CTCM device driver

LPAR and z/VM: The CTCM device driver applies to Linux in LPAR mode and to Linux on z/VM.

The CTCM device driver provides Channel-to-Channel (CTC) connections and CTC-based Multi-Path Channel (MPC) connections. The CTCM device driver is required by Communications Server for Linux.

Deprecated connection type: CTC connections are deprecated. Do not use for new network setups.

CTC connections are high-speed point-to-point connections between two mainframe operating system instances.

Communications Server for Linux uses MPC connections to connect Red Hat Enterprise Linux 9.1 to VTAM® on traditional mainframe operating systems.

Features

The CTCM device driver provides different kinds of CTC connections between mainframes, z/VM guests, and LPARs.

The CTCM device driver provides:

- MPC connections to VTAM on traditional mainframe operating systems.
- ESCON or FICON CTC connections (standard CTC and basic CTC) between mainframes in basic mode, LPARs or z/VM guests.

  For more information about FICON, see Redpaper *FICON CTC Implementation*, REDP-0158.
- Virtual CTCA connections between guests of the same z/VM system.
- CTC connections to other Linux instances or other mainframe operating systems.

What you should know about CTCM

The CTCM device driver assigns network interface names to CTCM group devices.

CTCM group devices

The CTCM device driver requires two I/O subchannels for each interface, a read subchannel and a write subchannel.

The CTCM device driver requires two I/O subchannels for each interface, a read subchannel and a write subchannel (see Figure 81 on page 321). The device bus-IDs that correspond to the two subchannels must be configured for control unit type 3088.



Figure 81. I/O subchannel interface

The device bus-IDs that correspond to the subchannel pair are grouped as one CTCM group device. There are no constraints on the device bus-IDs of read subchannel and write subchannel. In particular, it is possible to group non-consecutive device bus-IDs.

On the communication-peer operating system instance, read and write subchannels are reversed. That is, the write subchannel of the local interface is connected to the read subchannel of the remote interface and vice versa.

Depending on the protocol, the interfaces can be CTC interfaces or MPC interfaces. MPC interfaces are used by Communications Server for Linux and connect to peer interfaces that run under VTAM. For more information about Communications Server for Linux and on using MPC connections, go to www.ibm.com/software/network/commserver/linux.

Interface names assigned by the CTCM device driver

When a CTCM group device is set online, the CTCM device driver automatically assigns an interface name to it. The interface name depends on the protocol.

If the protocol is set to 4, you get an MPC connection and the interface names are of the form `mpc<n>`.

If the protocol is set to `0`, 1, or 3, you get a CTC connection and the interface name is of the form `ctc<n>`.

<n> is an integer that identifies the device. When the first device is set online it is assigned 0, the second is assigned 1, the third 2, and so on. The devices are counted separately for CTC and MPC.

Network connections

If your CTC connection is to a router or z/VM TCP/IP service machine, you can connect CTC interfaces to an external network.

Figure 82 on page 322 shows a CTC interface that is connected to a network.



Figure 82. Network connection

Setting up the CTCM device driver

There are no module parameters for the CTCM device driver. Load the ctcm module before you use it.

Load the ctcm module with the **modprobe** command to ensure that any other required modules are loaded:

```
# modprobe ctcm
```

Working with CTCM devices

When you work with CTCM devices you might create a CTCM group device, set the protocol, and activate an interface.

The following sections describe typical tasks that you need when you work with CTCM devices.

- “Creating a CTCM group device” on page 323
- “Removing a CTCM group device” on page 324
- “Displaying the channel type” on page 324

- "Setting the protocol" on page 324
- "Setting a device online or offline" on page 325
- "Setting the maximum buffer size" on page 326 (CTC only)
- "Activating and deactivating a CTC interface" on page 326 (CTC only)
- "Recovering a lost CTC connection" on page 328 (CTC only)

See the Communications Server for Linux documentation for information about configuring and activating MPC interfaces.

Creating a CTCM group device

Use the `group` attribute to create a CTCM group device.

Before you begin

You must know the device bus-IDs that correspond to the local read and write subchannel of your CTCM connection as defined in your IOCDS.

Procedure

To define a CTCM group device, write the device bus-IDs of the subchannel pair to `/sys/bus/ccwgroup/drivers/ctcm/group`.

Issue a command of this form:

```
# echo <read_device_bus_id>,<write_device_bus_id> > /sys/bus/ccwgroup/drivers/ctcm/group
```

Results

The CTCM device driver uses the device bus-ID of the read subchannel to create a directory for a group device:

```
/sys/bus/ccwgroup/drivers/ctcm/<read_device_bus_id>
```

This directory contains a number of attributes that determine the settings of the CTCM group device.

Example

Assuming that device bus-ID 0.0.2000 corresponds to a read subchannel:

```
# echo 0.0.2000,0.0.2001 > /sys/bus/ccwgroup/drivers/ctcm/group
```

This command results in the creation of the following directories in sysfs:

- `/sys/bus/ccwgroup/drivers/ctcm/0.0.2000`
- `/sys/bus/ccwgroup/devices/0.0.2000`
- `/sys/devices/ctcm/0.0.2000`

Note: When the device subchannels are added, device types 3088/08 and 3088/1f can be assigned to either the CTCM or the LCS device driver.

To check which devices are assigned to which device driver, issue the following commands:

```
# ls -l /sys/bus/ccw/drivers/ctcm
# ls -l /sys/bus/ccw/drivers/lcs
```

To change a faulty assignment, use the unbind and bind attributes of the device. For example, to change the assignment for device bus-IDs `0.0.2000` and `0.0.2001` issue the following commands:

```
# echo 0.0.2000 > /sys/bus/ccw/drivers/lcs/unbind
# echo 0.0.2000 > /sys/bus/ccw/drivers/ctcm/bind
# echo 0.0.2001 > /sys/bus/ccw/drivers/lcs/unbind
# echo 0.0.2001 > /sys/bus/ccw/drivers/ctcm/bind
```

Removing a CTCM group device

Use the ungroup attribute to remove a CTCM group device.

Before you begin

The device must be set offline before you can remove it.

Procedure

To remove a CTCM group device, write 1 to the ungroup attribute.

Issue a command of the form:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/ungroup
```

Example

This command removes device 0.0.2000:

```
echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/ungroup
```

Displaying the channel type

Use the type attribute to display the channel type of a CTCM group device.

Procedure

Issue a command of this form to display the channel type of a CTCM group device:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/type
```

where *<device_bus_id>* is the device bus-ID that corresponds to the CTCM read channel. Possible values are: CTC/A, ESCON, and FICON.

Example

In this example, the channel type is displayed for a CTCM group device with device bus-ID 0.0.f000:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f000/type
ESCON
```

Setting the protocol

Use the protocol attribute to set the protocol.

Before you begin

The device must be offline while you set the protocol.

About this task

The type of interface depends on the protocol. Protocol 4 results in MPC interfaces with interface names mpc*<n>*. Protocols 0, 1, or 3 result in CTC interfaces with interface names of the form ctc*<n>*.

To choose a protocol, set the protocol attribute to one of the following values:

0
This protocol provides compatibility with peers other than z/OS, for example, a z/VM TCP service machine. This value is the default.

1
This protocol provides enhanced package checking for Linux peers.

3
This protocol provides for compatibility with z/OS peers.

4
This protocol provides for MPC connections to VTAM on traditional mainframe operating systems.

Procedure

Issue a command of this form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/protocol
```

Example

In this example, the protocol is set for a CTCM group device 0.0.2000:

```
# echo 4 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/protocol
```

Setting a device online or offline

Use the `online` device group attribute to set a CTCM device online or offline.

About this task

Setting a group device online associates it with an interface name. Setting the group device offline and back online with the same protocol preserves the association with the interface name. If you change the protocol before you set the group device back online, the interface name can change as described in “Interface names assigned by the CTCM device driver” on page 322.

You must know the interface name to access the CTCM group device. To determine the assigned interface name, use the **`lszdev --existing`** command. For each online interface, the interface name is shown in the Name column. Alternatively, to determine the assigned interface name issue a command of the form:

```
# ls /sys/devices/ctcm/<device_bus_id>/net/
```

For each online interface, there is a symbolic link of the form `/sys/class/net/<interface_name>/device` in sysfs. You can confirm that you found the correct interface name by reading the link.

Procedure

To set a CTCM group device online, set the online device group attribute to `1`. To set a CTCM group device offline, set the online device group attribute to `0`.

Issue a command of this form:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/online
```

Example

To set a CTCM device with bus ID 0.0.2000 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/online
```

To determine the interface name issue:

```
# lszdev --existing
TYPE         ID                          ON   PERS  NAMES
ctcm         0.0.2000:0.0.2001           yes  no    slc2000
```

or

```
# ls /sys/devices/ctcm/0.0.2000/net/
slc2000
```

To set group device 0.0.2000 offline issue:

```
# echo 0 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/online
```

Setting the maximum buffer size

Use the `buffer` device group attribute to set a maximum buffer size for a CTCM group device.

Before you begin

- Set the maximum buffer size for CTC interfaces only. MPC interfaces automatically use the highest possible maximum buffer size.
- The device must be online when you set the buffer size.

About this task

You can set the maximum buffer size for a CTC interface. The permissible range of values depends on the MTU settings. It must be in the range *<minimum MTU + header size>* to *<maximum MTU + header size>*. The header space is typically 8 byte. The default for the maximum buffer size is 32768 byte (32 KB).

Changing the buffer size is accompanied by an MTU size change to the value *<buffer size - header size>*.

Procedure

To set the maximum buffer size, issue a command of this form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/buffer
```

where *<value>* is the number of bytes you want to set. If you specify a value outside the valid range, the command is ignored.

Example

In this example, the maximum buffer size of a CTCM group device 0.0.f000 is set to 16384 byte.

```
# echo 16384 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f000/buffer
```

Activating and deactivating a CTC interface

Use **ip** or an equivalent command to activate or deactivate an interface.

Before you begin

- Activate and deactivate a CTC interfaces only. For information about how to activate MPC interfaces, see the Communications Server for Linux documentation.
- You must know the interface name. See “Setting a device online or offline” on page 325.

About this task

Syntax for setting an IP address for a CTC interface with the ip command

Syntax for activating a CTC interface with the ip command



Where:

<interface>
is the interface name that was assigned when the CTCM group device was set online.

<ip_address>
is the IP address that you want to assign to the interface.

<peer_ip_address>
is the IP address of the remote side.

<max_transfer_unit>
is the size of the largest IP packet that might be transmitted. Be sure to use the same MTU size on both sides of the connection. The MTU must be in the range of 576 byte to 65,536 byte (64 KB).

Syntax for deactivating a CTC interface with the ip command

Where:

<interface>
is the interface name that was assigned when the CTCM group device was set online.

Procedure

- Use **ip** or an equivalent command to activate the interface.
- To deactivate an interface, issue a command of this form:

```
# ip link set dev <interface> down
```

Examples

- This example activates a CTC interface slc2000 with an IP address 10.0.51.3 for a peer with address 10.0.50.1 and an MTU of 32760.

```
# ip addr add 10.0.51.3 dev slc2000 peer 10.0.50.1
# ip link set dev slc2000 up mtu 32760
```

- This example deactivates slc2000:

```
# ip link set dev slc2000 down
```

Recovering a lost CTC connection

If one side of a CTC connection crashes, you cannot simply reconnect after a reboot. You must also deactivate the interface of the peer of the crashed side.

Before you begin

These instructions apply to CTC interfaces only.

Procedure

Proceed as follows to recover a lost CTC connection:

1. Reboot the crashed side.
2. Deactivate the interface on the peer. See “Activating and deactivating a CTC interface” on page 326.
3. Activate the interface on the crashed side and on the peer.

   For details, see “Activating and deactivating a CTC interface” on page 326.

   If the connection is between a Linux instance and a non-Linux instance, activate the interface on the Linux instance first. Otherwise, you can activate the interfaces in any order.

Results

If the CTC connection is uncoupled, you must couple it again and reconfigure the interface of both peers with the **ip** command. See “Activating and deactivating a CTC interface” on page 326.

CTCM scenarios

Typical use cases of CTC connections include connecting to a peer in a different LPAR and connecting Linux instances running as z/VM guests to each other.

This section provides some typical scenarios for CTC connections:

- “Connecting to a peer in a different LPAR” on page 328
- “Connecting Linux on z/VM to another guest of the same z/VM system ” on page 330

Connecting to a peer in a different LPAR

A Linux instance and a peer run in LPAR mode on the same or on different mainframes and are to be connected with a CTC FICON or CTC ESCON network interface.

Assumptions:

- Locally, the read and write channels have been configured for type 3088 and use device bus-IDs 0.0.f008 and 0.0.f009.
- IP address 10.0.50.4 is to be used locally and 10.0.50.5 for the peer.

Figure 83 on page 329 illustrates a CTC setup with a peer in a different LPAR.



Figure 83. CTC scenario with peer in a different LPAR

Procedure

1. Create a CTCM group device.
   Issue:

```
# echo 0.0.f008,0.0.f009 > /sys/bus/ccwgroup/drivers/ctcm/group
```

2. Confirm that the device uses CTC FICON or CTC ESCON:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/type
ESCON
```

   In this example, ESCON is used. You would proceed the same for FICON.
3. Select a protocol.
   The choice depends on the peer.

If the peer is ...	Choose ...
Linux	1
z/OS or OS/390	3
Any other operating system	0

   Assuming that the peer is Linux:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/protocol
```

4. Set the CTCM group device online and find out the assigned interface name:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/online
# ls /sys/devices/ctcm/0.0.f008/net/
slcf008
```

   In the example, the interface name is slcf008.
5. Assure that the peer interface is configured.
6. Activate the interface locally and on the peer.
   If you are connecting two Linux instances, either instance can be activated first. If the peer is not Linux, activate the interface on Linux first. To activate the local interface:

```
# ip addr add 10.0.50.4 dev slcf008 peer 10.0.50.5
# ip link set dev slcf008 up
```

Connecting Linux on z/VM to another guest of the same z/VM system

A virtual CTCA connection is to be set up between an instance of Linux on z/VM and another guest of the same z/VM system.

Assumptions:

- The guest ID of the peer is "guestp".
- A separate subnet has been obtained from the TCP/IP network administrator. The Linux instance will use IP address 10.0.100.100 and the peer will use IP address 10.0.100.101.

Figure 84 on page 330 illustrates a CTC setup with a peer in the same z/VM.



Figure 84. CTC scenario with peer in the same z/VM

Procedure

1. Define two virtual channels to your user ID.

   The channels can be defined in the z/VM user directory using directory control SPECIAL statements, for example:

   ```
   special f004 ctca
   special f005 ctca
   ```

   Alternatively, you can use the CP commands:

   ```
   define ctca as f004
   define ctca as f005
   ```

2. Assure that the peer interface is configured.
3. Connect the virtual channels.

   Assuming that the read channel on the peer corresponds to device number 0xf010 and the write channel to 0xf011 issue:

   ```
   couple f004 to guestp f011
   couple f005 to guestp f010
   ```

   Be sure that you couple the read channel to the peers write channel and vice versa.
4. From your booted Linux instance, create a CTCM group device. Issue:

   ```
   # echo 0.0.f004,0.0.f005 > /sys/bus/ccwgroup/drivers/ctcm/group
   ```

5. Confirm that the group device is a virtual CTCA device:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/type
CTC/A
```

6. Select a protocol.

   The choice depends on the peer.

If the peer is ...	Choose ...
Linux	1
z/OS or OS/390®	3
Any other operating system	0

   Assuming that the peer is Linux:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/protocol
```

7. Set the CTCM group device online and find out the assigned interface name:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/online
# ls /sys/devices/ctcm/0.0.f004/net/
slcf004
```

   In the example, the interface name is slcf004.

8. Activate the interface locally and on the peer.

   If you are connecting two Linux instances, either can be activated first. If the peer is not Linux, activate the local interface first. To activate the local interface:

```
# ip addr add 10.0.100.100 dev slcf004 peer 10.0.100.101
# ip link set dev slcf004 up
```

   Be sure that the MTU on both sides of the connection is the same. If necessary change the default MTU (see “Activating and deactivating a CTC interface” on page 326).

9. Ensure that the buffer size on both sides of the connection is the same. For the Linux side see “Setting the maximum buffer size” on page 326. If the peer is not Linux, see the operating system documentation of the peer.

Chapter 20. AF_IUCV address family support

LPAR and z/VM: The AF_IUCV address family support applies to Linux in LPAR mode and to Linux on z/VM.

The AF_IUCV address family provides an addressing mode for communications between applications that run on IBM Z.

This addressing mode can be used for connections through real HiperSockets and through the z/VM Inter-User Communication Vehicle (IUCV).

Support for AF_IUCV based connections through real HiperSockets requires Completion Queue Support.

HiperSockets devices facilitate connections between applications across LPARs within an IBM Z. In particular, an application that runs on an instance of Linux on IBM Z can communicate with:

- Itself
- Other applications that run on the same Linux instance
- An application on an instance of Linux on IBM Z in another LPAR

IUCV facilitates connections between applications across z/VM guest virtual machines within a z/VM system. In particular, an application that runs on Linux on z/VM can communicate with:

- Itself
- Other applications that run on the same Linux instance
- Applications running on other instances of Linux on z/VM, within the same z/VM system
- Applications running on a z/VM guest other than Linux, within the same z/VM system
- The z/VM control program (CP)

The AF_IUCV address family supports stream-oriented sockets (SOCK_STREAM) and connection-oriented datagram sockets (SOCK_SEQPACKET). Stream-oriented sockets can fragment data over several packets. Sockets of type SOCK_SEQPACKET always map a particular socket write or read operation to a single packet.

Features

The AF_IUCV address family provides socket connections for HiperSockets and IUCV.

For all instances of Linux on IBM Z, the AF_IUCV address family provides the following features:

- Multiple outgoing socket connections for real HiperSockets
- Multiple incoming socket connections for real HiperSockets

For instances of Linux on z/VM, the AF_IUCV address family also provides the following features:

- Multiple outgoing socket connections for IUCV
- Multiple incoming socket connections for IUCV
- Socket communication with applications that use the CMS AF_IUCV support

Setting up the AF_IUCV address family support

You must authorize your z/VM guest virtual machine and load those components that were compiled as separate modules.

There are no module parameters for the AF_IUCV address family support.

Setting up HiperSockets devices for AF_IUCV addressing

In AF_IUCV addressing mode, HiperSockets devices in layer 3 mode are identified through their `hsuid` sysfs attribute.

You set up a HiperSockets device for AF_IUCV by assigning a value to this attribute (see "Configuring a HiperSockets device for AF_IUCV addressing" on page 283).

Setting up your z/VM guest virtual machine for IUCV

You must specify suitable IUCV statements for your z/VM guest virtual machine.

For details and for general IUCV setup information for z/VM guest virtual machines, see *z/VM: CP Programming Services*, SC24-6272 and *z/VM: CP Planning and Administration*, SC24-6271.

Granting IUCV authorizations

Use the IUCV statement to grant the necessary authorizations.

IUCV ALLOW
: allows any other z/VM virtual machine to establish a communication path with this z/VM virtual machine. With this statement, no further authorization is required in the z/VM virtual machine that initiates the communication.

IUCV ANY
: allows this z/VM guest virtual machine to establish a communication path with any other z/VM guest virtual machine.

IUCV *<user ID>*
: allows this z/VM guest virtual machine to establish a communication path to the z/VM guest virtual machine with the z/VM user ID *<user ID>*.

You can specify multiple IUCV statements. To any of these IUCV statements you can append the MSGLIMIT *<limit>* parameter. *<limit>* specifies the maximum number of outstanding messages that are allowed for each connection that is authorized by the statement. If no value is specified for MSGLIMIT, AF_IUCV requests 65 535, which is the maximum that is supported by IUCV.

Setting a connection limit

Use the OPTION statement to limit the number of concurrent connections.

OPTION MAXCONN *<maxno>*
: *<maxno>* specifies the maximum number of IUCV connections that are allowed for this virtual machine. The default is 64. The maximum is 65 535.

Example

These sample statements allow any z/VM guest virtual machine to connect to your z/VM guest virtual machine with a maximum of 10 000 outstanding messages for each incoming connection. Your z/VM guest virtual machine is permitted to connect to all other z/VM guest virtual machines. The total number of connections for your z/VM guest virtual machine cannot exceed 100.

```
IUCV ALLOW MSGLIMIT 10000
IUCV ANY
OPTION MAXCONN 100
```

Loading the IUCV modules

Red Hat Enterprise Linux 9.1 loads the af_iucv module when an application requests a socket in the AF_IUCV domain.

You can also use the **modprobe** command to load the AF_IUCV address family support module af_iucv:

```
# modprobe af_iucv
```

Addressing AF_IUCV sockets in applications

To use AF_IUCV sockets in applications, you must code a special AF_IUCV sockaddr structure.

Application programmers: This information is intended for programmers who want to use connections that are based on AF_IUCV addressing in their applications.

The primary difference between AF_IUCV sockets and TCP/IP sockets is how communication partners are identified (for example, how they are named). To use the AF_IUCV support in an application, code a sockaddr structure with AF_IUCV as the socket address family and with AF_IUCV address information.

For details, see the af_iucv man page.

Chapter 21. SMC protocol support

The shared memory communication (SMC) protocol is an addition to TCP/IP and can be used transparently for shared memory communications.

The SMC protocol can be used for connections through:

- Shared Memory Communications through RDMA (SMC-R) with RoCE devices.
- Shared Memory Communications Direct (SMC-D) with ISM devices

If both variants are available for a connection, SMC-D is used.

Prerequisites

SMC connections are initiated through TCP/IP. Hence, the communication partners must be able to reach each other through TCP/IP.

An SMC connection requires both communication partners to support SMC. Unless both partners support SMC, the connection falls back to TCP/IP.

Similarly, a version 2 SMC connection requires both communication partners to support version 2. If one partner does not support version 2, the connection falls back to version 1.

The SMC-R protocol requires:

- A system with a RoCE Express adapter. See Chapter 22, “RDMA over Converged Ethernet,” on page 349
- For SMC-R version 1, the communication partners must be in the same subnet. As of RoCE Express2 using SMC-R version 2, communication partners can be in different IP subnets.

The SMC-D protocol requires:

- A system with an Internal Shared Memory (ISM) device. For more information about ISM devices, see Chapter 23, “Internal shared memory device driver,” on page 353. ISM devices are supported for Linux in LPAR mode and for Linux on z/VM.
- The communication partners must be running on the same CPC.
- ISM devices must have the same virtual channel ID (VCHID) on both peers to be usable for SMC-D version 2 communication.
- For SMC-D version 1, the communication partners must be in the same subnet. As of IBM z15 using SMC-D version 2, communication partners can be in different IP subnets.

To use SMC on Linux, a socket application must use the AF_SMC address family. For AF_SMC support in existing applications without code changes, the SMC-Tools package provides a preload library and the **smc_run** command. For more information about these tools and how to convert socket applications from AF_INET or AF_INET6 to AF_SMC, see “Setting up the SMC support” on page 338.

Features

The AF_SMC address family provides DMA communication through remote or internal shared memory. Benefits include:

- Transparency to existing TCP/IP applications with the preload library and **smc_run**.
- Low latency
- Lower CPU usage compared to native TCP/IP

Information and troubleshooting tools

Tools are available to help you retrieve information about SMC and troubleshoot.

smc-tools

The smc-tools package provides commands that help you to manage connections that use the SMC protocol.

- Use the **smcd info** and **smcr info** verify the setup and provides information on the capabilities of the hardware and Linux.
- Use the **smcd** and **smcr** commands to investigate your SMC links, link groups, and devices.

Wireshark

To help with troubleshooting, you can use the open source tool Wireshark to analyze SMC handshake traffic. The traffic is visually presented in the tool. The network packets sent during the SMC handshake are presented in human readable format with explanatory titles.

You can also use **tcpdump** to capture handshake traffic.

Setting up the SMC support

SMC traffic requires two associated network interfaces: an interface for a traditional TCP/IP connection and an interface for an SMC-capable device.

Any network interface that can reach the communication peer can provide the TCP/IP connection, including HiperSockets interfaces and interfaces of OSA-Express or RoCE Express adapters. The SMC-capable devices are ISM devices for SMC-D or PCI functions of RoCE Express adapters for SMC-R.

How to associate network interfaces for SMC connections depends on your version of SMC-D or SMC-R. Issue an **smcd info** or **smcr info** command to display the supported versions.

In the following example, both the hardware and software support v1 and v2 of SMC-D and SMC-R.

```
# smcr info
Kernel Capabilities
SMC Version: 2.0
SMC Hostname: t8345009.lnxne.boe
SMC-D Features: v1 v2
SMC-R Features: v1 v2

Hardware Capabilities
SEID: IBM-SYSZ-ISMSEID000000002E488561
ISM:  v1 v2
RoCE: v1 v2
```

For SMC-Dv2, you need an IBM z15, LinuxONE III, or later hardware system. The **smcd info** command must list v2 for the SMC-D Features and for ISM.

For SMC-Rv2, your SMC-capable network adapter must be RoCE Express2 or later. The **smcr info** command must list v2 for the SMC-R Features and for RoCE.

Setting up connections with SMC-Dv1 or SMC-Rv1

With version 1 of SMC-D or SMC-R, use physical network (PNET) IDs to associate network interfaces for TCP/IP and for ISM devices or RoCE Express PCI functions. If these interfaces have the same PNET ID, they are connected to the same physical network and can be used together for SMC.

LPAR and z/VM

For Linux in LPAR mode and for Linux on z/VM you can assign PNET IDs to OSA, HiperSockets, RoCE, and ISM devices through the IOCDS.

Figure 85 on page 339 illustrates how the IOCDS assigns the PNETID NET1 to an ISM device and a network interface for an Ethernet device. In Linux, the matching PNETID associates the ISM device with an Ethernet device.



Figure 85. PNET ID and ISM device association

As a fallback, you can also use a software PNET table that maps network interfaces to PCI functions of RoCE Express adapters. For more information about PNET tables, see the KVM information that follows.

KVM
For SMC on Linux on KVM, you need a software PNET table that maps network interfaces of TCP/IP connections to those of PCI functions of RoCE Express adapters.. Use the **smc_pnet** command to create a physical network (PNET) table with this mapping. See "smc_pnet - Create network mapping table" on page 726.

Note: z/OS does not support the RoCE Express adapter as an IP device, and therefore uses OSA adapters for the initial handshake for SMC-R connections. Linux has no such constraint.

Setting up connections with version 2 of SMC-D or SMC-R

Other than version 1, version 2 of SMC-D and SMC-R supports connections across IP subnets.

How to associate the TCP/IP network interfaces and SMC-capable devices that can reach a communication peer is different for SMC-Dv2 and SMC-Rv2.

SMC-Dv2
Other than for SMC-Dv1, SMC-Dv2 does not require PNET IDs to explicitly associate the interfaces, but PNET IDs must also not contradict the association. If set for both interfaces, the PNET ID must be the same, thus enabling the fallback to SMC-Dv1. This fallback would otherwise not be available, and is required when connecting to peers that support SMC-Dv1 only.

SMC-Rv2
Like SMC-Rv1, SMC-Rv2 requires PNET IDs to explicitly associate the interfaces.

SMC traffic is regulated through enterprise IDs (EIDs), which are assigned at the operating system level. Operating system instances that share an EID constitute a group that, with associated interfaces of TCP/IP and SMC-capable devices in place, can exchange SMC traffic. You can use EIDs to establish groups that are isolated from one another with respect to SMC. This isolation can separate operating system instances for data privacy. It can also prevent SMC-R connections between peers that are geographically too distant for efficient RDMA traffic.

EIDs apply to both SMC-Dv2 and SMC-Rv2. With SMC-D already limited to traffic within a hardware system, EIDs are useful mainly for SMC-Rv2.

An EID can be pre-defined in the hardware system or it can be user-defined.

System-defined EID
Hardware systems as of IBM z15 and LinuxONE III have unique system-defined EIDs. This EID is relevant to SMC-D traffic between operating system instances on the hardware system. Operating

system instances with the same system-defined EID run on the same hardware system and are eligible to exchange SMC-D traffic.

By default, Linux instances use the system-defined EID. With the **smcd seid** command, you can disable or enable the system-defined EID (see "smcd - Display information about SMC-D link groups and devices" on page 718).

On z/OS peers, the system-defined EID is enabled or disabled through a configuration parameter, see *z/OS Communications Server: IP Configuration Guide*.

With user-defined EIDs you can restrict SMC traffic to groups of operating system instances.

User-defined EIDs

User-defined EIDs are relevant to both SMC-D and SMC-R, and the same user-defined EIDs apply to both SMC variants.

Assign user-defined EIDs to set up groups of operating system instances that are eligible for SMC traffic within the groups, but not across groups. For SMC-R, user-defined EIDs can span multiple hardware systems.

If EIDs are used to group operating system instances that are geographical close, guests of the same z/VM system can all share an EID. Similarly, for SMC-R traffic, KVM guests on the same KVM host often have the same EID.

A Linux instance can have up to four EIDs, and so be a member of up to four groups.

You can use the **smcd ueid** command or the **smcr ueid** command to manage user-defined EIDs (see "smcr - Display information about SMC-R link groups, links and devices" on page 722 and "smcd - Display information about SMC-D link groups and devices" on page 718).

Each instance of Linux on IBM z15, LinuxONE III, or later has at least one active EID.

- You cannot disable the system-defined EID unless at least one user-defined EID is assigned.
- Deleting the last user-defined EID automatically enables the system-defined EID.

Figure 86 on page 340 shows an example with three Linux instances on an IBM Z system. For all instances, the system-defined EID is enabled. With IP connectivity and eligible ISM devices in place, all instances can exchange SMC-D traffic, across IP subnets.



Figure 86. SMC-D version 2 with system-defined EID

In Figure 87 on page 341, two of the Linux instances disabled their system-defined EID and use a matching user-defined EID instead. With this setup, only the instances with matching user-defined EIDs can exchange SMC-D traffic, Linux 1 and Linux 3 in the example.



Figure 87. SMC-D version 2 with user-defined EIDs

If Linux instances with matching user-defined EIDs are connected through RoCE Express adapters, the connection can be SMC-R instead of SMC-D.

SMC-R connections can span both IP subnets and hardware systems, as illustrated in Figure 88 on page 341. In the example, Linux 1 and Linux 4 can exchange SMC-R traffic.



Figure 88. SMC-R across IP subnets and hardware systems

Network device settings for SMC-R

On the network device that is associated with the RoCE Express PCI function that you want to use for SMC traffic, check the settings with the **ethtool** command and ensure that pause settings are turned on.

For example, if enP2s13 is the network device that is associated with the desired IB device port:

```
# ethtool -a enP2s13
Pause parameters for enP2s13:
Autonegotiate: off
RX: on
TX: on
```

RoCE Express PCI functions provide both, interfaces for SMC-R RDMA traffic and Ethernet interfaces for TCP traffic. To use a PCI function as a failover device for RDMA, the Ethernet interface must be active but not permit any traffic. The following example shows how this condition can be attained. The example uses the **ip** command. For a persistent configuration, use the network manager of your distribution.

1. Set up a link `mylnk_eth0` for an interface `eth0`

```
# ip link add dev mylnk_eth0 link eth0
```

To set up the link in the context of a VLAN, append the VLAN specifications to this command. For example, for a VLAN with ID 661, the command becomes:

```
# ip link add dev mylnk_eth0 link eth0 type vlan id 661
```

2. Assign an IP address to the link.

```
# ip addr add 10.2.1.1/16 dev mylnk_eth0
```

3. Activate the link.

```
# ip link set mylnk_eth0 up
```

4. Remove all auto-generated routes for the new link.

```
# ip route flush scope link dev mylnk_eth0
```

5. The network manager of your distribution might interpret this stale link setup as a configuration error. Prevent the network manager from reversing your settings to make the link functional. The example shows a NetworkManager command. Your distribution might use a different network manager, for example, wicked or netplan.

```
# nmcli device set eth0 managed no
```

Sysctl settings

SMC requires contiguous memory. The minimum is 16 KB, and the maximum is 512 MB. The SMC implementation selects a value as follows:

- Some socket applications define the socket send- and receive buffer sizes with a **setsockopt** call, whose upper limits are defined in `net.core.wmem_max` and `net.core.rmem_max`.
- If **setsockopt** SO_SNDBUF is not used, the socket send buffer size is taken from the value of `net.ipv4.tcp_wmem`.
- If **setsockopt** SO_RCVBUF is not used, the socket receive buffer is taken from the value of `net.ipv4.tcp_rmem`, rounded to the next higher power of 2.

Make an existing application use SMC

Use the preload library to make the unmodified socket application use SMC. Existing TCP/IP applications can benefit from the SMC protocol without recompiling, if they are invoked with the SMC preload library `libsmc_preload.so`. See the smc-tools package for the **smc_run** script (see “smc_run - Run a TCP socket program with the SMC protocol using a preloaded library” on page 730), which makes an existing TCP/IP socket program use SMC.

As an alternative to **smc_run**, you can use the LC_PRELOAD environment variable to specify the preload library with the application's start command:

```
# LD_PRELOAD=libsmc-preload.so <application_start_cmd>
```

Converting an application to use SMC

Alternatively, if you need to, you can convert an application. To convert an application from TCP/IP to SMC sockets, change the `socket()` function call from AF_INET to AF_SMC with protocol "0" and from AF_INET6 to AF_SMC with protocol "1". For example, change:

```
sd = socket(AF_INET, SOCK_STREAM, 0);
```

to:

```
sd = socket(AF_SMC, SOCK_STREAM, 0);
```

and

```
sd = socket(AF_INET6,SOCK_STREAM, 0);
```

to:

```
sd = socket(AF_SMC, SOCK_STREAM, 1);
```

Use the `sockets.h` header file from the glibc-header package. For more programming information, see the af_smc(7) man page.

Investigating PNET IDs

You can find the PNET IDs for PCIe devices and for CCW group devices in sysfs.

PCIe devices

Use the **smc_chk** command from the smc-tools package to display PNET IDs. Issue a command of the following form:

```
# smc_chk -i <interface>
```

For example:

```
# smc_chk -i enP10p0s0
PNET5
```

For more information about the **smc_chk** command, see “smc_chk - Verify SMC setups” on page 717.

The **smc_rnics** command, that is part of the smc-tools package, also shows the PNET IDs for PCIe devices.

Alternatively, you can use sysfs. The PNET IDs of PCI devices can be read, in EBCDIC format, as the value of the `util_string` attribute of the device in sysfs. If the PCIe device is connected through a RoCE adapter, the contents of the `util_string` attribute depends on the adapter:

- On RoCE Express adapters, the attribute contains two PNET IDs as fixed 16-character blocks in sequence.
- On RoCE Express2 adapters, the attribute contains a single PNET ID, because adapters have one PCI device per port.

You can use a command of the following form to read PNET IDs and convert them to ASCII:

```
# cat /sys/bus/pci/devices/<function_address>/util_string | iconv -f IBM-1047 -t ASCII
```

In the command, `/sys/bus/pci/devices/<function_address>` represents the PCI device in sysfs.

Example:

```
# cat /sys/bus/pci/devices/0000:00:00.0/util_string | iconv -f IBM-1047 -t ASCII
NET1
```

The PNET ID of the example is NET1. If there is no command output or if the output is blank, no PNET ID is assigned to the device.

Alternatively, with **smc_rnics**:

```
# smc_rnics
    FID  Power  PCI_ID        PCHID  Type           PPrt  PNET_ID           Net-Dev
--------------------------------------------------------------------------------
    8ca  1      0002:00:00.0  01c8   RoCE_Express2  0     NET1              enP2p0s0np0
    8ea  1      0003:00:00.0  01c8   RoCE_Express2  1     NET2              enP3p0s0np0
    ...
```

CCW group devices

Use the **smc_chk** command to display PNET IDs of CCW group devices. Issue a command of the following form:

```
# smc_chk -i <interface>
```

For example:

```
# smc_chk -i encb1f0
NET1
```

For more information about the **smc_chk** command, see “smc_chk - Verify SMC setups” on page 717.

The PNET ID of the example is NET1. If there is no command output or if the output is blank, no PNET ID is assigned to the device.

Alternatively, the PNET IDs of CCW group devices can be read, in EBCDIC format, as the value of the `util_string` of the corresponding channel path ID in sysfs. For adapters with multiple ports, the PNET IDs are given in sequential 16-character blocks corresponding to the ports. To find the channel path ID of a CCW group device, read its `chpid` attribute in sysfs.

Example:

```
# cat cat /sys/bus/ccwgroup/devices/0.0.b1f0/chpid
4a
```

To find the PNET IDs, issue a command of this form:

```
# cat /sys/devices/css0/chp0.<chpid>/util_string | iconv -f IBM-1047 -t ASCII
```

where *<chpid>* is the channel path ID. For example:

```
# cat /sys/devices/css0/chp0.4a/util_string | iconv -f IBM-1047 -t ASCII
NET1
```

The PNET ID of the example is NET1. If there is no command output or if the output is blank, no PNET ID is assigned to the device.

Tips

- The output of the **iconv** command does not have a trailing line break, so displayed PNET IDs are followed by a command prompt. Pipe the output to a suitable **sed** command, for example `sed 's/$/\n/'`, to display the PNET IDs on a separate line.
- Use the following command to display a list of all CCW devices and their PNET IDs:

```
# for device in `ls -1 /sys/bus/ccwgroup/devices`; do
chpid=`cat /sys/bus/ccwgroup/devices/$device/chpid | tr [A-F] [a-f]`;
pnetid="`cat /sys/devices/css0/chp0.$chpid/util_string | iconv -f IBM-1047 -t ASCII | sed 's/^/ /'`";
echo " device: $device chpid: $chpid pnetID: $pnetid";
done
```

Obtaining statistics for SMC connections

Separate statistical information is available about SMC-R and SMC-D connections. Use the **smcr stats** command to show the SMC-R statistics, and the **smcd stats** command to show the SMC-D statistics.

Command syntax



Where:

-d or --details
displays detailed statistics, see “Expanded output for details mode” on page 346.

-a or --absolute
ignores any counter resets and displays statistics beginning with smc module load.

reset
displays the current statistics and resets the counters for SMC-R or SMC-D to zero.

json
displays the current statistics in JSON format.

For command help information, enter **smcr stats help**, **smcd stats help**, or see the `smcr-stats` or `smcd-stats` man page.

Examples

- To show SMC-D statistics:

```
# smcd stats
```

- To show detailed SMC-R statistics and reset the SMC-R statistics counters:

```
# smcr -d stats reset
```

- To ignore any counter resets and show detailed SMC-R statistics since module load in JSON format:

```
# smcr -da stats json
```

Command output

The command output shows several counters with the following meanings:

Total connections handled
The total number of connections handled by the smc module. This number includes TCP fallback connections and handshake errors.

SMC connections
The number of connections that successfully entered the SMC mode.

Handshake errors
The number of connections that failed because of errors during the handshake phase, for example, because the peer stopped responding.

Avg requests per SMC conn
The average number of requests sent and received per SMC connection. This number includes special socket calls.

TCP fallback
The number of connections that fell back to TCP/IP.

Data transmitted
The amount of data sent (TX) or received (RX) in Bytes.

Total requests
The total number of individual send (TX) or receive (RX) requests handled. This number includes requests that ended with errors or did not transfer any data.

Buffer full
The number of occurrences where the respective send buffer (TX) could not contain all data to be sent, or did not contain as much data as requested in a receive call (RX).

Bufs
A histogram of buffer sizes for all connections, including buffer downgrades and buffer reuses. The histogram scale presents exact buffer sizes.

Reqs
A histogram of request sizes. The histogram scale includes upper boundaries of request sizes. Counts reflect requested send sizes for TX, and actual receive sizes for RX. Other than `Total requests`, this count omits erroneous requests and requests that do not transfer any data.

Special socket calls
Summarizes the total number of sockets calls that require special handling in SMC. The -d option categorizes these calls, see “Expanded output for details mode” on page 346.

Expanded output for details mode

With the -d option, the command output includes all counters of the regular mode, some of them with more detailed information:

SMC connections
Shows the SMC connections by SMC version, and shows separate counts for client and server.

Handshake errors | TCP fallback
Show separate counts for client and server.

Special socket calls
Shows the total number of sockets calls that require special handling in SMC and categorizes them into the following individual counters:

cork
The number of sockopt TCP_CORK enablements. This counter does not reflect the number of send requests with TCP_CORK enabled.

nodelay
The number of sockopt TCP_NODELAY enablements. This counter does not reflect the number of send requests with TCP_NODELAY enabled.

sendpage
The number of AF_SMC implementations of the sendpage() call.

splice
The number calls of the splice() system call.

urgent data
The number of send() and receive() calls with MSG_OOB set.

The counters with the following labels are shown only with the -d option:

Buffer full (remote)
The number of occurrences where the peer's receive buffer was exceeded by writing data. Requests that fill the buffer to the last bit are not included in this count.

Buffer too small
The number of occurrences where a send request was larger than the local send buffer's total capacity.

Buffer too small (remote)
The number of occurrences where a send request exceeded the total capacity of the peer's receive buffer.

Buffer downgrades
The number of occurrences where a buffer of the requested size could not be allocated for a new connection, and a smaller buffer was used.

Buffer reuses
The number of occurrences where a buffer was provided as requested for a new connection by reusing a buffer from a previous connection.

Chapter 22. RDMA over Converged Ethernet

Linux on IBM Z supports RDMA over Converged Ethernet (RoCE) in the form of RoCE Express features.

Red Hat Enterprise Linux supports RoCE features as shown in Table 50 on page 349. Note that the mapping of ports to function keys depend on the adapter hardware.

Table 50. Support for RDMA over Converged Ethernet features as of IBM z15

Feature	IBM z16	IBM z15
RoCE Express3	10 Gigabit Ethernet 25 Gigabit Ethernet	Not supported
RoCE Express2 Two adapter ports, different function IDs	10 Gigabit Ethernet 25 Gigabit Ethernet	10 Gigabit Ethernet 25 Gigabit Ethernet
RoCE Express Two adapter ports, same function ID	Not supported	10 Gigabit Ethernet

Table 51. Support for RDMA over Converged Ethernet features as of IBM z13

Feature	z14 and z14 ZR1	z13 and z13s
RoCE Express3	Not supported	Not supported
RoCE Express2 Two adapter ports, different function IDs	10 Gigabit Ethernet 25 Gigabit Ethernet	Not supported
RoCE Express Two adapter ports, same function ID	10 Gigabit Ethernet	10 Gigabit Ethernet

The RoCE support requires PCI Express support, see Chapter 33, “PCI Express support,” on page 401.

You can use a PCI function as a base for MacVTab or OpenVSwitch similarly to an OSA adapter, see “Using an HSCI interface as a base device for MacVTap or OpenVSwitch” on page 292.

More information

For more information about RoCE Express, see *Networking with RoCE Express*, SC34-7745. You can find this publication and further information about using RoCE Express with Linux on IBM Z and LinuxONE on IBM Documentation at ibm.com/docs/en/linux-on-systems?topic=configuration-roce-express.

Using a RoCE device for SMC-R

SMC-R requires RoCE devices that are associated with network devices of TCP networks through a PNET ID, for example through statements in the IOCDS.

The following figure illustrates how a RoCE device and a Ethernet device are associated by a matching PNET ID. A communication peer has a similarly associated pair of an RoCE device and Ethernet device.

With this setup, the TCP connection can switch over to an SMC-R connection over the SMC protocol. The communication peer can but need not be on the same CPC.



Figure 89. A matching PNET ID associates RoCE devices and Ethernet device

For more information about PNET IDs, see "Setting up the SMC support" on page 338.

Using SMC-R link groups

Once established, SMC connections do not fall back to regular TCP communications should an SMC-R link fail. To protect against link failure, SMC-R creates link groups for you. Link groups use multiple RoCE devices with the same PNET ID. A similar association of an Ethernet device with multiple RoCE devices on the communication peer then results in multiple, independent SMC-R links within a link group.



Figure 90. Multiple SMC-R links protect against link failure

The SMC-R connection survives failures of individual RoCE devices if at least one device remains operational on each side.

Use the **smcr** command to explore SMC-R links, link groups, and devices (see "smcr - Display information about SMC-R link groups, links and devices" on page 722).

Note: SMC-R does not work with multiple SMC-R links, if the links are used in a bonding setup.

Network interface names

RoCE devices on Red Hat Enterprise Linux 9.1 use the network interface naming described here.

Network interface names for RoCE devices can be based on the devices' user-defined identifiers (UIDs) or on their function IDs (FIDs). Which of the two naming schemes is used depends on whether UID uniqueness checking is enabled for your environment.

Read the `uid_is_unique` attribute for any PCIe device that is available to your Linux instance to find out which naming scheme applies.

Example:

```
# cat /sys/bus/pci/devices/0000:00:00.0/uid_is_unique
1
```

If the value is 1, UID uniqueness checking is enabled, and the network interface names are based on UIDs. For any other value, UID uniqueness checking is not enabled, and the network interface names are based on FIDs.

Network interface names based on UIDs

For Linux in LPAR mode, UIDs are specified in the PCIe device definition for RoCE adapters in the hardware configuration (IOCDS). UIDs are available only if supported by the hardware and if the LPAR is enabled for UID uniqueness checking.

UIDs are always checked for uniqueness in the following environments:

- For Linux on LinuxONE in DPM mode.
- For Linux on IBM Z as a KVM guest.
- For Linux as a z/VM guest.
- For Linux in LPAR mode, if the LPAR is in DPM mode.

For Linux in classical LPAR mode, UID uniqueness checking must be enabled through an LPAR setting in the IOCDS. With UID uniqueness checking enabled, UIDs are generated for any RoCE adapters for which none are assigned explicitly.

UIDs need not be unique across LPARs. For example, you can deliberately assign the same UID for the same physical RoCE device to simplify migrations between the LPARs. You can also assign the same UID to RoCE devices that connect to a specific physical or virtual LAN from different LPARs.

UID-based network interface names are of the form eno*<decimal_uid>*, where *<decimal_uid>* is the decimal representation of the hexadecimal UID. For example, for a RoCE device with UID 0010, the interface name is `eno16`.

Interface names based on function IDs

FIDs are associated with the slots at which RoCE adapters are plugged. Depending on your environment, you can specify FIDs in the IOCDs or they are generated for you. In contrast to UIDs, FIDs are unique across LPARs on the same IBM Z or LinuxONE hardware.

FID-based network interface names are of the form ens*<decimal_fid>*, where *<decimal_fid>* is the decimal representation of the hexadecimal FID. For example, for a RoCE device with FID 001A, the interface name is `ens26`.

Working with the RoCE support

Because the RoCE Express feature hardware physically consists of a Mellanox adapter, you must ensure that the following prerequisites are fulfilled before you can work with it.

Procedure

1. Ensure that PCIe support is enabled and the required PCI cards are active on your system. See "Setting up the PCIe support" on page 401 and "Using PCIe hotplug on LPAR" on page 402.
2. Use the appropriate Mellanox device driver:
   - To use TCP/IP, you need the `mlx4_core` and `mlx4_en` or `mlx5_core` module. If it is not already loaded, load it using for example, **modprobe**.
   - To also use RDMA with InfiniBand you need the `mlx4_ib` or `mlx5_ib` module. You can use SMC sockets or reliable datagram sockets (RDS).
     - For SMC, the SMC protocol support must be in place, see Chapter 21, "SMC protocol support," on page 337.
     - For RDS, you need the `rds` module and the `rds_rdma` module, see `Documentation/networking/rds.txt` in the Linux source tree and the `rds` and `rds-rdma` man pages.

   Load any modules that are already loaded, for example, with **modprobe**.
3. Activate the network interfaces.

   You need to know the network interface name, which you can find under:
   - `/sys/bus/pci/drivers/mlx4_core/<pci_slot>/net/<interface>` for RoCE Express.
   - `/sys/bus/pci/drivers/mlx5_core/<pci_slot>/net/<interface>` for RoCE Express2.

   Use the **ip** command or equivalent to activate an interface. See the `dev_port` sysfs attribute of the interface name to ensure that you are working with the correct port. Note that the numbering of network device ports start with 0, but the numbering of InfiniBand device ports start with 1. For example:

```
# cat /sys/class/infiniband/mlx4_0/ports/
1/ 2/
```

What to do next

For further information about Mellanox, see:

- http://www.mellanox.com/page/products_dyn?product_family=27&mtag=linux_driver
- http://www.mellanox.com/page/products_dyn?product_family=79&mtag=roce

Enabling debugging

The mlx4 and the mlx5 device drivers can be configured for debugging with a sysfs parameter.

Procedure

- For mlx4: Load the mlx4 module with the sysfs parameter `debug_level=1` to write debug messages to the syslog.

  Check the value of the `debug_level` parameter . If the parameter is set to 0, you can set it to 1 with the following command:

```
echo 1 > /sys/module/mlx4_core/parameters/debug_level
```

- For mlx5: Load the mlx5 module with the sysfs parameter `debug_mask=1` to write debug messages to the syslog.

  Check the value of the `debug_mask` parameter . If the parameter is set to 0, you can set it to 1 with the following command:

```
echo 1 > /sys/module/mlx5_core/parameters/debug_mask
```

Chapter 23. Internal shared memory device driver

LPAR and z/VM: The ISM device driver applies to Linux in LPAR mode and to Linux on z/VM.

The internal shared memory (ISM) device driver provides virtual PCI devices for shared memory communications direct (SMC-D).
Red Hat Enterprise Linux loads the ISM module when a device is activated. The module has no parameters.

ISM devices are defined in the IOCDS. Each ISM definition includes a physical network ID (PNET ID) to associate the ISM device with Ethernet devices.

The following figure illustrates how an ISM device and a HiperSockets device are associated by a matching PNET ID. A communication peer on the same CPC has a similarly associated pair of an ISM device and HiperSockets device. With this setup, the TCP connection can switch over to an SMC-D connection over the SMC protocol.



Figure 91. A matching PNET ID associates ISM devices and Ethernet devices

For information about how to find the PNET ID of PCI devices from your Linux instance, see "Investigating PNET IDs" on page 343.

For more information on SMC and SMC-D, see Chapter 21, "SMC protocol support," on page 337.

Use the **smcd** command to explore SMC-D link groups and devices, see "smcd - Display information about SMC-D link groups and devices" on page 718.

Listing ISM devices

Because ISM devices are PCI devices, you can list them with the **lspci** command.

Example

```
# lspci -v
0001:00:00.0 Non-VGA unclassified device: IBM Internal Shared Memory (ISM) virtual PCI device
        Physical Slot: 000002e1
        Flags: bus master, fast devsel, latency 0, IRQ 8
        Memory at 8001000000000000 (64-bit, prefetchable) [size=256T]
        Memory at 8002000000000000 (64-bit, prefetchable) [size=256]
        Capabilities: [40] MSI: Enable+ Count=1/32 Maskable- 64bit+
        Kernel driver in use: ism
        Kernel modules: ism
```

Part 5. System resources

These device drivers and features help you to manage the resources of your real or virtual hardware.

Newest version

You can find the newest version of this book at
ibm.com/docs/en/linux-on-systems?topic=linuxone-distributions

Restrictions

For prerequisites and restrictions see the IBM Z architecture specific information in the Red Hat Enterprise Linux 9.1 release notes at
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

Chapter 24. Managing CPUs

You can control the online status, check the capability, and, for LPAR mode, examine the topology of CPUs.

Use the **lscpu** and **chcpu** commands to manage CPUs. These commands are part of the util-linux package. For details, see the man pages. Alternatively, you can manage CPUs through the attributes of their entries in sysfs.

Some attributes that govern CPUs are available in sysfs under:

```
/sys/devices/system/cpu/cpu<N>
```

where *<N>* is the number of the logical CPU. Both the sysfs interface and the **lscpu** and **chcpu** commands manage CPUs through their logical representation in Linux.

You can obtain a mapping of logical CPU numbers to physical CPU addresses by issuing the **lscpu** command with the **-e** option.

Example:

```
# lscpu -e
CPU NODE DRAWER BOOK SOCKET CORE L1d:L1i:L2d:L2i ONLINE CONFIGURED POLARIZATION ADDRESS
0   1    0      0    0      0    0:0:0:0         yes    yes        horizontal   0
1   1    0      0    0      0    1:1:1:1         yes    yes        horizontal   1
2   1    0      0    0      1    2:2:2:2         yes    yes        horizontal   2
3   1    0      0    0      1    3:3:3:3         yes    yes        horizontal   3
4   1    0      0    0      2    4:4:4:4         yes    yes        horizontal   4
5   1    0      0    0      2    5:5:5:5         yes    yes        horizontal   5
6   1    0      0    0      3    6:6:6:6         yes    yes        horizontal   6
7   1    0      0    0      3    7:7:7:7         yes    yes        horizontal   7
8   0    1      1    1      4    8:8:8:8         yes    yes        horizontal   8
...
```

The logical CPU numbers are shown in the CPU column and the physical address in the ADDRESS column of the output table.

Alternatively, you can find the physical address of a CPU in the sysfs `address` attribute of a logical CPU.

Example:

```
# cat /sys/devices/system/cpu/cpu0/address
0
```

Simultaneous multithreading

Linux in LPAR mode can use the simultaneous multithreading technology on mainframes.

IBM z13 introduced the simultaneous multithreading technology to the mainframe. In Linux terminology, simultaneous multithreading is also known as SMT or Hyper-Threading.

With multithreading enabled, a single *core* on the hardware is mapped to multiple logical CPUs on Linux. Thus, multiple threads can issue instructions to a core simultaneously during each cycle.

To find out whether multithreading is enabled for a particular Linux instance, compare the number of cores with the number of threads that are available in the LPAR. You can use the **hyptop** command to obtain this information.

Simultaneous multithreading is designed to enhance performance. Whether this goal is achieved strongly depends on the available resources, the workload, and the applications that run on a particular Linux instance. Depending on these conditions, it might be advantageous to not make full use of mutithreading or to disable it completely. Use the **hyptop** command to obtain utilization data for threads while Linux runs with multithreading enabled.

You can use the `smt=` and `nosmt` kernel parameters to control multithreading. By default, Linux in LPAR mode uses multithreading if it is provided by the hardware.

CPU capability change

When the CPUs of a mainframe heat or cool, the Linux kernel generates a uevent for all affected online CPUs.

You can read the CPU capability from the Capability and, if present, Secondary Capability fields in `/proc/sysinfo`.

The capability value is an unsigned integer as defined in the system information block (SYSIB) 1.2.2 (see *z/Architecture Principles of Operation*, SA22-7832). A smaller value indicates a proportionally greater CPU capacity. Beyond that, there is no formal description of the algorithm that is used to generate this value. The value is used as an indication of the capability of the CPU relative to the capability of other CPU models.

Changing the configuration state of CPUs

A CPU on an LPAR can be in a `configured`, `standby`, or `reserved` state. You can change the state of `standby` CPUs to `configured` state and vice versa.

Before you begin

- You can change the configuration state of CPUs for Linux in LPAR mode only. For Linux on z/VM, CPUs are always in a `configured` state.
- Daemon processes like **cpuplugd** can change the state of any CPU at any time. Such changes can interfere with manual changes.

About this task

When Linux is booted, only CPUs that are in a `configured` state are brought online and used. The kernel does not detect CPUs in `reserved` state.

Procedure

Issue a command of this form to change the configuration state of a CPU:

```
# chcpu -c|-g <N>
```

where

<N>
is the number of the logical CPU.

-c
changes the configuration state of a CPU from `standby` to `configured`.

-g
changes the configuration state of a CPU from `configured` to `standby`. Only offline CPUs can be changed to the `standby` state.

Alternatively, you can write `1` to the `configure` sysfs attribute of a CPU to set its configuration state to `configured`, or `0` to change its configuration state to `standby`.

Examples:

- The following **chcpu** command changes the state of the logical CPU with number 2 from `standby` to `configured`:

```
# chcpu -c 2
```

The following command achieves the same results by writing 1 to the `configure` sysfs attribute of the CPU.

```
# echo 1 > /sys/devices/system/cpu/cpu2/configure
```

- The following **chcpu** command changes the state of the logical CPU with number 2 from `configured` to `standby`:

```
# chcpu -g 2
```

The following command achieves the same results by writing `0` to the `configure` sysfs attribute of the CPU.

```
# echo 0 > /sys/devices/system/cpu/cpu2/configure
```

Setting CPUs online or offline

Use the **chcpu** command or the `online` sysfs attribute of a logical CPU to set a CPU online or offline.

Before you begin

- Daemon processes like **cpuplugd** can change the state of any CPU at any time. Such changes can interfere with manual changes.

Procedure

1. Optional: Rescan the CPUs to ensure that Linux has a current list of configured CPUs.

   To initiate a rescan, issue the **chcpu** command with the **-r** option.

   ```
   # chcpu -r
   ```

   Alternatively, you can write 1 to `/sys/devices/system/cpu/rescan`.

   You might need a rescan for Linux on z/VM after one or more CPUs have been added to the z/VM guest virtual machine by the z/VM hypervisor. Linux in LPAR mode automatically detects newly available CPUs.

   Linux on KVM and Linux in LPAR mode automatically detect newly available CPUs.
2. Change the online state of a CPU by issuing a command of this form:

   ```
   # chcpu -e|-d <N>
   ```

   where

   <N>
   : is the number of the logical CPU.

   -e
   : sets an offline CPU online. Only CPUs that are in the configuration state `configured` can be set online. For Linux on z/VM, all CPUs are in the `configured` state.

   -d
   : sets an online CPU offline.

   Alternatively, you can write 1 to the `online` sysfs attribute of a CPU to set it online, or `0` to set it offline.

   Examples:

- The following **chcpu** commands force a CPU rescan, and then set the logical CPU with number 2 online.

```
# chcpu -r
# chcpu -e 2
```

  The following commands achieve the same results by writing 1 to the `online` sysfs attribute of the CPU.

```
# echo 1 > /sys/devices/system/cpu/rescan
# echo 1 > /sys/devices/system/cpu/cpu2/online
```

- The following **chcpu** command sets the logical CPU with number 2 offline.

```
# chcpu -d 2
```

  The following command achieves the same results by writing `0` to the `online` sysfs attribute of the CPU.

```
# echo 0 > /sys/devices/system/cpu/cpu2/online
```

Examining the CPU topology

If supported by your hardware, an interface is available that you can use to get information about the CPU topology of an LPAR.

Before you begin

Meaningful CPU topology information is available only to Linux in LPAR mode.

About this task

Use this information, for example, to optimize the Linux scheduler, which bases its decisions on which process gets scheduled to which CPU. Depending on the workload, this optimization might increase cache hits and therefore overall performance.

Note: By default CPU topology support is enabled in the Linux kernel. If it is not suitable for your workload, disable the support by specifying the kernel parameter `topology=off` in your parmfile or `zipl.conf`. See “Specifying kernel parameters” on page 25 for information about specifying kernel parameters.

The following sysfs attributes provide information about the CPU topology:

```
/sys/devices/system/cpu/cpu<N>/topology/thread_siblings
/sys/devices/system/cpu/cpu<N>/topology/core_siblings
/sys/devices/system/cpu/cpu<N>/topology/book_siblings
/sys/devices/system/cpu/cpu<N>/topology/drawer_siblings
```

where *<N>* specifies a particular logical CPU number. These attributes contain masks that specify sets of CPUs.

Because the mainframe hardware is evolving over time, the terms *drawer*, *book*, *core*, and *thread* do not necessarily correspond to fixed hardware entities. What matters for the Linux scheduler is the levels of relatedness that these terms signify, not the physical embodiment of the levels. In this context, more closely related means sharing more resources, like caches.

The `thread_siblings`, `core_siblings`, `book_siblings`, and `drawer_siblings` attribute each contain a mask that specifies the CPU and its peers at a particular level of relatedness.

1. The `thread_siblings` attribute covers the CPU and its closely related peers.

2. The `core_siblings` attribute covers all CPUs of the `thread_siblings` attribute and peers related at the core level.
3. The `book_siblings` attribute covers all CPUs of the `core_siblings` attribute and peers related at the book level.
4. The `drawer_siblings` attribute covers all CPUs of the `book_siblings` attribute and peers related at the drawer level.

If a machine reconfiguration causes the CPU topology to change, `change` uevents are created for each online CPU.

If the kernel also supports standby CPU activation and deactivation (see "Changing the configuration state of CPUs" on page 358), the masks also contains the CPUs that are in a configured, but offline state. Updating the masks after a reconfiguration might take some time.

CPU polarization

You can modify the operation of a vertical SMP environment by adjusting the SMP factor.

Before you begin

CPU polarization is relevant only to Linux in LPAR mode.

Warning: Turning on vertical CPU polarization without careful configuration can result in significant performance degradation. See Configuration note for details.

About this task

Horizontal CPU polarization means that the underlying hypervisor dispatches each virtual CPU of an LPAR for the same amount of time.

If vertical CPU polarization is active, the hypervisor dispatches certain CPUs for a longer time than others for maximum performance. For example, if a guest has three virtual CPUs, each of them with a share of 33%, then in case of vertical CPU polarization all of the processing time would be combined to a single CPU, which would run all the time, while the other two CPUs would get nearly no CPU time.

There are three types of vertical CPUs: high, medium, and low. Low CPUs hardly get any real CPU time, while high CPUs get a full real CPU. Medium CPUs get something in between.

Configuration note: Switching to vertical CPU polarization usually results in a system with different types of vertical CPUs. Running a system with different types of vertical CPUs can result in significant performance degradation. If possible, use only one type of vertical CPUs. Set all other CPUs offline and deconfigure them.

Procedure

To change the polarization, issue a command of this form:

```
# chcpu -p horizontal|vertical
```

Alternatively, you can write a `0` for horizontal polarization (the default) or a `1` for vertical polarization to `/sys/devices/system/cpu/dispatching`.

Example: The following **chcpu** command sets the polarization to `vertical`.

```
# chcpu -p vertical
```

You can achieve the same results by issuing the following command:

```
# echo 1 > /sys/devices/system/cpu/dispatching
```

What to do next

You can issue the **lscpu** command with the **-e** option to find out the polarization of your CPUs. For more detailed information for a particular CPU, read the `polarization` attribute of the CPU in sysfs.

```
# cat /sys/devices/system/cpu/cpu<N>/polarization
```

The polarization can have one of the following values:

- `horizontal` - each of the guests' virtual CPUs is dispatched for the same amount of time.
- `vertical:high` - full CPU time is allocated.
- `vertical:medium` - medium CPU time is allocated.
- `vertical:low` - very little CPU time is allocated.
- unknown - temporary value following a polarization change until the change is completed and the kernel has established the new polarization of each CPU.

Chapter 25. Memory hotplug

LPAR and z/VM: Hotplug memory can be used by Linux in LPAR mode and by Linux on z/VM.

You can dynamically increase or decrease the memory for your running Linux instance.

To make memory available as hotplug memory, you must define it to your LPAR or z/VM. Hotplug memory is supported by z/VM 5.4 with the PTF for APAR VM64524 and by later z/VM versions.

For more information about memory hotplug, see `Documentation/memory-hotplug.txt` in the Linux source tree.

What you should know about memory hotplug

Hotplug memory is represented in sysfs. After rebooting Linux, all hotplug memory might be offline.

Hotplug memory management overhead

Linux requires 64 bytes of memory to manage a 4-KB page of hotplug memory.

Use the following formula to calculate the total amount of initial memory that is consumed to manage your hotplug memory:

```
<hotplug memory> / 64
```

Example: 4.5 TB of hotplug memory consume `4.5 TB / 64 = 72 GB`.

For large amounts of hotplug memory, you might have to increase the initial memory that is available to your Linux instance. Otherwise, booting Linux might fail with a kernel panic and a message that there is not enough free memory.

How memory is represented in sysfs

Both the core memory of a Linux instance and the available hotplug memory are represented by directories in sysfs.

The memory with which Linux is started is the *core memory*. On the running Linux system, additional memory can be added as *hotplug memory*. The Linux kernel requires core memory to allocate its own data structures.

In sysfs, both the core memory of a Linux instance and the available hotplug memory are represented in form of memory blocks of equal size. Each block is represented as a directory of the form `/sys/devices/system/memory/memory<n>`, where *<n>* is an integer. You can find out the block size by reading the `/sys/devices/system/memory/block_size_bytes` attribute.

In the naming scheme, the memory blocks with the lowest address ranges are assigned the lowest integer numbers. The core memory always begins with memory0. The hotplug memory blocks follow the core memory blocks.

You can calculate where the hotplug memory begins. To find the number of core memory blocks, divide the base memory by the block size.

Example:

- With a core memory of 512 MB and a block size of 128 MB, the core memory is represented by four blocks, memory0 through memory3. Therefore, first hotplug memory block on this Linux instance is memory4.
- Another Linux instance with a core memory of 1024 MB and access to the same hotplug memory, represents this first hotplug memory block as memory8.

The hotplug memory is available to all operating system instances within the z/VM system or LPAR to which it was defined. The `state` sysfs attribute of a memory block indicates whether the block is in use by your own Linux system. The `state` attribute does not indicate whether a block is in use by another operating system instance. Attempts to add memory blocks that are already in use fail.

Memory state and reboot

On a running Linux instance, memory hotplug can change the online state of memory blocks for both hotplug memory and core memory. For core memory, the state is always preserved across boot cycles. Depending on multiple conditions, the state of hotplug memory might be reset to offline.

Booting with memory clearing

With memory clearing, an IPL or re-IPL resets all hotplug memory to offline.

Exception: The online status of hotplug memory is preserved for Linux on z/VM after a regular shutdown with a subsequent IPL from a CCW device.

Booting without memory clearing

Without memory clearing, the status of hotplug memory after an IPL or re-IPL depends on the type of IPL device:

- For CCW IPL devices the state is preserved.
- For FCP-attached IPL devices and for PCIe-attached NVMe IPL devices the state is reset to offline.

Reboot without memory clearing is the default. To force memory clearing, configure your re-IPL device with the clear option, see “Rebooting from an alternative source” on page 117.

Memory zones

The Linux kernel divides memory into memory zones. On a mainframe, three zones are used: `DMA`, `Normal`, and `Movable`.

- Memory in the `DMA` zone is below 2 GB, and some I/O operations require that memory buffers are located in this zone.
- Memory in the `Normal` zone is above 2 GB, and it can be used for all memory allocations that do not require zone DMA.
- Memory in the `Movable` zone cannot be used for arbitrary kernel allocations, but only for memory buffers that can easily be moved by the kernel, such as user memory allocations and page cache memory. Memory in the `Movable` zone can more easily be taken offline than memory in other zones.

The zones that are available to a memory block are listed in the `valid_zones` sysfs attribute. For more information, see “Adding memory” on page 366.

Setting up hotplug memory

Before you can use hotplug memory on your Linux instance, you must define this memory as hotplug memory on your physical or virtual hardware.

Defining hotplug memory to an LPAR

You use the Hardware Management Console (HMC) to define hotplug memory as *reserved storage* on an LPAR.

For information about defining reserved storage for your LPAR, see the *Processor Resource/Systems Manager Planning Guide*, SB10-7041 for your mainframe.

Defining hotplug memory to z/VM

In z/VM, you define hotplug memory as *standby storage*.

There is also *reserved storage* in z/VM, but other than reserved memory defined for an LPAR, reserved storage that is defined in z/VM is not available as hotplug memory.

Always align the z/VM guest storage with the Linux memory block size. Otherwise, memory blocks might be missing or impossible to set offline in Linux.

For information about defining standby memory for z/VM guests, see the "DEFINE STORAGE" section in *z/VM: CP Commands and Utilities Reference*, SC24-6268.

Performing memory management tasks

Typical memory management tasks include finding out the memory block size, adding memory, and removing memory.

- "Finding out the memory block size" on page 365
- "Listing the available memory blocks" on page 365
- "Adding memory" on page 366
- "Removing memory" on page 367

Finding out the memory block size

On an IBM Z mainframe, memory is provided to Linux as memory blocks of equal size.

Procedure

- Use the **lsmem** command to find out the size of your memory blocks.

  Example:

```
# lsmem
Address range                       Size (MB)  State    Removable  Device
==========================================================================
0x0000000000000000-0x000000000fffffff        256  online   no         0
0x0000000010000000-0x000000002fffffff        512  online   yes        1-2
0x0000000030000000-0x000000003fffffff        256  online   no         3
0x0000000040000000-0x000000006fffffff        768  online   yes        4-6
0x0000000070000000-0x00000000ffffffff       2304  offline  -          7-15

Memory device size  : 256 MB
Memory block size   : 256 MB
Total online memory : 1792 MB
Total offline memory: 2304 MB
```

  In the example, the block size is 256 MB.

- Alternatively, you can read `/sys/devices/system/memory/block_size_bytes`. This sysfs attribute contains the block size in byte in hexadecimal notation.

  Example:

```
# cat /sys/devices/system/memory/block_size_bytes
10000000
```

  This hexadecimal value corresponds to 256 MB.

Listing the available memory blocks

List the available memory to find out how much memory is available and which memory blocks are online.

Procedure

- Use the **lsmem** command to list your memory blocks.

  Example:

```
# lsmem -a
Address range                       Size (MB)  State    Removable  Device
==========================================================================
0x0000000000000000-0x000000000fffffff        256  online   no         0
0x0000000010000000-0x000000001fffffff        256  online   no         1
0x0000000020000000-0x000000002fffffff        256  online   no         2
0x0000000030000000-0x000000003fffffff        256  online   yes        3
0x0000000040000000-0x000000004fffffff        256  online   yes        4
0x0000000050000000-0x000000005fffffff        256  offline  -          5
0x0000000060000000-0x000000006fffffff        256  offline  -          6
0x0000000070000000-0x000000007fffffff        256  offline  -          7

Memory device size  : 256 MB
Memory block size   : 256 MB
Total online memory : 1280 MB
Total offline memory: 786 MB
```

- Alternatively, you can list the available memory blocks by listing the contents of `/sys/devices/system/memory`. Read the `state` attributes of each memory block to find out whether it is online or offline.

  Example: The following command results in an overview for all available memory blocks.

```
# grep -r --include="state" "line" /sys/devices/system/memory/
/sys/devices/system/memory/memory0/state:online
/sys/devices/system/memory/memory1/state:online
/sys/devices/system/memory/memory2/state:online
/sys/devices/system/memory/memory3/state:online
/sys/devices/system/memory/memory4/state:online
/sys/devices/system/memory/memory5/state:offline
/sys/devices/system/memory/memory6/state:offline
/sys/devices/system/memory/memory7/state:offline
```

  Note: Online blocks are in use by your Linux instance. An offline block can be free to be added to your Linux instance but it might also be in use by another Linux instance.

Adding memory

You can add memory to your Linux instance by setting unused memory blocks online.

Procedure

- Use the **chmem** command with the **-e** parameter to set memory online.

  You can specify the amount of memory you want to add with the command without specifying particular memory blocks. If there are enough eligible memory blocks to satisfy your request, the tool finds them for you and sets the most suitable blocks online.

  For information about the **chmem** command, see the man page. The **chmem** command is part of the util-linux package.

- Alternatively, you can write `online` to the sysfs `state` attribute of an unused memory block.

  Issue a command of the form:

```
# echo online_value > /sys/devices/system/memory/memory<n>/state
```

  where *online_value* is one of:

  online
  : sets the memory block online to the default zone. The default zone is the first zone listed in the `valid_zones` sysfs attribute.

online_movable
sets the memory block online to the `Movable` zone. Setting the block online fails if the `Movable` zone is not listed in the `valid_zones` sysfs attribute.

online_kernel
sets the memory block online to the first non-Movable zone listed in the `valid_zones` directory. Setting the block online fails if the `Movable` zone is the only zone listed in the `valid_zones` sysfs attribute.

<n> is an integer that identifies the memory unit.

Results

Adding the memory block fails if the memory block is already in use. The `state` attribute changes to `online` when the memory block has been added successfully.

Removing memory

You can remove memory from your Linux instance by setting memory blocks offline.

About this task

Avoid removing core memory. The Linux kernel requires core memory to allocate its own data structures.

Procedure

- Use the **chmem** command with the **-d** parameter to set memory offline.

  You can specify the amount of memory you want to remove with the command without specifying particular memory blocks. The tool finds eligible memory blocks for you and sets the most suitable blocks offline.

  For information about the **chmem** command, see the man page. The **chmem** command is part of the util-linux package.
- Alternatively, you can write `offline` to the sysfs `state` attribute of an unused memory block.

  Issue a command of the form:

```
# echo offline > /sys/devices/system/memory/memory<n>/state
```

  where *<n>* is an integer that identifies the memory unit.

Results

The hotplug memory functions first relocate memory pages to free the memory block and then remove it. The `state` attribute changes to offline when the memory block has been removed successfully.

The memory block is not removed if it cannot be freed completely.

Chapter 26. Huge-page support

Note: Across the IT industry, *huge pages* and *large pages* are used synonymously for memory pages that exceed 4 KB. In keeping with the more commonly used term in the context of Linux, this publication uses *huge pages*.

Huge-page support entails support for the Linux hugetlbfs file system.

The huge-page support virtual file system is backed by larger memory pages than the usual 4 K pages; for IBM Z the hardware page size is 1 MB.

To check whether 1 MB huge pages are supported in your environment, issue the command:

```
# grep edat /proc/cpuinfo
features        : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te
```

An output line that lists `edat` as a feature indicates 1 MB huge-page support.

Applications that use huge-page memory save a considerable amount of page table memory. Another benefit from the support might be an acceleration in the address translation and overall memory access speed.

As of version 7, Red Hat Enterprise Linux also supports libhugetlbfs linking. For more information, see the libhugetlbfs package, `libhugetlbfs-<version>.s390x.rpm`, and the how-to document that is included in the package.

As of version 7, Red Hat Enterprise Linux supports transparent hugepages. For more information, see `Documentation/vm/transhuge.txt` in the Linux source tree.

As of zEC12, you can also configure 2 GB huge pages if Linux is running on an LPAR or as a KVM guest. There is no flag that indicates 2GB support; the support is always there as of zEC12. See “Pre-allocating 2 GB huge pages” on page 370.

Setting up hugetlbfs huge-page support

You configure hugetlbfs huge-page support by adding parameters to the kernel parameter line.

With huge-page support built into the kernel, you can use 1 MB huge pages without further configuration. Preallocate huge pages through kernel parameters to reserve continuous large blocks of memory and so assure that a sufficient number of huge pages is available when required.

Huge-page support kernel parameter syntax



where:

default_hugepagesz=<*size*>
specifies the default page size in byte. You can use suffixes K, M, and G to specify KB, MB, and GB. The default value is 1 MB.

hugepages=<*number*>
is the number of huge pages to be allocated at boot time.

hugepagesz=<*size*>
specifies the page size in byte. You can use suffixes K, M, and G to specify KB, MB, and GB.

If your Linux instance is a KVM host and you want to back guest memory with huge pages, you must enable this option through a module parameter when you load the kvm module, or through a kernel parameter, see Chapter 47, “Setting up Red Hat Enterprise Linux 9.1 as a KVM host,” on page 475.

Note: If you specify more pages than available, Linux reserves as many as possible. As a likely result, too few general pages remain for the boot process, and your system stops with an out-of-memory error.

Pre-allocating 2 GB huge pages

If Linux is running in an LPAR or as a KVM guest, you can use 2 GB huge pages.

Before you can use 2 GB huge pages, you must pre-allocate them to the kernel page pool. To pre-allocate 2 GB pages, precede the **hugepages=** parameter with the page size selection parameter, **hugepagesz=2G**.

Tip: Memory quickly becomes fragmented after booting, and new 2 GB huge pages cannot be allocated. Use kernel boot parameters to allocate 2 GB huge pages to avoid the memory fragmentation problem.

To pre-allocate a number of pages of 2 GB size and also set the default size to 2 GB:

```
default_hugepagesz=2G hugepagesz=2G hugepages=<number>
```

Setting up multiple huge-page pools

You can allocate multiple huge-page pools and use them simultaneously. To allocate multiple huge-page pools, specify the `hugepagesz=` parameter several times, each time followed by a corresponding `hugepages=` parameter.

For example, to specify two pools, one with 1 MB pages and one with 2 GB pages, specify:

```
hugepagesz=1M hugepages=8 hugepagesz=2G hugepages=2
```

This creates a sysfs attribute for each pool, `/sys/kernel/mm/hugepages/hugepages-<size>kB/nr_hugepages`, where *<size>* is the page size in KB. For the example given, the following attributes are created:

```
/sys/kernel/mm/hugepages/hugepages-1024kB/nr_hugepages
/sys/kernel/mm/hugepages/hugepages-2097152kB/nr_hugepages
```

Huge pages and hotplug memory

Hotplug memory that is added to a running Linux instance is movable and can be allocated to movable resources only.

By default, huge pages are not movable and cannot be allocated from movable memory. You can enable allocation from movable memory with the sysctl setting `hugepages_treat_as_movable`.

To enable allocation of huge pages from movable hotplug memory, issue:

```
# echo 1 > /proc/sys/vm/hugepages_treat_as_movable
```

Although this setting makes huge pages eligible for allocation through movable memory, it does not make huge pages movable. As a result, the allocated hotplug memory cannot be set offline until all huge pages are released from that memory.

To disable allocation of huge pages from movable hotplug memory, issue:

```
# echo 0 > /proc/sys/vm/hugepages_treat_as_movable
```

Working with hugetlbfs huge-page support

Typical tasks for working with hugetlbfs huge-page support include reading the current number of huge pages, changing the number of huge pages, and display information about available huge pages.

About this task

The huge-page memory can be used through `mmap()` or SysV shared memory system calls, more detailed information can be found in the Linux kernel source tree under `Documentation/vm/hugetlbpage.txt`, including implementation examples.

Your database product might support huge-page memory. See your database documentation to find out if and how it can be configured to use huge-page memory.

Depending on your version of Java™, you might require specific options to make a Java program use the huge-page feature. For IBM SDK, Java Technology Edition 7, specify the **-Xlp** option. If you use the SysV shared memory interface, which includes **java -Xlp**, you must adjust the shared memory allocation limits to match the workload requirements. Use the following sysctl attributes:

/proc/sys/kernel/shmall
: Defines the global maximum amount of shared memory for all processes, specified in number of 4 KB pages.

/proc/sys/kernel/shmmax
: Defines the maximum amount of shared memory per process, specified in number of bytes.

For example, the following commands would set both limits to 20 GB:

```
# echo 5242880 > /proc/sys/kernel/shmall
# echo 21474836480 > /proc/sys/kernel/shmmax
```

Procedure

- Specify the `hugepages=` kernel parameter with the number of huge pages to be allocated at boot time. To read the current number of default huge pages, issue:

  ```
  # cat /proc/sys/vm/nr_hugepages
  ```

- To change the number of default-sized huge pages dynamically during runtime, write to procfs:

  ```
  # echo 20 > /proc/sys/vm/nr_hugepages
  ```

  If there is not enough contiguous memory available to fulfill the request, the maximum number of huge pages are reserved.
- To obtain information about the number of huge pages currently available and the huge-page size, issue:

  ```
  # cat /proc/meminfo

  ...
  HugePages_Total: 20
  HugePages_Free: 14
  HugePages_Rsvd: 0
  HugePages_Surp: 0
  ...
  Hugepagesize: 1024 KB
  ...
  ```

- To adjust characteristics of a huge-page pool, when more than one pool exists, use the sysfs attributes of the pool.

  These can be found under

```
/sys/kernel/mm/hugepages/hugepages-<size>/nr_hugepages
```

Where *<size>* is the page size in KB.

Example

To allocate 2 GB huge pages:

1. Specify 2 GB huge pages and pre-allocate them to the page pool at boot time. Use the following kernel boot parameters:

```
default_hugepagesz=2G hugepagesz=2G hugepages=4
```

2. After booting, read `/proc/meminfo` to see information about the amount of huge pages currently available and the huge-page size:

```
cat /proc/meminfo
...
HugePages_Total: 4
HugePages_Free: 4
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: 2097152 kB
...
```

Chapter 27. S/390 hypervisor file system

The S/390® hypervisor file system (hypfs) provides a mechanism to access LPAR and z/VM hypervisor data.

Directory structure

When the hypfs file system is mounted, the accounting information is retrieved and a file system tree is created. The tree contains a full set of attribute files with the hypervisor information.

By convention, the mount point for the hypervisor file system is `/sys/hypervisor/s390`.

LPAR directories and attributes

There are hypfs directories and attributes with hypervisor information for Linux in LPAR mode.

Figure 92 on page 373 illustrates the file system tree that is created for LPAR.



Figure 92. The hypervisor file system for LPAR

update
Write-only file to trigger an update of all attributes.

cpus/
Directory for all physical cores.

cpus/<*core ID*>
Directory for one physical core. *<core_ ID>* is the logical (decimal) core number.

type
Type name of physical core, such as CP or IFL.

mgmtime
Physical-LPAR-management time in microseconds (LPAR overhead).

hyp/
Directory for hypervisor information.

hyp/type
Type of hypervisor (LPAR hypervisor).

systems/
Directory for all LPARs.

systems/*<lpar name>*/
Directory for one LPAR.

systems/*<lpar name>*/cpus/*<core_ID>*/
Directory for the virtual cores for one LPAR. The *<core_ID>* is the logical (decimal) core number.

type
Type of the logical core, such as CP or IFL.

mgmtime
LPAR-management time. Accumulated number of microseconds during which a physical core was assigned to the logical core and the core time was consumed by the hypervisor and was not provided to the LPAR (LPAR overhead).

cputime
Accumulated number of microseconds during which a physical core was assigned to the logical core and the core time was consumed by the LPAR.

onlinetime
Accumulated number of microseconds during which the logical core has been online.

Note: For LPARs with multithreading enabled, the entities in the cpus directories represent hardware cores, not threads.

Note: For older machines, the `onlinetime` attribute might be missing. Generally, it is advantageous for applications to tolerate missing attributes or new attributes that are added to the file system. To check the content of the files, you can use tools such as **`cat`** or **`less`**.

z/VM directories and attributes

There are hypfs directories and attributes with hypervisor information for Linux on z/VM.

update
Write-only file to trigger an update of all attributes.

cpus/
Directory for all physical CPUs.

cpus/count
Total current CPUs.

hyp/
Directory for hypervisor information.

hyp/type
Type of hypervisor (z/VM hypervisor).

systems/
Directory for all z/VM guest virtual machines.

systems/*<guest name>*/
Directory for one z/VM guest virtual machine.

systems/*<guest name>*/onlinetime_us
Time in microseconds that the guest virtual machine has been logged on.

systems/<*guest name*>/cpus/
Directory for the virtual CPUs for one guest virtual machine.

capped
Flag that shows whether CPU capping is on for the guest virtual machine (0 = off, 1 = soft, 2 = hard).

count
Total current virtual CPUs in the guest virtual machine.

cputime_us
Number of microseconds where the guest virtual machine CPU was running on a physical CPU.

dedicated
Flag that shows if the guest virtual machine has at least one dedicated CPU (0 = no, 1 = yes).

weight_cur
Current share of guest virtual machine (1-10000); 0 for ABSOLUTE SHARE guests.

weight_max
Maximum share of guest virtual machine (1-10000); 0 for ABSOLUTE SHARE guests.

weight_min
Number of operating CPUs. Do not be confused by the attribute name, which suggests a different meaning.

systems/<*guest name*>/samples/
Directory for sample information for one guest virtual machine.

cpu_delay
Number of CPU delay samples that are attributed to the guest virtual machine.

cpu_using
Number of CPU using samples attributed to the guest virtual machine.

idle
Number of idle samples attributed to the guest virtual machine.

mem_delay
Number of memory delay samples that are attributed to the guest virtual machine.

other
Number of other samples attributed to the guest virtual machine.

total
Number of total samples attributed to the guest virtual machine.

systems/<*guest name*>/mem/
Directory for memory information for one guest virtual machine.

max_KiB
Maximum memory in KiB (1024 bytes).

min_KiB
Minimum memory in KiB (1024 bytes).

share_KiB
Guest estimated core working set size in KiB (1024 bytes).

used_KiB
Resident memory in KiB (1024 bytes).

To check the content of the files, you can use tools such as **cat** or **less**.

Setting up the S/390 hypervisor file system

To use the file system, it must be mounted. You can mount the file system with the mount command or with an entry in `/etc/fstab`.

To mount the file system manually, issue the following command:

```
# mount none -t s390_hypfs <mount point>
```

where *<mount point>* is where you want the file system mounted. Preferably, use `/sys/hypervisor/s390`.

To mount hypfs by using `/etc/fstab`, add the following line:

```
none <mount point> s390_hypfs defaults 0 0
```

If your z/VM system does not support DIAG 2fc, the s390_hypfs is not activated and it is not possible to mount the file system. Instead, an error message like this is issued:

```
mount: unknown filesystem type 's390_hypfs'
```

To get data for all z/VM guests, privilege class B is required for the guest where hypfs is mounted. For non-class B guests, data is provided only for the local guest.

To get data for all LPARs, select the **Global performance data control** check box in the HMC or SE security menu of the LPAR activation profile. Otherwise, data is provided only for the local LPAR.

Working with the S/390 hypervisor file system

Typical tasks that you must perform when working with the S/390 hypervisor file system include defining access permissions and updating hypfs information.

- "Defining access permissions" on page 376
- "Updating hypfs information" on page 377

Defining access permissions

The root user usually has access to the hypfs file system. It is possible to explicitly define access permissions.

About this task

If no mount options are specified, the files and directories of the file system get the uid and gid of the user who mounted the file system (usually root). It is possible to explicitly define uid and gid by using the mount options `uid=<number>` and `gid=<number>`.

Example

You can define uid=1000 and gid=2000 with the following mount command:

```
# mount none -t s390_hypfs -o "uid=1000,gid=2000" <mount point>
```

Alternatively, you can add the following line to the `/etc/fstab` file:

```
none <mount point> s390_hypfs uid=1000,gid=2000 0 0
```

The first mount defines uid and gid. Subsequent mounts automatically have the same uid and gid setting as the first one.

The permissions for directories and files are as follows:

- Update file: 0220 (`--w--w----`)
- Regular files: 0440 (`-r--r-----`)
- Directories: 0550 (`dr-xr-x---`)

Updating hypfs information

You trigger the update process by writing something into the `update` file at the top-level hypfs directory.

Procedure

With hypfs mounted at `/sys/hypervisor/s390`, you can trigger the update process by issuing the following command:

```
# echo 1 > /sys/hypervisor/s390/update
```

During the update, the entire directory structure is deleted and rebuilt. If a file was open before the update, subsequent reads return the old data until the file is opened again. Within 1 second only one update can be done If multiple updates are triggered within a second, only the first update is performed and subsequent write system calls return `-1` and errno is set to EBUSY.

Applications can use the following procedure to ensure consistent data:

1. Read modification time through `stat(2)` from the `update` attribute.
2. If data is too old, write to the `update` attribute and start again with step 1.
3. Read data from file system.
4. Read modification time of the `update` attribute again and compare it with first timestamp. If the timestamps do not match, return to step 2.

Chapter 28. TOD clock synchronization

Your Linux instance might be part of an extended remote copy (XRC) setup that requires synchronization of the Linux time-of-day (TOD) clock with a Coordinated Timing Network (CTN).

Linux in LPAR mode supports system time protocol (STP) based TOD synchronization. For information about STP, see

```
www.ibm.com/systems/z/advantages/pso/stp.html
```

Use the **lsstp** command to display the STP configuration for your Linux instance (see "lsstp - Show STP configuration information" on page 680).

Attention: To avoid hanging I/O operations on XRC-enabled DASD, be sure that a reliable timing signal is available before enabling clock synchronization.

STP support is included in Red Hat Enterprise Linux.

Note: STP synchronizes leap seconds with a better resolution than Network Time Protocol (NTP). With STP enabled, do not use NTP daemons like `chrony` or `ntpd`.

How STP synchronization works

With STP enabled at boot time, STP synchronizes the TOD clock of a Linux instance with the STP timing network during the boot process. STP then steers the TOD clock to keep it in sync with the network. This synchronization is driven by STP, without active participation of the Linux kernel. You cannot enable STP for KVM guests, but KVM hosts pass their synchronized TODs on to their guests.

In contrast, the Linux kernel takes an active role if the TOD clock gets out-of-sync with the timing network. An out-of-sync situation usually occurs when STP is enabled on a running Linux instance (see "Enabling and disabling clock synchronization" on page 379). To bring the TOD clock back into sync, STP notifies the Linux kernel through a *sync check*. The TOD clock then leaps to the corrected time. Linux now shields applications from inconsistent time stamps by gradually steering the values returned by `gettimeofday()` towards the corrected TOD. Such corrections do not feed through to KVM guests, which remain out-of-sync with their host and with the timing network.

Enabling clock synchronization when booting

Use the `stp=` kernel parameter to enable clock synchronization when booting.

stp syntax



By default, clock synchronization is not enabled.

The kernel parameter specifies the initial synchronization setting. On a running Linux instance, you can change these settings through attributes in sysfs (see "Enabling and disabling clock synchronization" on page 379).

Enabling and disabling clock synchronization

Use the STP sysfs attribute `online` to enable or disable clock synchronization.

Procedure

To enable clock synchronization, set `/sys/devices/system/stp/online` to 1. To disable clock synchronization, set this attribute to 0.

Example

To disable clock synchronization, enter:

```
# echo 0 > /sys/devices/system/stp/online
```

Leap second handling

Through STP, Linux on IBM Z can process leap seconds from a coordinated time network (CTN) and adjust the TOD clock accordingly.

STP can schedule leap second insertions or deletions for your Linux instance. With one or more leap seconds scheduled, the Linux kernel checks, at regular intervals, whether the day for a leap second adjustment is reached.

STP schedules leap second adjustments for the end of day according to UTC.

Leap second deletion
A second is deleted at 23:59:59, that is, 23:59:58 is followed by 00:00:00.

Leap second insertion
A second is inserted at 23:59:59, that is, 23:59:59 is followed by 23:59:60.

Use the **`lsstp`** command to display information about scheduled leap seconds for your Linux instance (see “lsstp - Show STP configuration information” on page 680).

Note: Do not run an NTP daemon like chrony or ntpd with STP enabled. NTP daemons can interfere with leap second handling through STP.

Chapter 29. Identifying the IBM Z hardware

In installations with several IBM Z mainframes, you might need to identify the particular hardware system on which a Linux instance is running.

On Linux in LPAR mode, two attributes in `/sys/firmware/ocf` can help you to identify the hardware.

cpc_name
: contains the name that is assigned to the central processor complex (CPC). This name identifies the mainframe system on a Hardware Management Console (HMC).

hmc_network
: contains the name of the HMC network to which the mainframe system is connected.

The two attributes contain the empty string if the Linux instance runs as a guest of a hypervisor that does not support the operations command facility (OCF) communication parameters interface.

Use the **cat** command to read these attributes.

Example:

```
# cat /sys/firmware/ocf/cpc_name
Z05
# cat /sys/firmware/ocf/hmc_network
SNA00
```

Chapter 30. HMC media device driver

LPAR and z/VM: The HMC media device driver applies to Linux in LPAR mode and to Linux on z/VM.

You use the HMC media device driver to access files on removable media at a system that runs the Hardware Management Console (HMC).

Before you begin: You must log in to the HMC on the system with the removable media and assign the media to the LPAR.

The HMC media device driver supports the following removable media:

- A DVD in the DVD drive of the HMC system
- A CD in the DVD drive of the HMC system
- USB-attached storage that is plugged into the HMC system

The most commonly used removable media at the HMC is a DVD.

The HMC media device driver uses the `/dev/hmcdrv` device node to support these capabilities:

- List the media contents with the `lshmc` command (see "lshmc - List media contents in the HMC media drive" on page 665).
- Mount the media contents as a file system with the `hmcdrvfs` command (see "hmcdrvfs - Mount a FUSE file system for remote access to media in the HMC media drive" on page 639).

Installers on suitably prepared installation DVDs can use these capabilities to install Linux in an LPAR.

Setting up the HMC media device driver

You can set the cache size for the HMC media device driver.

Kernel parameters

If the HMC media device driver has been compiled into the kernel, you set the cache size by adding the `hmcdrv.cachesize=` parameter to the kernel parameter line.

hmcdrv kernel parameter syntax



where *<size>* is the cache size in bytes. The specification must be a multiple of 2048. You can use the suffixes K for kilobytes, M for megabytes, or G for gigabytes. Specify `0` to not cache any media content. By default, the cache size is 534288 bytes (0.5 MB).

Loading the `hmcdrv` module creates a device node at `/dev/hmcdrv`.

Example

The following specifications are equivalent:

```
hmcdrv.cachesize=153600
```

```
hmcdrv.cachesize=150K
```

Module parameters

If the HMC media device driver has been built as a separate module, `hmcdrv`, you set the cache size through the `cachesize=` module parameter.

Before you can work with the HMC media device driver and with the dependent **`lshmc`** and **`hmcdrvfs`** commands, you must load the `hmcdrv` kernel module.

hmcdrv module parameter syntax



where *<size>* is the cache size in bytes. The specification must be a multiple of 2048. You can use the suffixes K for kilobytes, M for megabytes, or G for gigabytes. Specify `0` to not cache any media content. By default, the cache size is 534288 bytes (0.5 MB).

Loading the `hmcdrv` module creates a device node at `/dev/hmcdrv`.

Example

The following specifications are equivalent:

```
# modprobe hmcdrv cachesize=153600
```

```
# modprobe hmcdrv cachesize=150K
```

Working with the HMC media

You can list files on media that are inserted into the HMC system and you can mount the media content on the Linux file system.

- "Assigning the removable media of the HMC to an LPAR" on page 384
- "Listing files on the removable media at the HMC" on page 385
- "Mounting the content of the removable media at the HMC" on page 385

Assigning the removable media of the HMC to an LPAR

Use the HMC to assign the removable media to the LPAR where your Linux instance runs.

Before you begin

- You need access to the HMC, and you must be authorized to use the **Access Removable Media** task for the LPAR to which you want to assign the media.
- For Linux on z/VM, the z/VM guest virtual machine must have at least privilege class B.
- For Linux in LPAR mode, the LPAR activation profile must allow issuing SCLP requests.

About this task

You can list files on the removable media at the HMC without having to first mount the contents on the Linux file system.

Procedure

1. Insert the removable media into the HMC system.

2. Use the **Access Removable Media** task on your HMC to assign the removable media to the LPAR where your Linux instance runs.

   For Linux on z/VM, this is the LPAR where the z/VM hypervisor runs that provides the guest virtual machine to your Linux instance.

   For details, see the HMC documentation for the HMC at your installation.

Results

You can now access the removable media from your Linux instance.

Listing files on the removable media at the HMC

Use the **lshmc** command to list files on the removable media at the HMC.

Before you begin

Your Linux instance must have access to the removable media at the HMC (see "Assigning the removable media of the HMC to an LPAR" on page 384).

About this task

You can list files on the removable media at the HMC without having to first mount the contents on the Linux file system.

Procedure

Issue a command of this form:

```
# lshmc <filepath>
```

where *<filepath>* is an optional specification for a particular path and file. Path specifications are interpreted as relative to the root directory of the removable media. You can use the asterisk (*) and question mark (?) as wildcards. If you omit *<filepath>*, all files in the root directory of the media are listed.

Example: The following command lists all `.html` files in the `www` subdirectory of the media.

```
# lshmc www/*.html
```

For more information about the **lshmc** command, see "lshmc - List media contents in the HMC media drive" on page 665.

Mounting the content of the removable media at the HMC

Use the **hmcdrvfs** command to mount the content of the removable media at the HMC.

Before you begin

Your Linux instance must have access to the removable media of the HMC (see "Assigning the removable media of the HMC to an LPAR" on page 384).

About this task

You can mount the content of the removable media at the HMC in read-only mode on the Linux file system.

Procedure

1. Optional: Confirm that your are accessing the intended content by issuing the **lshmc** command.
2. Mount the media content by issuing a command of this form:

```
# hmcdrvfs <mountpoint>
```

where *<mountpoint>* is the mount point on the Linux file system.

Example: The following command mounts the media content at `/mnt/hmc`:

```
# hmcdrvfs /mnt/hmc
```

Results

You can now access the files on the media in read-only mode through the Linux file system.

What to do next

When you no longer need access to the media content, unmount the media with the **fusermount** command.

Chapter 31. Data compression with the Integrated Accelerator for zEDC

Hardware dependency: The Integrated Accelerator for zEnterprise Data Compression (zEDC) is available on IBM Z and LinuxONE hardware as of z15 and LinuxONE III.

The Integrated Accelerator for zEDC replaces hardware-acceleration through zEDC Express as available for earlier hardware, see Chapter 32, “Data compression with GenWQE and zEDC Express,” on page 393.

The Integrated Accelerator for zEDC provides on-chip hardware-acceleration for data compression and decompression.

The prerequisites for using the Integrated Accelerator for zEDC depend on your virtualization environment.

Linux in LPAR mode
The on-chip accelerator is always available if the hardware provides it.

Linux as a guest of KVM or z/VM
The hypervisor must run on hardware that provides the on-chip accelerator, and it must support the z15 CPU model and provide it to the guest..

Linux container
The on-chip accelerator must be available to the Linux instance that runs the container. The requirements for the image are the same as for the user space of any Linux instance.

Tip: Read `/proc/cpuinfo`. If the features line includes `dflt`, your real or virtual hardware provides the on-chip accelerator.

For technical resources related to the Integrated Accelerator for zEDC, see www.ibm.com/support/z-content-solutions/compression.

Features

Acceleration with the on-chip Integrated Accelerator for zEDC is available to applications that use zlib or gzip in user space and to the kernel zlib.

Acceleration for applications in user space

Applications can use the on-chip accelerator through zlib and gzip. Red Hat Enterprise Linux 9.1 includes the required versions.

For Linux containers, the image must contain the required versions of zlib and gzip. You can search the zlib and gzip binaries for "DFLTCC" to verify that you have the required versions, as in the following example:

```
# grep DFLTCC /usr/lib64/libz.so
Binary file /usr/lib64/libz.so matches
# grep DFLTCC /usr/bin/gzip
Binary file /usr/bin/gzip matches
```

Acceleration for Java workloads

Support for Java workloads depends on your Java platform implementation.

Java implementations that use the system zlib, for example OpenJDK, support the on-chip accelerator if the system zlib supports it.

The IBM SDK for Java includes a zlib library, so its support of the on-chip accelerator is independent of the system zlib. As of IBM SDK for Java 8 SR6, the included zlib supports the on-chip accelerator.

Acceleration for the kernel

The kernel zlib can use the on-chip accelerator.

Compression levels and defaults

The *compression level* is a measure of the compression quality. It is expressed as an integer in the range 1 - 9.

Compression quality and performance are conflicting goals. Compression level 1 provides best performance at the expense of quality. Compression level 9 provides the best quality at the expense of performance. The compression level that is provided by the Integrated Accelerator for zEDC is approximately equivalent to level 1.

Acceleration defaults

The following defaults apply to both on-chip acceleration for gzip and zlib in user space and for the kernel zlib:

- By default, decompression is accelerated.
- By default, compression is accelerated only if compression level 1 is requested.

Expanding the scope of compression acceleration

Configure software with a configurable compression level to request level 1 to enable on-chip compression.

For other types of software you must configure the on-chip accelerator.

- Software that hardcodes a compression level other than 1.
- Software that neither requests a particular level nor provides an option to configure a level. Such software requests level 6 by default.

If level 1 compression is acceptable for your purposes, use overrides to apply on-chip compression to any requested compression level:

- For applications in user space, including Java workloads, see “Overrides for applications” on page 389.
- For the kernel, see “Overrides for the kernel zlib” on page 391.

Confirming that the on-chip accelerator is used

Expect a significant performance gain when using the Integrated Accelerator for zEDC for data compression and decompression workloads, especially when processing large files.

Before you begin

Ensure that your workload is configured to request compression level 1. For software that is hardcoded to request a level other than 1, use the techniques that are described in “Overriding the defaults” on page 389 to force compression with the on-chip accelerator.

Procedure

- Confirm by comparison.

  Run the same workload twice: once with the on-chip accelerator enabled and once with the on-chip accelerator off. Compare the results to assess the effect of the on-chip accelerator.

  By default, the on-chip accelerator is enabled for workloads in both user space and the kernel. Use the applicable control to turn off the on-chip accelerator for the reference run:

  User space
  Set the environment variable DFLTCC to 0.

Kernel
Restart Linux with the kernel parameter setting `dfltcc=off`.

- Confirm through hardware counters.

  Evaluate hardware counters to directly confirm that the on-chip accelerator is active. For example, you can evaluate the counters with the following symbolic names:

  DFLT_ACCESS
  Cycles CPU spent obtaining access to Deflate unit.

  DFLT_CYCLES
  Cycles CPU is using Deflate unit.

  DFLT_CC
  Increments by one for every DEFLATE CONVERSION CALL instruction executed that ended in Condition Codes 0, 1, or 2.

  Issue the **`lscpumf`** command with the `-C` option to find out how these names map to the counter numbers on your IBM Z hardware. In the edition of *IBM The CPU-Measurement Facility Extended Counters Definition for z10, z196/z114, zEC12/zBC12, z13/z13s, z14, z15 and z16*, SA23-2261 for your hardware model, the counters are listed by counter number.

  For information about working with hardware counters, see Chapter 56, “Using the CPU-measurement facilities,” on page 543.

Overriding the defaults

You can override the defaults for compression and decompression with the Integrated Accelerator for zEDC.

Overrides for applications

Use the `DFLTCC` and `DFLTCC_LEVEL_MASK` environment variables to override the defaults for Java applications and, generally, for applications that use zlib or gzip in user space.

For Linux containers, specify these environment variables with the command that instantiates the container. For example, if you manage your containers with **podman**, use the `-e` option of the **podman `run`** command.

Turning off acceleration

Set the `DFLTCC` environment variable to 0 to turn off on-chip compression and decompression with the Integrated Accelerator for zEDC.

The `DFLTCC` environment variable can take the following values:

1
turns on-chip acceleration on. This is the default.

0
turns on-chip acceleration off.

Configuring accelerated compression for any compression level

By default, software that requests compression level 1 uses the on-chip accelerator if it is enabled. Use the `DFLTCC_LEVEL_MASK` environment variable to configure on-chip acceleration for any combination of compression levels.

The values of the `DFLTCC_LEVEL_MASK` environment variable are 4-digit hexadecimal numbers in the range 0x0000 - 0x03ff. Of the ten corresponding binary digits that can be 1, each represents a compression level. The least significant bit represents an assumed level 0, the most significant bit represents level 9.

The following examples demonstrate how the mask works:

0x0000
The bits for all compression levels are off. No on-chip compression is performed. This setting has the same effect on compression as setting the DFLTCC environment variable to 0.

0x0001
The bit for the assumed compression level 0 is on and overrides the default behavior for level 0. Instead of transferring data into a compressed format without a size reduction, data is actually compressed, which can have unintended consequences.

Note: Do not set this bit unless you are a compression expert who understands the implications and wants to experiment with this setting.

0x0002
The bit for compression level 1 is on; all other bits are off. On-chip compression is performed only for software that requests compression level 1.

This is the default. You distribution might, in effect, override this default by compiling the source code with a different value for the -DDFLTCC_LEVEL_MASK preprocessor option.

0x0006
The bits for compression level 1 and 2 are on; the other bits are off. On-chip compression is performed for software that requests compression level 1 or 2.

0x000e
The bits for compression level 1, 2, and 3 are on; the other bits are off. On-chip compression is performed for software that requests compression level 1, 2, or 3.

0x007e
The bits for compression level 1 - 6 are on; the bits for level 0, 7, 8, and 9 are off. On-chip compression is performed for software that requests a compression level in the range 1 - 6. Level 6 is the default for software that does not request a particular compression level.

0x01fe
The bits for compression level 1 - 8 are on; the bits for level 0 and 9 are off. On-chip compression is performed for software that requests a compression level in the range 1 - 8.

Note: On-chip compression with the Integrated Accelerator for zEDC is approximately equivalent to compression level 1. Forcing On-chip compression for software that requests a higher compression level can result in a larger compressed data volume than intended by the author of the software.

You can set the environment variable for all users, programs and system services of a Linux instance by writing the setting to `/etc/environment`.

```
# echo DFLTCC_LEVEL_MASK=0x1fe >> /etc/environment
```

The following examples take a more cautious approach by limiting the scope of the setting:

- Use **env** to limit the setting to an individual command call:

```
# env DFLTCC_LEVEL_MASK=0x1fe <command>
```

- Use an entry in your `~/.bashrc` for the scope of your bash sessions:

```
# echo DFLTCC_LEVEL_MASK=0x1fe >> ~/.bashrc
```

- Use a systemd unit override for a service *<your_service>* for the scope of that systemd service:

```
# printf "[Service]\nEnvironment=DFLTCC_LEVEL_MASK=0x1fe\n" > \
/etc/systemd/system/<your_service>.service.d/dfltcc.conf
```

- Use an override in the global systemd configuration file for the scope of all systemd services:

```
# printf "[Manager]\nDefaultEnvironment=DFLTCC_LEVEL_MASK=0x1fe\n" > \
/etc/systemd/system.conf.d/dfltcc.conf
```

Overrides for the kernel zlib

Use the `dfltcc=` kernel parameter to override the defaults for the kernel zlib.

Format

dfltcc= syntax



on
enables on-chip acceleration for compression level 1 and for decompression. This is the default.

off
turns off on-chip acceleration for both compression and decompression.

def_only
enables on-chip acceleration for compression on level 1 but not for decompression.

inf_only
enables on-chip acceleration for decompression only.

always
enables on-chip acceleration for decompression and for compression regardless of the requested compression level.

Note: On-chip compression with the Integrated Accelerator for zEDC is approximately equivalent to compression level 1. Forcing On-chip compression for software that requests a higher compression level can result in a larger compressed data volume than intended by the author of the software.

Examples

```
dfltcc=inf_only
```

Accelerating btrfs

If the kernel zlib is compiled with support for the Integrated Accelerator for zEDC, you can enable it for btrfs through a mount option.

By default, btrfs requests compression level 3, but the Integrated Accelerator for zEDC provides compression level 1. If compression level 1 is acceptable for your purposes, mount your instance of btrfs with the `compress=zlib:1` option.

Chapter 32. Data compression with GenWQE and zEDC Express

LPAR and z/VM: Data compression with GenWQE and zEDC Express applies to Linux in LPAR mode and to Linux on z/VM.

Generic Work Queue Engine (GenWQE) supports hardware-accelerated data compression and decompression through zEDC Express, a PCIe-attached Field Programmable Gate Array (FPGA) acceleration adapter.

zEDC Express is available on IBM Z and LinuxONE hardware up to z14 and LinuxONE II. As of z15 and LinuxONE III, zEDC hardware acceleration is available through on-chip compression and decompression, see Chapter 31, “Data compression with the Integrated Accelerator for zEDC,” on page 387.

zEDC hardware-acceleration is available for both Linux and z/OS. For more information about zEDC on z/OS and about setting up zEDC Express, see *Reduce Storage Occupancy and Increase Operations Efficiency with IBM zEnterprise Data Compression*, SG24-8259. You can obtain this publication from the IBM Redbooks website at www.redbooks.ibm.com/abstracts/sg248259.html.

Features

GenWQE supports hardware-accelerated data compression and decompression with common standards.

- GenWQE implements the `zlib` API.
- GenWQE adheres to the following RFCs:
  - RFC 1950 (zlib)
  - RFC 1951 (deflate)
  - RFC 1952 (gzip)

  These standards ensure compatibility among different `zlib` implementations.
  - Data that is compressed with GenWQE can be decompressed through a `zlib` software library.
  - Data that is compressed through a software `zlib` software library can be decompressed with GenWQE.
- GenWQE supports the following PCIe FPGA acceleration hardware:
  - zEDC Express

What you should know about GenWQE

Learn about the GenWQE components, how to enable GenWQE accelerated zlib for user applications, and device representation in Linux.

The GenWQE accelerated zlib

The GenWQE accelerated zlib can replace a `zlib` software library.

For data compression and decompression tasks, Red Hat Enterprise Linux 9.1 includes software libraries. The zlib library, which provides the `zlib` API, is one of the most commonly used libraries for data compression and decompression. For information about `zlib`, see www.zlib.net.

Because the GenWQE accelerated zlib offers the `zlib` API, applications can use it instead of the default `zlib` software library. The GenWQE hardware-accelerated `zlib` is designed to enhance performance by offloading tasks to a hardware accelerator.



Figure 93. GenWQE accelerated zlib

Applications

You can make the user space components of the GenWQE hardware-accelerated zlib available to applications that request data compression functions through the `zlib` API. Red Hat Enterprise Linux 9.1 provides these user space components with the `genwqe-zlib` RPM.

A second RPM, `genwqe-tools`, provides tools that use the GenWQE hardware-accelerated zlib.

IBM Java version 7.1 or later includes components of the GenWQE hardware-accelerated zlib. Through these components, it can directly address the GenWQE device nodes. With the required environment variables in place, it uses hardware-acceleration if it is available (see "GenWQE hardware-acceleration for IBM Java" on page 398).

Hardware-accelerated zlib

The hardware-accelerated zlib is a `zlib` implementation that acts as a wrapper for two included libraries:

libzHW
: a hardware library that prepares requests for processing by the hardware accelerator. The hardware library is intended to handle the bulk of the requests.

  This library also manages data buffers for optimized hardware compression.

libz
: a software implementation of the `zlib` interface. Because it provides the same interface as its wrapper library, it can handle any requests unmodified.

The hardware-accelerated zlib arbitrates between the two included libraries. It uses the software library as a backup if no hardware accelerator is available. It also evaluates the expected performance gain against the extra processing for channeling requests to the accelerator. For small or fragmented data, software processing might be advantageous, especially for decompression. The evaluation takes available resources, such as buffer space, into account.

Card library

The card library, `libcard`, mediates between the hardware-accelerated zlib library and the GenWQE device driver. It provides recovery features and can move jobs between available accelerators.

Device driver

The GenWQE device driver is the kernel part of GenWQE. It serializes requests to an accelerator in form of device driver control blocks (DDCBs), and it enables multi-process and multi-thread usage.

GenWQE device nodes

GenWQE user space components use device nodes to exchange data with the GenWQE device driver.

Red Hat Enterprise Linux 9.1 automatically loads the GenWQE device driver module when it is required. It also creates a device node of the form `/dev/genwqe<n>_card` for each available virtual acceleration card. *<n>* is an index number that identifies an individual virtual card. Node `/dev/genwqe0_card` is assigned to the first card that is detected, `/dev/genwqe1_card` to the second card, and so on.

Do not directly use these device nodes. The nodes are intended to be used by the user space components of the GenWQE hardware-accelerated zlib and by IBM Java.

Virtual accelerators

Each physical accelerator card can provide up to 15 virtual cards. In PCIe terminology, these virtual cards are called virtual functions.

GenWQE accelerator cards, as detected by Linux on IBM Z, are virtual cards. Which and how many cards are available to a particular Linux instance depends on the mainframe configuration and, if applicable, the hypervisor configuration.

As for most mainframe devices, availability can be enhanced by assigning virtual accelerator cards from different physical cards.

A degree of load distribution can be achieved by unevenly distributing accelerator cards among different Linux instances.

Tradeoff between best compression and speed

A minimum size of compressed data and fast compression are conflicting goals.

For hardware-accelerated compression with GenWQE, the compression ratio is roughly equivalent to **`gzip --fast`**.

Data that was compressed with GenWQE hardware-acceleration might have a different size from data that was compressed in software. The data compression standards are not violated by this difference. Despite possible differences in size of the compressed data, data that is compressed with GenWQE hardware-acceleration can be decompressed in software and vice versa.

Setting up GenWQE hardware acceleration

Install the GenWQE components and understand how environment variables can override default settings.

Installing the GenWQE hardware-accelerated zlib

Install the `genwqe-zlib` and `genwqe-tools` RPMs that are included in Red Hat Enterprise Linux 9.1.

The `genwqe-zlib` RPM includes the user space components of the GenWQE hardware-accelerated zlib.

The `genwqe-tools` RPM provides the following tools:

- **`genwqe_gzip`** and **`genwqe_gunzip`**, which are GenWQE versions of **`gzip`** and **`gunzip`** (see "Examples for using GenWQE" on page 397).

These tools can be used for most purposes, but they do not implement all of the more unusual options of their common code counterparts. See the man pages to find out which options are supported.

- **genwqe_echo**, a tool to confirm the availability of accelerator hardware through the GenWQE accelerated zlib. See "Confirming that the accelerator hardware can be reached" on page 399 for details.

Environment variables

You can set environment variables to control the GenWQE hardware-accelerated zlib.

The GenWQE hardware-accelerated zlib uses defaults that correspond to the following environment variable settings:

```
ZLIB_ACCELERATOR=GENWQE
ZLIB_CARD=-1
ZLIB_TRACE=0x0
ZLIB_DEFLATE_IMPL=0x41
ZLIB_INFLATE_IMPL=0x41
```

You can override these defaults by setting the following environment variables:

ZLIB_ACCELERATOR
Sets the accelerator type. For zEDC Express, the type is `GENWQE`.

ZLIB_CARD
`-1`, uses all accelerators that are available to the Linux instance. Failed requests are retried on alternative accelerators.

You can specify the ID of a particular virtual accelerator card to be used. The ID is the index number that makes the nodes unique. All other cards are ignored, and no retry on alternative cards is performed if the specified card fails. Specify an ID only if you want to test a particular card.

`0` uses the first card that is found by the device driver. As for specifying an individual card, all other cards are ignored.

ZLIB_TRACE
Sets tracing bits:

0x1
General trace.

0x2
Hardware trace.

0x4
Software trace.

0x8
Trace summary at the end of a process.

Tracing requires extra processing and incurs a performance penalty. The least performance impact is to be expected from the trace summary. By default, tracing is off.

ZLIB_DEFLATE_IMPL
`0x01` and `0x41` enable hardware compression, where `0x41` adds an optimization setting. `0x00` forces software compression and is intended for experimentation, for example, for gathering performance data with and without hardware acceleration.

ZLIB_INFLATE_IMPL
`0x01` and `0x41` enable hardware decompression, where `0x41` adds an optimization setting. `0x00` forces software decompression and is intended for experimentation, for example, for gathering performance data with and without hardware acceleration.

You can find more details about the environment variables in the GenWQE wiki on GitHub at github.com/ibm-genwqe/genwqe-user/wiki/Environment Variables.

Examples for using GenWQE

You can use the GenWQE hardware-accelerated zlib through GenWQE tools.

Activating the GenWQE hardware-accelerated zlib for an application

Whether and how you can make an application use the GenWQE hardware-accelerated zlib depends on how the application links to `libz.so`.

Examine the application for links to `libz.so`, for example with the **ldd** tool.

- If the application does not link to `libz.so` or if it statically links to `libz.so`, it would require recompilation, and possibly code changes, to make acceleration through GenWQE possible.
- If an application dynamically links to `libz.so`, you might be able to redirect the library calls from the default implementation to the GenWQE hardware-accelerated `zlib`.

Some applications require `zlib` features that are not available from the GenWQE hardware-accelerated zlib. Such applications fail if a global redirect is put in place. The following technique redirects calls for the scope of a particular application.

Specify the LD_PRELOAD environment variable to load the GenWQE hardware-accelerated zlib. Set the variable with the start command for your application.

Example:

```
# LD_PRELOAD=/lib/s390x-linux-gnu/genwqe/libz.so.1 <application_start_cmd>
```

Compressing data with genwqe_gzip

GenWQE provides two tools, **genwqe_gzip** and **genwqe_gunzip** that can be used in place of the common code **gzip** and **gunzip** tools. The GenWQE versions of the tools use hardware acceleration if it is available.

Procedure

Run the **genwqe_gzip** command with the `-AGENWQE` parameter to compress a file.

```
# genwqe_gzip -AGENWQE <file>
```

The `-AGENWQE` parameter ensures that the correct, PCIe-attached, accelerator card is used. Also use this option when decompressing data with the **genwqe_gunzip** command. See the man pages for other options.

Running tar with GenWQE hardware-acceleration

You can make tar use **genwqe_gzip** in place of the common code **gzip**.

About this task

If called with the z option, the **tar** utility uses the first **gzip** tool in the search path, which is usually the common code version. By inserting the path to the GenWQE **gzip** tool at the beginning of the PATH variable, you can make the **tar** utility use hardware acceleration.

The path points to `/usr/lib64/genwqe/gzip` and `/usr/lib64/genwqe/gunzip`, which are symbolic links to **genwqe_gzip** and **genwqe_gunzip**.

The acceleration is most marked for a single large text file. The example that follows compresses a directory with the Linux source code.

Procedure

1. Run the **tar** command as usual to use software compression. To obtain performance data, specify the **tar** command as an argument to the **time** command.

```
# time tar cfz linux-src.sw.tar.gz linux-src
real 0m22.329s
user 0m22.147s
sys 0m0.849s
```

2. Run the **tar** command with an adjusted PATH variable to use GenWQE hardware acceleration. Again, use the **time** command to obtain performance data.

```
# time PATH=/usr/lib64/genwqe:$PATH \
tar cfz linux-src.hw.tar.gz linux-src
real 0m1.323s
user 0m0.242s
sys 0m1.023s
```

Results

In the example, the accelerated operation is significantly faster. The hardware-compressed data is slightly larger than the software-compressed version of the same data

GenWQE hardware-acceleration for IBM Java

IBM Java version 7.1 or later can use the GenWQE hardware-accelerated zlib.

To activate the GenWQE hardware-accelerated zlib for IBM Java, you must set environment parameters. See the documentation for your Java version to find out which settings are required.

Note: Any values that you set for the environment variables override the default settings for the GenWQE user space components (see “Environment variables” on page 396).

Exploring the GenWQE setup

You might want to ensure that your GenWQE setup works as intended.

- “Listing your GenWQE accelerator cards” on page 398
- “Checking the GenWQE device driver setup” on page 399
- “Confirming that the accelerator hardware can be reached” on page 399

Listing your GenWQE accelerator cards

Use the **lspci** command to list the available GenWQE accelerator cards.

Procedure

1. Issue the **lspci** command and look for GenWQE.

   Example:

```
# lspci |grep GenWQE
0002:00:00.0 Processing accelerators: IBM GenWQE Accelerator Adapter
```

2. Issue the **lspci** command with the verbose option to display details about a particular card.

   Example:

```
# lspci -vs 0002:00:00.0
0002:00:00.0 Processing accelerators: IBM GenWQE Accelerator Adapter
        Subsystem: IBM GenWQE Accelerator Adapter
        Physical Slot: 000000ff
        Flags: bus master, fast devsel, latency 0, IRQ 3
        Memory at 8002000000000000 (64-bit, prefetchable) [disabled] [size=128M]
        Capabilities: [50] MSI: Enable+ Count=1/1 Maskable- 64bit+
        Capabilities: [80] Express Endpoint, MSI 00
        Capabilities: [100] Alternative Routing-ID Interpretation (ARI)
        Kernel driver in use: genwqe
        Kernel modules: genwqe_card
```

Checking the GenWQE device driver setup

Perform these tasks if GenWQE does not work as expected.

Procedure

1. Confirm that the device driver is loaded.

   ```
   # lsmod | grep genwqe
   genwqe_card 88997 0
   crc_itu_t 1910 1 genwqe_card
   ```

   If the `genwqe_card` module is not listed in the command output, load it with **modprobe**.

   ```
   # modprobe genwqe_card
   ```

   The `genwqe_card` module does not have module parameters.
2. Confirm that GenWQE device nodes exist and that the nodes have the required permissions.

   The nodes must grant read and write permissions to all users, for example:

   ```
   # ls -l /dev/genwqe*
   crwrwrw 1 root root 249, 0 Jun 30 10:01 /dev/genwqe0_card
   crwrwrw 1 root root 248, 0 Jun 30 10:01 /dev/genwqe1_card
   ```

   If the permissions are not `crwrwrw`, create a file `/etc/udev/rules.d/52-genwqedevices.rules` with this rule as its content:

   ```
   KERNEL=="genwqe*", MODE="0666"
   ```

   The new rule takes effect next time the GenWQE device driver is loaded.

   Tip: Use the **chmod** command to temporarily set the permissions.

What to do next

You can find debug information in the Linux source tree at `/sys/kernel/debug/genwqe` and at `/sys/class/genwqe`.

Confirming that the accelerator hardware can be reached

The **genwqe_echo** command is similar to a **ping** command.

Before you begin

The **genwqe_echo** command is included in the `genwqe-tools` RPM (see “Installing the GenWQE hardware-accelerated zlib” on page 395).

Procedure

Issue a command of this form to confirm that you can reach the accelerator hardware.

```
# genwqe_echo -AGENWQE -C <n> -c <m>
```

In the command, *<n>* is the index number of the card and *<m>* is a positive integer that specifies how many requests are sent to the card. The `-AGENWQE` parameter ensures that the correct, PCIe-attached, accelerator card is used.

Example: The following command sends four requests to the card with device node `/dev/genwqe1_card`:

```
# genwqe_echo -AGENWQE -C 1 -c 4
1 x 33 bytes from UNIT #1: echo_req time=37.0 usec
1 x 33 bytes from UNIT #1: echo_req time=19.0 usec
1 x 33 bytes from UNIT #1: echo_req time=23.0 usec
1 x 33 bytes from UNIT #1: echo_req time=18.0 usec
--- UNIT #1 echo statistics ---
4 packets transmitted, 4 received, 0 lost, 0% packet loss
```

See the **genwqe_echo** man page for other command options.

External programming interfaces

The GenWQE hardware-accelerated zlib implements a large subset of the original software zlib.

For information about programming against the GenWQE hardware-accelerated zlib, see the section about implemented zlib functions in *Accelerated Data Compressing using the GenWQE Linux Driver and Corsa FPGA PCIe card*.

Chapter 33. PCI Express support

The Peripheral Component Interconnect Express (PCIe) device driver supports various PCI devices, including but not limited to devices that implement the SMC network protocol.For more information about RoCE, see Chapter 22, “RDMA over Converged Ethernet,” on page 349. For more information about ISM, see Chapter 23, “Internal shared memory device driver,” on page 353.

PCIe functions are seen by Linux as devices, hence devices is used here synonymously. You can assign PCIe devices to LPARs in the IOCDS.

PCIe function addresses

The function addresses uniquely identifies a PCIe function within a Linux instance. Function addresses adhere to this format: *<domain>*:*<bus>*:*<device>*.*<function>*. For Linux on IBM Z, the address components have the following values:

<domain>
: UID as specified for the PCI function in the hardware configuration (IOCDS). UIDs are unique hexadecimal values in the range 1 - FFFF. For example, with a UID of `0x318`, *<domain>* would be: `0318`.

  UIDs are available only if supported by the hardware and if the LPAR is enabled for UID uniqueness checking. If your environment does not support UIDs for PCIe functions, consecutive numbers, starting from `0000`, are assigned to the functions. The mapping of assigned numbers and physical functions does not persist across reboots.

<bus>
: Two zeros: `00`.

<device>.<function>
: If your environment supports the Alternative Routing-ID Interpretation (ARI) compatible address format, *<device>*.*<function>* might represent the PCIe Routing-ID (RID) . The previous constant value, `00.0`, continues to be used by some device types, for example by NVMe devices. It is also used as a fallback for environments that do not support the ARI compatible address format. You can force this previous value with the `pci=norid` kernel parameter (see “Setting up the PCIe support” on page 401).

Setting up the PCIe support

Configure the PCIe support through the `pci=` kernel parameter.

PCIe devices are automatically configured during the system boot process. In contrast to most IBM Z devices, all PCIe devices that are in a configured state are automatically set online. PCIe devices that are in stand-by state are not automatically enabled.

Scanning of PCIe devices is enabled by default. To disable use of PCI devices, set the kernel command line parameter **`pci=off`**.

PCI kernel parameter syntax



where:

off
: disables automatic scanning of PCIe devices.

on
enables automatic scanning of PCIe devices (default).

pci=nomio
if available, PCIe uses enhanced instructions as introduced with z15. Specify this kernel parameter to use the previous instructions.

pci=norid
PCI function addresses follow an Alternative Routing-ID Interpretation (ARI) conform format if it is supported by the system environment. Specify this kernel parameter to use the previous format.

Attention: Other PCI kernel parameters do not apply to IBM Z and might have adverse effects on your system.

Example

The following kernel parameter enables automatic scanning of PCIe devices.

```
pci=on
```

Using PCIe hotplug on LPAR

Use PCIe hotplug to change the availability of a shared PCIe device.

About this task

Only one LPAR can access a PCIe device. Other LPARs can be candidates for access. Use the HMC or SE to define which LPAR is connected and which LPARs are on the candidate list. A PCIe device that is defined, but not yet used, is shown as a PCIe slot in Linux.

On Linux, you use the `power` sysfs attribute of a PCIe slot to connect the device to the LPAR where Linux runs. While a PCIe device is connected to one LPAR, it is in the reserved state for all other LPARs that are in the candidates list. A reserved PCIe device is invisible to the operating system. The slot is removed from sysfs.

Procedure

The `power` attribute of a slot contains 0 if a PCIe device is in stand-by state, or 1 if the device is configured and usable.

1. Locate the slot for the card you want to work with.

   To locate the slot, read the `function_id` attribute of the PCIe device from sysfs.

   For example, to read the `/sys/bus/pci/devices/0000:00:00.0/function_id` issue:

   ```
   # cat /sys/bus/pci/devices/0000:00:00.0/function_id
   0x00000011
   ```

   where `00000011` is the slot. Alternatively, you can use the **`lspci -v`** command to find the slot.
2. Write the value that you want to the `power` attribute:

   - Write 1 to `power` to connect the PCIe device to the LPAR in which your Linux instance is running. Linux automatically scans the device, registers it, and brings it online. For example:

     ```
     echo 1 > /sys/bus/pci/slots/00000011/power
     ```

   - Write 0 to `power` to stop using the PCIe device. The device state changes to stand-by. The PCIe device is set offline automatically. For example:

     ```
     echo 0 > /sys/bus/pci/slots/00000011/power
     ```

A PCIe device in standby is also in the standby state to all other LPARs in the candidates list. A standby PCIe device appears as a slot, but without a PCIe device.

Recovering a PCIe device

Use the `recover` sysfs attribute to recover a PCIe device.

About this task

A kernel message is displayed when a PCIe device enters the error state. It is not possible to automatically relieve the PCIe device from this state.

Procedure

1. Find out which PCIe device is in an error state by issuing the **`lspci`** command.
   In the following example, the device in error state can be identified by the trailing "(rev ff)" in the output line.

   ```
   # lspci
   0000:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4 Lx Virtual Function] (rev ff)
   0001:00:00.0 Ethernet controller: Mellanox Technologies MT27710 Family [ConnectX-4 Lx Virtual Function]
   0002:00:00.0 Non-VGA unclassified device: IBM Internal Shared Memory (ISM) virtual PCI device
   ```

2. Find the PCIe device directory in sysfs.

   PCIe device directories are of the form `/sys/bus/pci/devices/<function_address>`, where *<function_address>* identifies the PCIe device.

   For example: `/sys/bus/pci/devices/0000:00:00.0`.

3. Write 1 to the `recover` attribute of the PCIe device.
   For example:

   ```
   # echo 1 > /sys/bus/pci/devices/0000:00:00.0/recover
   ```

   After a successful recovery, the PCI device is de-registered and reprobed.

Reporting defective PCIe devices

For Linux in LPAR mode or Linux on z/VM, use the **`zpcictl`** command to report defective PCIe devices to the Support Element (SE). Such devices might require physical repair actions.

Before you begin

- You need to know the function address of the defective PCIe device or a device node that represents the device.
- To send diagnostic data with the error report you need to install the `smartmontools` package. Whether data is collected and which data is available depends on the PCI device type. For example, S.M.A.R.T. data is gathered for NVMe devices.

Procedure

Issue a command of this form to report a device with function address `0000:00:00.0` to the SE:

```
# zpcictl --report-error <device>
```

where *<device>* is the device's function address or a device node that represents the device.

Example:

```
# zpcictl --report-error 0000:00:00.0
```

Displaying PCIe information

To display information about PCIe devices, read the attributes of the devices in sysfs.

About this task

The sysfs representation of a PCIe device or slot is a directory of the form `/sys/bus/pci/devices/<function_address>`, where *<function_address>* identifies the PCIe device. This sysfs directory contains a number of attributes with information about the PCIe device.

Table 52. Read-only attributes with PCIe device information

Attribute	Explanation
function_handle	Eight-character, hexadecimal PCI-function (device) handle.
function_id	Eight-character, hexadecimal PCI-function (device) ID. The ID identifies the PCIe device within the processor configuration. This value specifies the slot at `/sys/bus/pci/slots`.
pchid	Four-character, hexadecimal, physical channel ID. Specifies the slot of the PCIe adapter in the I/O drawer. Thus identifies the adapter that provides the device.
pfgid	Two-character, hexadecimal, physical function group ID.
pfip/segment0 /segment1 /segment2 /segment3	Two-character, hexadecimal, PCI-function internal path. Provides an abstract indication of the path that is used to access the PCI function. This can be used to compare the paths used by two or more PCI functions, to give an indication of the degree of isolation between them.
uid	Up to eight-character, hexadecimal, user-defined identifier.
vfn	Four-character, hexadecimal, number that identifies the virtual function within the adapter.
util_string	Type-specific information about the device. For RoCE devices and ISM devices, it contains the PNET ID if a PNET ID has been assigned in the I/O configuration.

Procedure

Issue a command of this form to read an attribute:

```
# cat /sys/bus/pci/devices/<function_address>/<attribute>
```

where *<attribute>* is one of the attributes of Table 52 on page 404.

Reading statistics for a PCIe device

Use the `statistics` attribute file to see measurement data for a PCIe device.

About this task

All PCIe devices collect measurement data by default. You can read the data in a sysfs attribute file in the debug file system, by default mounted at `/sys/kernel/debug`.

You can turn data collection on and off. To switch off measurement data collecting for the current session, write "0" to the `statistics` attribute. To enable data collection again, write "1" to the `statistics` attribute.

Example

To read measurement data for a (RoCE) function named 0000:00:00.0 use:

```
# cat /sys/kernel/debug/pci/0000:00:00.0/statistics
```

The statistics attribute file might look similar to this example:

```
FMB @ 0000000078cd8000
Update interval: 4000 ms
Samples: 14373
Last update TOD: cefa44fa50006378
           Load operations:     1002780
          Store operations:     1950622
    Store block operations:     0
        Refresh operations:     0
            Received bytes:     0
          Received packets:     0
         Transmitted bytes:     0
       Transmitted packets:     0
           Allocated pages:     9104
              Mapped pages:     16633
            Unmapped pages:     2337
```

Part 6. z/VM virtual server integration

z/VM only: This part applies to Linux on z/VM only.

These device drivers and features help you to effectively run and manage a z/VM-based virtual Linux server farm.

Newest version

You can find the newest version of this book at
ibm.com/docs/en/linux-on-systems?topic=linuxone-distributions

Restrictions

For prerequisites and restrictions see the IBM Z architecture specific information in the Red Hat Enterprise Linux 9.1 release notes at
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

Chapter 34. z/VM concepts

z/VM only: This chapter applies to Linux on z/VM only.

The z/VM performance monitoring and cooperative memory management concepts help you to understand how the different components interact with Linux.

Performance monitoring for z/VM guest virtual machines

You can monitor the performance of z/VM guest virtual machines and their guest operating systems with performance monitoring tools on z/VM or on Linux.

These tools can be your own, IBM tools such as the Performance Toolkit for VM, or third-party tools. The guests being monitored require agents that write monitor data.

Monitoring on z/VM

z/VM monitoring tools must read performance data. For monitoring Linux instances, this data is APPLDATA monitor records.

Linux instances must write these records for the tool to read, as shown in Figure 94 on page 409.



Figure 94. Linux instances write APPLDATA records for performance monitoring tools

Both user space applications and the Linux kernel can write performance data to APPLDATA records. Applications use the monwriter device driver to write APPLDATA records. The Linux kernel can be configured to collect system level data such as memory, CPU usage, and network-related data, and write it to data records.

For file system size data, there is a command, **mon_fsstatd**. This user space tool uses the monwriter device driver to write file system size information as defined records.

For process data, there is a command, **mon_procd**. This user space tool uses the monwriter device driver to write system information as defined records.

In summary, Red Hat Enterprise Linux 9.1 for IBM Z supports writing and collecting performance data as follows:

- The Linux kernel can write z/VM monitor data for Linux instances, see Chapter 35, “Writing kernel APPLDATA records,” on page 413.
- Linux applications that run on z/VM guests can write z/VM monitor data, see Chapter 36, “Writing z/VM monitor records,” on page 419.

- You can collect monitor file system size information, see “mon_fsstatd – Monitor z/VM guest file system size” on page 695.
- You can collect system information about up to 100 concurrently running processes, see “mon_procd – Monitor Linux on z/VM” on page 699.

Monitoring on Linux

A Linux instance can read the monitor data by using the monreader device driver.

Figure 95 on page 410 illustrates a Linux instance that is set up to read the monitor data. You can use an existing monitoring tool or write your own software.



Figure 95. Performance monitoring using monitor DCSS data

In summary, Red Hat Enterprise Linux 9.1 for IBM Z supports reading performance data in the form of read access to z/VM monitor data for Linux instances. See Chapter 37, “Reading z/VM monitor records,” on page 423 or more details.

Further information

Several z/VM publications include information about monitoring.

- See *z/VM: Getting Started with Linux on System z*, SC24-6287, the chapter about monitoring performance for information about using the CP Monitor and the Performance Toolkit for VM.
- See *z/VM: Saved Segments Planning and Administration*, SC24-6322 for general information about DCSSs (z/VM keeps monitor records in a DCSS).
- See *z/VM: Performance*, SC24-6301 for information about creating a monitor DCSS.
- See *z/VM: CP Commands and Utilities Reference*, SC24-6268 for information about the CP commands that are used in the context of DCSSs and for controlling the z/VM monitor system service.
- For the layout of the monitor records, see Chapter 35, “Writing kernel APPLDATA records,” on page 413 and visit

  ```
  www.ibm.com/vm/pubs/ctlblk.html
  ```

- For more information about performance monitoring on z/VM, visit

  ```
  www.ibm.com/vm/perf
  ```

Cooperative memory management background

Cooperative memory management (CMM, or "cmm1") dynamically adjusts the memory available to Linux.

For information about setting up CMM, see Chapter 43, “Cooperative memory management,” on page 453.

In a virtualized environment it is common practice to give the virtual machines more memory than is actually available to the hypervisor. Linux tends to use all of its available memory. As a result, the hypervisor (z/VM) might start swapping.

To avoid excessive z/VM swapping, the memory available to Linux can be reduced. CMM allocates pages to page pools that make the pages unusable to Linux. There are two such page pools as shown in Figure 96 on page 411.



Figure 96. Page pools

There are two page pools:

A static page pool

The page pool is controlled by a resource manager that changes the pool size at intervals according to guest activity and overall memory usage on z/VM (see Figure 97 on page 411).



Figure 97. Static page pool

A timed page pool

Pages are released from this pool at a speed that is set in the *release rate* (see Figure 98 on page 412). According to guest activity and overall memory usage on z/VM, a resource manager adds pages at intervals. If no pages are added and the release rate is not zero, the pool empties.



Figure 98. Timed page pool

The external resource manager that controls the pools can be the z/VM resource monitor (VMRM) or a third-party systems management tool.

VMRM controls the pools over a message interface. Setting up the external resource manager is beyond the scope of this information. For more details, see the chapter on VMRM in *z/VM: Performance*, SC24-6301.

Third-party tools can provide a Linux deamon that receives commands for the memory allocation through TCP/IP. The deamon, in turn, uses the procfs-based interface. You can use the procfs interface to read the pool sizes. These values are useful diagnostic data.

Linux guest relocation

Information about guest relocations is stored in the s390 debug feature (s390dbf).

You can access this information in a kernel dump or from a running Linux instance. For more information, see *Troubleshooting*, SC34-2612.

Chapter 35. Writing kernel APPLDATA records

z/VM only: APPLDATA records apply to Linux on z/VM only.

z/VM is a convenient point for collecting z/VM guest performance data and statistics for an entire server farm. Linux instances can export such data to z/VM by using APPLDATA monitor records.

z/VM regularly collects these records. The records are then available to z/VM performance monitoring tools.

A virtual CPU timer on the Linux instance to be monitored controls when data is collected. The timer accounts for only busy time to avoid unnecessarily waking up an idle guest. The APPLDATA record support comprises several modules. A base module provides an intra-kernel interface and the timer function. The intra-kernel interface is used by *data gathering modules* that collect actual data and determine the layout of a corresponding APPLDATA monitor record (see "APPLDATA monitor record layout" on page 415).

For an overview of performance monitoring support, see "Performance monitoring for z/VM guest virtual machines" on page 409.

Setting up the APPLDATA record support

You must enable your z/VM guest virtual machine for data gathering and load the APPLDATA record support modules.

Procedure

1. On z/VM, ensure that the user directory of the guest virtual machine includes the option APPLMON.
2. On Linux, use the **modprobe** command to load any required modules.

APPLDATA record support module parameter syntax



where appldata_mem, appldata_os, and appldata_net_sum are the modules for gathering memory-related data, operating system-related data, and network-related data.

See the **modprobe** man page for command details.

Generating APPLDATA monitor records

You can set the timer interval and enable or disable data collection.

You control the monitor stream support through the procfs. APPLDATA monitor records are produced if both a particular data gathering module and the monitoring support in general are enabled.

Enabling or disabling the support

Use the procfs `timer` attribute to enable or disable the monitoring support.

Procedure

To read the current setting, issue:

```
# cat /proc/sys/appldata/timer
```

To enable the monitoring support issue:

```
# echo 1 > /proc/sys/appldata/timer
```

To disable the monitoring support issue:

```
# echo 0 > /proc/sys/appldata/timer
```

Activating or deactivating individual data-gathering modules

Each data-gathering module has a procfs entry that contains a value 1 if the module is active and 0 if the module is inactive.

About this task

The following procfs entries control the data-gathering modules:

`/proc/sys/appldata/mem` for the memory data-gathering module
`/proc/sys/appldata/os` for the CPU data-gathering module
`/proc/sys/appldata/net_sum` for the net data-gathering module

To check whether a module is active look at the content of the corresponding procfs entry.

Procedure

Issue a command of this form:

```
# echo <flag> > /proc/sys/appldata/<data_type>
```

where *<data_type>* is one of mem, os, or net_sum.

Note: An active data-gathering module produces APPLDATA monitor records only if the monitoring support is enabled (see “Enabling or disabling the support” on page 413).

Example

To find out whether memory data-gathering is active, issue:

```
# cat /proc/sys/appldata/mem
0
```

In the example, memory data-gathering is off. To activate memory data-gathering, issue:

```
# echo 1 > /proc/sys/appldata/mem
```

To deactivate the memory data-gathering module, issue:

```
# echo 0 > /proc/sys/appldata/mem
```

Setting the sampling interval

You can set the time that lapses between consecutive data samples.

About this task

The time that you set is measured by the virtual CPU timer. Because the virtual timer slows down as the guest idles, the sampling interval in real time can be considerably longer than the value you set.

The value in `/proc/sys/appldata/interval` is the sample interval in milliseconds. The default sample interval is 10 000 ms.

Procedure

To read the current value, issue:

```
# cat /proc/sys/appldata/interval
```

To set the sample interval to a different value, write the new value (in milliseconds) to `/proc/sys/appldata/interval`. Issue a command of this form:

```
# echo <interval> > /proc/sys/appldata/interval
```

where *<interval>* is the new sample interval in milliseconds. The specification must be in the range 1 - 2147483647, where 2,147,483,647 = 2^{31} - 1.

Example

To set the sampling interval to 20 s (20000 ms), issue:

```
# echo 20000 > /proc/sys/appldata/interval
```

APPLDATA monitor record layout

Each of the data gathering modules writes a different type of record.

- Memory data (see Table 53 on page 415)
- Processor data (see Table 54 on page 416)
- Networking (see Table 55 on page 417)

z/VM can identify the records by their unique product ID. The product ID is an EBCDIC string of this form: "LINUXKRNL*<record ID>*260100". The *<record ID>* is treated as a byte value, not a string.

The records contain data of the following types:

u32
unsigned 4 byte integer

u64
unsigned 8 byte integer

Table 53. APPLDATA_MEM_DATA record (Record ID 0x01)

Offset (Decimal)	Offset (Hex)	Type	Name	Description
0	0x0	u64	timestamp	TOD time stamp that is generated on the Linux side after record update
8	0x8	u32	sync_count_1	After z/VM collected the record data, sync_count_1 and sync_count_2 must be the same. Otherwise, the record was updated on the Linux side while z/VM was collecting the data. As a result, the data might be inconsistent.
12	0xC	u32	sync_count_2	
16	0x10	u64	pgpgin	Data that was read from disk (in KB)
24	0x18	u64	pgpgout	Data that was written to disk (in KB)
32	0x20	u64	pswpin	Pages that were swapped in

Table 53. APPLDATA_MEM_DATA record (Record ID 0x01) (continued)

Offset (Decimal)	Offset (Hex)	Type	Name	Description
40	0x28	u64	pswpout	Pages that were swapped out
48	0x30	u64	sharedram	Shared RAM in KB
56	0x38	u64	totalram	Total usable main memory size in KB
64	0x40	u64	freeram	Available memory size in KB
72	0x48	u64	totalhigh	Total high memory size in KB
80	0x50	u64	freehigh	Available high memory size in KB
88	0x58	u64	bufferram	Memory that was reserved for raw disk blocks, corresponding to "Buffers" from `/proc/meminfo`, in KB
96	0x60	u64	cached	Size of used cache, including "Cached" and "SwapCached" from `/proc/meminfo`, in KB
104	0x68	u64	totalswap	Total swap space size in KB
112	0x70	u64	freeswap	Free swap space in KB
120	0x78	u64	pgalloc	Page allocations
128	0x80	u64	pgfault	Page faults (major+minor)
136	0x88	u64	pgmajfault	Page faults (major only)

Table 54. APPLDATA_OS_DATA record (Record ID 0x02)

Offset (Decimal)	Offset (Hex)	Type (size)	Name	Description
0	0x0	u64	timestamp	TOD time stamp that is generated on the Linux side after record update
8	0x8	u32	sync_count_1	After z/VM collected the record data, sync_count_1 and sync_count_2 must be the same. Otherwise, the record was updated on the Linux side while z/VM was collecting the data. As a result, the data might be inconsistent.
12	0xC	u32	sync_count_2	See sync_count_1.
16	0x10	u32	nr_cpus	Number of virtual CPUs.
20	0x14	u32	per_cpu_size	Size of the per_cpu_data for each CPU (= 36).
24	0x18	u32	cpu_offset	Offset of the first per_cpu_data (= 52).
28	0x1C	u32	nr_running	Number of runnable threads.
32	0x20	u32	nr_threads	Number of threads.
36	0x24	3 × u32	avenrun[3]	Average number of running processes during the last 1 (1st value), 5 (2nd value) and 15 (3rd value) minutes. These values are "fake fix-point", each value is composed of a 10-bit integer and an 11-bit fractional part. See note “1” on page 417 at the end of this table.

Table 54. APPLDATA_OS_DATA record (Record ID 0x02) (continued)

Offset (Decimal)	Offset (Hex)	Type (size)	Name	Description
48	0x30	u32	nr_iowait	Number of blocked threads (waiting for I/O).
52	0x34	See note "2" on page 417.	per_cpu_data	Time that is spent in user, kernel, idle, nice, etc for every CPU. See note "3" on page 417 at the end of this table.
52	0x34	u32	per_cpu_user	Timer ticks that were spent in user mode.
56	0x38	u32	per_cpu_nice	Timer ticks that were spent with modified priority.
60	0x3C	u32	per_cpu_system	Timer ticks that were spent in kernel mode.
64	0x40	u32	per_cpu_idle	Timer ticks that were spent in idle mode.
68	0x44	u32	per_cpu_irq	Timer ticks that were spent in interrupts.
72	0x48	u32	per_cpu_softirq	Timer ticks that were spent in softirqs.
76	0x4C	u32	per_cpu_iowait	Timer ticks that were spent while waiting for I/O.
80	0x50	u32	per_cpu_steal	Timer ticks "stolen" by hypervisor.
84	0x54	u32	cpu_id	The number of this CPU.

Note:

1. The following C-Macros are used inside Linux to transform these into values with two decimal places:

```
#define LOAD_INT(x) ((x) >> 11)
#define LOAD_FRAC(x) LOAD_INT(((x) & ((1 << 11) - 1)) * 100)
```

2. nr_cpus * per_cpu_size
3. per_cpu_user through cpu_id are repeated for each CPU

Table 55. APPLDATA_NET_SUM_DATA record (Record ID 0x03)

Offset (Decimal)	Offset (Hex)	Type	Name	Description
0	0x0	u64	timestamp	TOD time stamp that is generated on the Linux side after record update
8	0x8	u32	sync_count_1	After z/VM collected the record data, sync_count_1 and sync_count_2 must be the same. Otherwise, the record was updated on the Linux side while z/VM was collecting the data. As a result, the data might be inconsistent.
12	0xC	u32	sync_count_2	See sync_count_1.
16	0x10	u32	nr_interfaces	Number of interfaces being monitored
20	0x14	u32	padding	Unused. The next value is 64-bit aligned, so these 4 byte would be padded out by compiler
24	0x18	u64	rx_packets	Total packets that were received
32	0x20	u64	tx_packets	Total packets that were transmitted

Table 55. APPLDATA_NET_SUM_DATA record (Record ID 0x03) (continued)				
Offset (Decimal)	**Offset (Hex)**	**Type**	**Name**	**Description**
40	0x28	u64	rx_bytes	Total bytes that were received
48	0x30	u64	tx_bytes	Total bytes that were transmitted
56	0x38	u64	rx_errors	Number of bad packets that were received
64	0x40	u64	tx_errors	Number of packet transmit problems
72	0x48	u64	rx_dropped	Number of incoming packets that were dropped because of insufficient space in Linux buffers
80	0x50	u64	tx_dropped	Number of outgoing packets that were dropped because of insufficient space in Linux buffers
88	0x58	u64	collisions	Number of collisions while transmitting

Programming interfaces

The monitor stream support base module exports two functions.

Application programmers: This information is intended for those who want to program against the monitor stream.

- appldata_register_ops() to register data-gathering modules
- appldata_unregister_ops() to undo the registration of data-gathering modules

Both functions receive a pointer to a struct appldata_ops as parameter. Additional data-gathering modules that want to plug into the base module must provide this data structure. You can find the definition of the structure and the functions in `arch/s390/appldata/appldata.h` in the Linux source tree.

See “APPLDATA monitor record layout” on page 415 for an example of APPLDATA data records that are to be sent to z/VM.

Tip: Include the timestamp, sync_count_1, and sync_count_2 fields at the beginning of the record as shown for the existing APPLDATA record formats.

Chapter 36. Writing z/VM monitor records

z/VM only: z/VM monitor records apply to Linux on z/VM only.

Applications can use the monitor stream application device driver to write z/VM monitor APPLDATA records to the z/VM *MONITOR stream.

For an overview of performance monitoring support, see “Performance monitoring for z/VM guest virtual machines” on page 409.

The monitor stream application device driver interacts with the z/VM monitor APPLDATA facilities for performance monitoring. A better understanding of these z/VM facilities might help when you are using this device driver. See *z/VM: Performance*, SC24-6301 for information about monitor APPLDATA.

The monitor stream application device driver provides the following functions:

- An interface to the z/VM monitor stream.
- A means of writing z/VM monitor APPLDATA records.

Setting up the z/VM *MONITOR record writer device driver

You must load the monwriter module on Linux and set up your guest virtual machine for monitor records on z/VM.

Module parameters

You can configure the monitor stream application device driver when you are loading the device driver module, monwriter.

Monitor stream application device driver module parameter syntax



where *<numbufs>* is the maximum number of monitor sample and configuration data buffers that can exist in the Linux guest at one time. The default is 255.

Example

To load the monwriter module and set the maximum number of buffers to 400, use the following command:

```
# modprobe monwriter max_bufs=400
```

Setting up the z/VM guest virtual machine

You must enable your z/VM guest virtual machine to write monitor records and configure the z/VM system to collect these records.

Procedure

Perform these steps:

1. Set this option in the z/VM user directory entry of the virtual machine in which the application that uses this device driver is to run:

- OPTION APPLMON

2. Issue the following CP commands to have CP collect the respective types of monitor data:
   - `MONITOR SAMPLE ENABLE APPLDATA ALL`
   - `MONITOR EVENT ENABLE APPLDATA ALL`

   You can log in to the z/VM console to issue the CP commands. These commands must be preceded with #CP. Alternatively, you can use the **vmcp** command for issuing CP commands from your Linux instance.

   See *z/VM: CP Commands and Utilities Reference*, SC24-6268 for information about the CP MONITOR command.

Working with the z/VM *MONITOR record writer

The monitor stream application device driver uses the z/VM CP instruction DIAG X'DC' to write to the z/VM monitor stream. Monitor data must be preceded by a data structure, monwrite_hdr.

See *z/VM: CP Programming Services*, SC24-6272 for more information about the DIAG X'DC' instruction and the different monitor record types (sample, config, event).

The application writes monitor data by passing a monwrite_hdr structure that is followed by monitor data. The only exception is the STOP function, which requires no monitor data. The monwrite_hdr structure, as described in monwriter.h, is filled in by the application. The structure includes the DIAG X'DC' function to be performed, the product identifier, the header length, and the data length.

All records that are written to the z/VM monitor stream begin with a product identifier. This device driver uses the product ID. The product ID is a 16-byte structure of the form ppppppppffnvvrrmm, where:

ppppppp
is a fixed ASCII string, for example, LNXAPPL.

ff
is the application number (hexadecimal number). This number can be chosen by the application. You can reduce the chance of conflicts with other applications, by requesting an application number from the IBM z/VM Performance team at

```
www.ibm.com/vm/perf
```

n
is the record number as specified by the application

vv, rr, and mm
can also be specified by the application. A possible use is to specify version, release, and modification level information, allowing changes to a certain record number when the layout is changed, without changing the record number itself.

The first 7 bytes of the structure (LNXAPPL) are filled in by the device driver when it writes the monitor data record to the CP buffer. The last 9 bytes contain information that is supplied by the application on the `write()` call when writing the data.

The monwrite_hdr structure that must be written before any monitor record data is defined as follows:

```
/* the header the app uses in its write() data */
struct monwrite_hdr {
    unsigned char mon_function;
    unsigned short applid;
    unsigned char record_num;
    unsigned short version;
    unsigned short release;
    unsigned short mod_level;
    unsigned short datalen;
    unsigned char hdrlen;
}__attribute__((packed));
```

The following function code values are defined:

```
/* mon_function values */
#define MONWRITE_START_INTERVAL 0x00 /* start interval recording */
#define MONWRITE_STOP_INTERVAL  0x01 /* stop interval or config recording */
#define MONWRITE_GEN_EVENT      0x02 /* generate event record */
#define MONWRITE_START_CONFIG   0x03 /* start configuration recording */
```

Writing data and stopping data writing

Applications use the `open()`, `write()`, and `close()` calls to work with the z/VM monitor stream.

Before an application can write monitor records, it must issue `open()` to open the device driver. Then, the application must issue `write()` calls to start or stop the collection of monitor data and to write any monitor records to buffers that CP can access.

When the application has finished writing monitor data, it must issue `close()` to close the device driver.

Using the monwrite_hdr structure

The structure monwrite_hdr is used to pass DIAG x'DC' functions and the application-defined product information to the device driver on `write()` calls.

When the application calls `write()`, the data it is writing consists of one or more monwrite_hdr structures. Each structure is followed by monitor data. The only exception is the STOP function, which is not followed by data.

The application can write to one or more monitor buffers. A new buffer is created by the device driver for each record with a unique product identifier. To write new data to an existing buffer, an identical monwrite_hdr structure must precede the new data on the `write()` call.

The monwrite_hdr structure also includes a field for the header length, which is useful for calculating the data offset from the beginning of the header. There is also a field for the data length, which is the length of any monitor data that follows. See `/usr/include/asm/monwriter.h` for the definition of the monwrite_hdr structure.

Chapter 37. Reading z/VM monitor records

z/VM only: z/VM monitor records apply to Linux on z/VM only.

Monitoring software on Linux can access z/VM guest data through the z/VM *MONITOR record reader device driver.

z/VM uses the z/VM monitor system service (*MONITOR) to collect monitor records from agents on its guests. z/VM writes the records to a discontiguous saved segment (DCSS). See *z/VM: Saved Segments Planning and Administration*, SC24-6322 for general information about DCSSs.

The z/VM *MONITOR record reader device driver uses IUCV to connect to *MONITOR and accesses the DCSS as a character device.

For an overview of performance monitoring support, see "Performance monitoring for z/VM guest virtual machines" on page 409.

The z/VM *MONITOR record reader device driver supports the following devices and functions:

- Read access to the z/VM *MONITOR DCSS.
- Reading *MONITOR records for z/VM.
- Access to *MONITOR records as described on

  ```
  www.ibm.com/vm/pubs/ctlblk.html
  ```

- Access to the kernel APPLDATA records from the Linux monitor stream (see Chapter 35, "Writing kernel APPLDATA records," on page 413).

What you should know about the z/VM *MONITOR record reader device driver

The data that is collected by *MONITOR depends on the setup of the monitor stream service.

The z/VM *MONITOR record reader device driver only reads data from the monitor DCSS; it does not control the system service.

z/VM supports only one monitor DCSS. All monitoring software that requires monitor records from z/VM uses the same DCSS to read *MONITOR data. Usually, a DCSS called MONDCSS is already defined and used by existing monitoring software.

If a monitor DCSS is already defined, you must use it. To find out whether a monitor DCSS exists, issue the following CP command from a z/VM guest virtual machine with privilege class E:

```
q monitor
```

The command output also shows the name of the DCSS.

Using kdump: If you use kdump, ensure that the monitor DCSS does not overlap with the storage area `0 - <crashkernel size>`. If the DCSS is already defined and overlaps with the crashkernel storage area, it must be removed and defined again at a suitable location.

Device node

Red Hat Enterprise Linux 9.1 creates a device node for you using udev. The device node is called `/dev/monreader` and is a miscellaneous character device that you can use to access the monitor DCSS.

Further information

- See *z/VM: Saved Segments Planning and Administration*, SC24-6322 for general information about DCSSs.
- See *z/VM: Performance*, SC24-6301 for information about creating a monitor DCSS.
- See *z/VM: CP Commands and Utilities Reference*, SC24-6268 for information about the CP commands that are used in the context of DCSSs and for controlling the z/VM monitor system service.
- For the layout of the monitor records, go to www.ibm.com/vm/pubs/ctlblk.html and click the link to the monitor record format for your z/VM version. Also, see Chapter 35, "Writing kernel APPLDATA records," on page 413.

Setting up the z/VM *MONITOR record reader device driver

You must set up Linux and the z/VM guest virtual machine for accessing an existing monitor DCSS with the z/VM *MONITOR record reader device driver.

Before you begin

Some of the CP commands you use for setting up the z/VM *MONITOR record reader device driver require class E authorization.

Setting up the monitor system service and the monitor DCSS on z/VM is beyond the scope of this information. See *z/VM: Performance*, SC24-6301 for information about creating a monitor DCSS.

Providing the required user directory entries for your z/VM guest

The z/VM guest where your Linux instance is to run must be permitted to establish an IUCV connection to the z/VM *MONITOR system service.

About this task

See *z/VM: CP Commands and Utilities Reference*, SC24-6268 for information about the CP commands that are used in the context of DCSSs and for controlling the z/VM monitor system service.

Procedure

Ensure that the guest entry in the user directory includes the following statement:

```
IUCV *MONITOR
```

If the DCSS is restricted, you also need this statement:

```
NAMESAVE <dcss>
```

where *<dcss>* is the name of the DCSS that is used for the monitor records. You can find out the name of an existing monitor DCSS by issuing the following CP command from a z/VM guest virtual machine with privilege class E:

```
q monitor
```

Assuring that the DCSS is addressable for your Linux instance

The DCSS address range must not overlap with the storage of you z/VM guest virtual machine.

Procedure

To find out the start and end address of the DCSS, issue the following CP command from a z/VM guest virtual machine with privilege class E:

```
q nss map
```

The output gives you the start and end addresses of all defined DCSSs in units of 4-kilobyte pages. For example:

```
00: FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
...
00: 0011 MONDCSS  CPDCSS   N/A     09000  097FF  SC   R   00003 N/A      N/A
...
```

What to do next

If the DCSS overlaps with the guest storage, follow the procedure in "Avoiding overlaps with your guest storage" on page 439.

Specifying the monitor DCSS name

Specify the DCSS name as a module parameter when you load the module.

About this task

By default, the z/VM *MONITOR record reader device driver assumes that the monitor DCSS on z/VM is called MONDCSS. If you want to use a different DCSS name, you must specify it.

Load the monitor read support module with **modprobe** to assure that any other required modules are also loaded. You need IUCV support if you want to use the monitor read support.

monitor stream support module parameter syntax



where *<dcss>* is the name of the DCSS that z/VM uses for the monitor records.

Example

To load the monitor read support module and specify MYDCSS as the DCSS issue:

```
modprobe monreader mondcss=mydcss
```

Working with the z/VM *MONITOR record reader

You can open the z/VM *MONITOR record character device to read records from it.

This section describes how to work with the monitor read support.

Opening and closing the character device

Only one user can open the character device at any one time. Once you have opened the device, you must close it to make it accessible to other users.

About this task

The open function can fail (return a negative value) with one of the following values for errno:

EBUSY
The device has already been opened by another user.

EIO
No IUCV connection to the z/VM MONITOR system service could be established. An error message with an IPUSER SEVER code is printed into syslog. See *z/VM: Performance*, SC24-6301 for details about the codes.

Once the device is opened, incoming messages are accepted and account for the message limit. If you keep the device open indefinitely, expect to eventually reach the message limit (with error code EOVERFLOW).

Reading monitor records

You can either read in non-blocking mode with polling, or you can read in blocking mode without polling.

About this task

Reading from the device provides a 12-byte monitor control element (MCE), followed by a set of one or more contiguous monitor records (similar to the output of the CMS utility MONWRITE without the 4 K control blocks). The MCE contains information about:

- The type of the following record set (sample/event data)
- The monitor domains contained within it
- The start and end address of the record set in the monitor DCSS

The start and end address can be used to determine the size of the record set. The end address is the address of the last byte of data. The start address is needed to handle "end-of-frame" records correctly (domain 1, record 13), that is, it can be used to determine the record start offset relative to a 4 K page (frame) boundary.

See "Appendix A: *MONITOR" in *z/VM: Performance*, SC24-6301 for a description of the monitor control element layout. For the layout of the monitor records go to www.ibm.com/vm/pubs/ctlblk.html and click the link to the monitor record format for your z/VM version. Also see Chapter 35, “Writing kernel APPLDATA records,” on page 413.

The layout of the data stream that is provided by the monreader device is as follows:

```
...
<0 byte read>
<first MCE>                \
<first set of records>    |...
...                              |- data set
<last MCE>                |
<last set of records>    /
<0 byte read>
...
```

There might be more than one combination of MCE and a corresponding record set within one data set. The end of each data set is indicated by a successful read with a return value of 0 (0 byte read). Received data is not to be considered valid unless a complete record set is read successfully, including the closing 0-Byte read. You are advised to always read the complete set into a user space buffer before processing the data.

When designing a buffer, allow for record sizes up to the size of the entire monitor DCSS, or use dynamic memory allocation. The size of the monitor DCSS will be printed into syslog after loading the module. You can also use the (Class E privileged) CP command `Q NSS MAP` to list all available segments and information about them (see “Assuring that the DCSS is addressable for your Linux instance” on page 424).

Error conditions are indicated by returning a negative value for the number of bytes read. For an error condition, the errno variable can be:

EIO
Reply failed. All data that was read since the last successful read with 0 size is not valid. Data is missing. The application must decide whether to continue reading subsequent data or to exit.

EFAULT
Copy to user failed. All data that was read since the last successful read with 0 size is not valid. Data is missing. The application must decide whether to continue reading subsequent data or to exit.

EAGAIN
Occurs on a non-blocking read if there is no data available at the moment. No data is missing or damaged. Retry or use polling for non-blocking reads.

EOVERFLOW
The message limit is reached. The data that was read since the last successful read with 0 size is valid, but subsequent records might be missing. The application must decide whether to continue reading subsequent data or to exit.

Chapter 38. z/VM recording device driver

z/VM only: The z/VM recording device driver applies to Linux on z/VM only.

The z/VM recording device driver enables Linux on z/VM to read from the CP recording services and, thus, act as a z/VM wide control point.

The z/VM recording device driver uses the z/VM CP RECORDING command to collect records and IUCV to transmit them to the Linux instance.

For general information about CP recording system services, see *z/VM: CP Programming Services*, SC24-6272.

Features

With the z/VM recording device driver, you can read from several CP services and collect records.

In particular, the z/VM recording device driver supports:

- Reading records from the CP error logging service, *LOGREC.
- Reading records from the CP accounting service, *ACCOUNT.
- Reading records from the CP diagnostic service, *SYMPTOM.
- Automatic and explicit record collection (see “Starting and stopping record collection” on page 430).

What you should know about the z/VM recording device driver

You can read records from different recording services, one record at a time.

The z/VM recording device driver is a character device driver that is grouped under the IUCV category of device drivers (see “Device categories” on page 7). There is one device for each recording service. The devices are created for you when the z/VM recording device driver module is loaded.

z/VM recording device nodes

Each recording service has a name that corresponds to the name of the service.

Table 56 on page 429 summarizes the names:

Table 56. z/VM recording device names

z/VM recording service	Standard device name
*LOGREC	logrec
*ACCOUNT	account
*SYMPTOM	symptom

About records

Records for different services are different in details, but follow the same overall structure.

The read function returns one record at a time. If there is no record, the read function waits until a record becomes available.

Each record begins with a 4-byte field that contains the length of the remaining record. The remaining record contains the binary z/VM data followed by the four bytes X'454f5200' to mark the end of the record. These bytes build the zero-terminated ASCII string "EOR", which is useful as an eye catcher.



Figure 99. Record structure

Figure 99 on page 430 illustrates the structure of a complete record as returned by the device. If the buffer assigned to the read function is smaller than the overall record size, multiple reads are required to obtain the complete record.

The format of the z/VM data (*LOGREC) depends on the record type that is described in the common header for error records HDRREC.

For more information about the z/VM record layout, see the *CMS and CP Data Areas and Control Blocks* documentation at

```
www.ibm.com/vm/pubs/ctlblk.html
```

Setting up the z/VM recording device driver

Before you can collect records, you must authorize your z/VM guest virtual machine and load the device driver module.

Procedure

1. Authorize the z/VM guest virtual machine on which your Linux instance runs to:
   - Use the z/VM CP RECORDING command.
   - Connect to the IUCV services to be used: one or more of *LOGREC, *ACCOUNT, and *SYMPTOM.
2. Load the z/VM recording device driver.

   You must load the z/VM recording device driver module before you can work with z/VM recording devices. Load the vmlogrdr module with the **modprobe** command to ensure that any other required modules are loaded in the correct order:

   ```
   # modprobe vmlogrdr
   ```

   There are no module parameters for the z/VM recording device driver.

Working with z/VM recording devices

Typical tasks that you perform with z/VM recording devices include starting and stopping record collection, purging records, and opening and closing devices.

- "Starting and stopping record collection" on page 430
- "Purging existing records" on page 431
- "Querying the z/VM recording status" on page 432
- "Opening and closing devices" on page 433

Starting and stopping record collection

By default, record collection for a particular z/VM recording service begins when the corresponding device is opened and stops when the device is closed.

About this task

You can use a device's `autorecording` attribute to be able to open and close a device without also starting or stopping record collection. You can use a device's `recording` attribute to start and stop record collection regardless of whether the device is opened or not.

You cannot start record collection if a device is open and records already exist. Before you can start record collection for an open device, you must read or purge any existing records for this device (see "Purging existing records" on page 431).

Procedure

To be able to open a device without starting record collection and to close a device without stopping record collection, write `0` to the device's `autorecording` attribute.

To restore the automatic starting and stopping of record collection, write 1 to the device's `autorecording` attribute. Issue a command of this form:

```
# echo <flag> > /sys/bus/iucv/drivers/vmlogrdr/<device>/autorecording
```

where *<flag>* is either 0 or 1, and *<device>* is one of: logrec, symptom, or account.

To explicitly turn on record collection write 1 to the device's `recording` attribute. To explicitly turn off record collection write `0` to the device's `recording` attribute. Issue a command of this form:

```
# echo <flag> > /sys/bus/iucv/drivers/vmlogrdr/<device>/recording
```

where *<flag>* is either 0 or 1, and *<device>* is one of: logrec, symptom, or account.

You can read both the `autorecording` and the `recording` attribute to find the current settings.

Examples

- In this example, first the current setting of the autorecording attribute of the logrec device is checked, then automatic recording is turned off:

```
# cat /sys/bus/iucv/drivers/vmlogrdr/logrec/autorecording
1
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/logrec/autorecording
```

- In this example record collection is started explicitly and later stopped for the account device:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
...
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
```

  To confirm whether recording is on or off, read the `recording_status` attribute as described in "Querying the z/VM recording status" on page 432.

Purging existing records

By default, existing records for a particular z/VM recording service are purged automatically when the corresponding device is opened or closed.

About this task

You can use a device's `autopurge` attribute to prevent records from being purged when a device is opened or closed. You can use a device's `purge` attribute to purge records for a particular device at any time without having to open or close the device.

Procedure

To be able to open or close a device without purging existing records write `0` to the device's `autopurge` attribute. To restore automatic purging of existing records, write `1` to the device's `autopurge` attribute. You can read the `autopurge` attribute to find the current setting. Issue a command of this form:

```
# echo <flag> > /sys/bus/iucv/drivers/vmlogrdr/<device>/autopurge
```

where *<flag>* is either 0 or 1, and *<device>* is one of: `logrec`, `symptom`, or `account`.

To purge existing records for a particular device without opening or closing the device, write `1` to the device's purge attribute. Issue a command of this form:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/<device>/purge
```

where *<device>* is one of: `logrec`, `symptom`, or `account`.

Examples

- In this example, the setting of the autopurge attribute for the logrec device is checked first, then automatic purging is switched off:

```
# cat /sys/bus/iucv/drivers/vmlogrdr/logrec/autopurge
1
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/logrec/autopurge
```

- In this example, the existing records for the symptom device are purged:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/symptom/purge
```

Querying the z/VM recording status

Use the `recording_status` attribute of the z/VM recording device driver representation in sysfs to query the z/VM recording status.

Example

This example runs the z/VM CP command QUERY RECORDING and returns the complete output of that command. This list does not necessarily have an entry for all three services and there might also be entries for other guests.

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
```

This command results in output similar to the following example:

```
RECORDING     COUNT     LMT     USERID     COMMUNICATION
EREP ON       00000000  002     EREP       ACTIVE
ACCOUNT ON    00001774  020     DISKACNT   INACTIVE
SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
ACCOUNT OFF   00000000  020     LINUX31    INACTIVE
```

where the lines represent:

- The service
- The recording status
- The number of queued records
- The number of records that result in a message to the operator
- The guest that is or was connected to that service and the status of that connection

A detailed description of the QUERY RECORDING command can be found in the *z/VM: CP Commands and Utilities Reference*, SC24-6268.

Opening and closing devices

You can open, read, and release the device. You cannot open the device multiple times. Each time the device is opened it must be released before it can be opened again.

About this task

You can use a device's `autorecord` attribute (see "Starting and stopping record collection" on page 430) to enable automatic record collection while a device is open.

You can use a device's `autopurge` attribute (see "Purging existing records" on page 431) to enable automatic purging of existing records when a device is opened and closed.

Scenario: Connecting to the *ACCOUNT service

A typical sequence of tasks is autorecording, turning autorecording off, purging records, and starting recording.

Procedure

1. Query the status of z/VM recording. As root, issue the following command:

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
```

   The results depend on the system, and look similar to the following example:

```
RECORDING     COUNT     LMT     USERID     COMMUNICATION
EREP ON       00000000  002     EREP       ACTIVE
ACCOUNT ON    00001812  020     DISKACNT   INACTIVE
SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
ACCOUNT OFF   00000000  020     LINUX31    INACTIVE
```

2. Open `/dev/account` with an appropriate application.

   This action connects the guest to the *ACCOUNT service and starts recording. The entry for *ACCOUNT on guest LINUX31 changes to ACTIVE and ON:

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status

RECORDING     COUNT     LMT     USERID     COMMUNICATION
EREP ON       00000000  002     EREP       ACTIVE
ACCOUNT ON    00001812  020     DISKACNT   INACTIVE
SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
ACCOUNT ON    00000000  020     LINUX31    ACTIVE
```

3. Switch autopurge and autorecord off:

```
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/autopurge
```

```
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/autorecording
```

4. Close the device by ending the application that reads from it and check the recording status.

   While the connection is INACTIVE, RECORDING is still ON:

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING     COUNT     LMT     USERID     COMMUNICATION
EREP ON       00000000  002     EREP       ACTIVE
ACCOUNT ON    00001812  020     DISKACNT   INACTIVE
SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
ACCOUNT ON    00000000  020     LINUX31    INACTIVE
```

5. The next status check shows that some event created records on the *ACCOUNT queue:

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING     COUNT     LMT     USERID     COMMUNICATION
EREP ON       00000000  002     EREP       ACTIVE
ACCOUNT ON    00001821  020     DISKACNT   INACTIVE
SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
ACCOUNT ON    00000009     020     LINUX31    INACTIVE
```

6. Switch recording off:

```
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
```

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING     COUNT     LMT     USERID     COMMUNICATION
EREP ON       000000000 002     EREP       ACTIVE
ACCOUNT ON    00001821  020     DISKACNT   INACTIVE
SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
ACCOUNT OFF   00000009  020     LINUX31    INACTIVE
```

7. Try to switch it on again, and check whether it worked by checking the recording status:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
```

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING     COUNT     LMT     USERID     COMMUNICATION
EREP ON       000000000 002     EREP       ACTIVE
ACCOUNT ON    00001821  020     DISKACNT   INACTIVE
SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
ACCOUNT OFF   00000009  020     LINUX31    INACTIVE
```

   Recording did not start, in the message logs you might find a message:

```
 vmlogrdr: recording response: HCPCRC8087I Records are queued for user LINUX31 on the
*ACCOUNT recording queue and must be purged or retrieved before recording can be turned on.
```

   This kernel message has priority 'debug' so it might not be written to any of your log files.

8. Now remove all the records on your *ACCOUNT queue either by starting an application that reads them from `/dev/account` or by explicitly purging them:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/account/purge
```

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING     COUNT     LMT     USERID     COMMUNICATION
EREP ON       00000000  002     EREP       ACTIVE
ACCOUNT ON    00001821  020     DISKACNT   INACTIVE
SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
ACCOUNT OFF   00000000  020     LINUX31    INACTIVE
```

9. Now start recording and check status again:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
```

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING     COUNT      LMT     USERID     COMMUNICATION
EREP ON       00000000   002     EREP       ACTIVE
ACCOUNT ON    00001821   020     DISKACNT   INACTIVE
SYMPTOM ON    00000000   002     OPERSYMP   ACTIVE
ACCOUNT ON    00000000      020     LINUX31    INACTIVE
```

Chapter 39. z/VM unit record device driver

z/VM only: The z/VM unit record device driver applies to Linux on z/VM only.

The z/VM unit record device driver provides Linux on z/VM with access to virtual unit record devices. Unit record devices comprise punch card readers, card punches, and line printers.

Linux access is limited to virtual unit record devices with default device types (2540 for reader and punch, 1403 for printer).

To write Linux files to the virtual punch or printer (that is, to the corresponding spool file queues) or to receive z/VM reader files (for example CONSOLE files) to Linux files, use the **vmur** command that is part of the s390utils RPM (see "vmur - Work with z/VM spool file queues" on page 747).

What you should know about the z/VM unit record device driver

The z/VM unit record device driver is compiled as a separate module, vmur.

To load the vmur module automatically at boot time, see the section on persistent module loading in *Configuring basic system settings* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_basic_system_settings/)

When the vmur module is loaded, it registers a character device. The following device nodes are created for a unit record device when it is set online:

- Reader: `/dev/vmrdr-0.0.<device_number>`
- Punch: `/dev/vmpun-0.0.<device_number>`
- Printer: `/dev/vmprt-0.0.<device_number>`

Working with z/VM unit record devices

After loading the vmur module, the required virtual unit record devices must be set online.

Procedure

Set the virtual unit record devices online.

For example, to set the devices with device bus-IDs 0.0.000c, 0.0.000d, and 0.0.000e online, issue:

```
# chzdev -e -a 0.0.000c-0.0.000e
```

Alternatively, use **chccwdev**:

```
# chccwdev -e 0.0.000c-0.0.000e
```

What to do next

You can now use the **vmur** command to work with the devices ("vmur - Work with z/VM spool file queues" on page 747).

If you want to unload the vmur module, close all unit record device nodes. Attempting to unload the module while a device node is open results in error message `Module vmur is in use`. You can unload the vmur module, for example, by issuing `modprobe -r`.

Serialization is implemented per device; only one process can open a particular device node at any one time.

Chapter 40. z/VM DCSS device driver

z/VM only: The z/VM DCSS device driver applies to Linux on z/VM only.

The z/VM discontiguous saved segments (DCSS) device driver provides disk-like fixed block access to z/VM discontiguous saved segments.

In particular, the DCSS device driver facilitates implementing a read-write RAM disk that can be shared among multiple Linux instances that run as guests of the same z/VM system. For example, such a RAM disk can provide a shared file system.

For information about DCSS, see *z/VM: Saved Segments Planning and Administration*, SC24-6322

What you should know about DCSS

The DCSS device names and nodes adhere to a naming scheme. There are different modes and options for mounting a DCSS.

Important: DCSSs occupy spool space. Be sure that you have enough spool space available (multiple times the DCSS size).

DCSS naming scheme

The standard device names are of the form dcssblk*<n>*, where *<n>* is the corresponding minor number.

The first DCSS device that is added is assigned the name dcssblk0, the second dcssblk1, and so on. When a DCSS device is removed, its device name and corresponding minor number are free and can be reassigned. A DCSS device that is added always receives the lowest free minor number.

DCSS device nodes

User space programs access DCSS devices by device nodes. Red Hat Enterprise Linux 9.1 creates standard DCSS device nodes for you.

Standard DCSS device nodes have the form `/dev/<device_name>`, for example:

```
/dev/dcssblk0
/dev/dcssblk1
...
```

Accessing a DCSS in exclusive-writable mode

You must access a DCSS in exclusive-writable mode, for example, to create or update the DCSS.

To access a DCSS in exclusive-writable mode at least one of the following conditions must apply:

- The DCSS fits below the maximum definable address space size of the z/VM guest virtual machine.

  For large read-only DCSS, you can use suitable guest sizes to restrict exclusive-writable access to a specific z/VM guest virtual machine with a sufficient maximum definable address space size.
- The z/VM user directory entry for the z/VM guest virtual machine includes a NAMESAVE statement for the DCSS. See *z/VM: CP Planning and Administration*, SC24-6271 for more information about the NAMESAVE statement.
- The DCSS has been defined with the LOADNSHR operand.

  See *z/VM: CP Commands and Utilities Reference*, SC24-6268 for information about the LOADNSHR operand.

  See “DCSS options” on page 438 about saving DCSSs with the LOADNSHR operand or with other optional properties.

DCSS options

The z/VM DCSS device driver always saves DCSSs with default properties. Any previously defined options are removed.

For example, a DCSS that was defined with the LOADNSHR operand loses this property when it is saved with the z/VM DCSS device driver.

To save a DCSS with optional properties, you must unmount the DCSS device, then use the CP DEFSEG and SAVESEG commands to save the DCSS. See "Workaround for saving DCSSs with optional properties" on page 443 for an example.

See *z/VM: CP Commands and Utilities Reference*, SC24-6268 for information about DCSS options.

Setting up the DCSS device driver

Before you can load and use DCSSs, you must load the DCSS block device driver. Use the `segments` module parameter to load one or more DCSSs when the DCSS device driver is loaded.

DCSS module parameter syntax



<dcss>
specifies the name of a DCSS as defined on the z/VM hypervisor. The specification for *<dcss>* is converted from ASCII to uppercase EBCDIC.

:
the colon (:) separates DCSSs within a set of DCSSs to be mapped to a single DCSS device. You can map a set of DCSSs to a single DCSS device if the DCSSs in the set form a contiguous memory space.

You can specify the DCSSs in any order. The name of the first DCSS you specify is used to represent the device under `/sys/devices/dcssblk`.

`(local)`
sets the access mode to exclusive-writable after the DCSS or set of DCSSs are loaded.

,
the comma (,) separates DCSS devices.

Examples

The following command loads the DCSS device driver and three DCSSs: DCSS1, DCSS2, and DCSS3. DCSS2 is accessed in exclusive-writable mode.

```
# modprobe dcssblk segments="dcss1,dcss2(local),dcss3"
```

The following command loads the DCSS device driver and four DCSSs: DCSS4, DCSS5, DCSS6, and DCSS7. The device driver creates two DCSS devices. One device maps to DCSS4 and the other maps to the combined storage space of DCSS5, DCSS6, and DCSS7 as a single device.

```
# modprobe dcssblk segments="dcss4,dcss5:dcss6:dcss7"
```

Avoiding overlaps with your guest storage

Ensure that your DCSSs do not overlap with the memory of your z/VM guest virtual machine (guest storage).

Using kdump: If you use kdump, a DCSS and its corresponding storage gap must not overlap with the storage area `0 - <crashkernel size>`.

About this task

To find the start and end addresses of the DCSSs, enter the following CP command; this command requires privilege class E:

```
#cp q nss map
```

the output gives you the start and end addresses of all defined DCSSs in units of 4-kilobyte pages:

```
00: FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
...
00: 0011 MONDCSS  CPDCSS   N/A     09000  097FF  SC   R   00003 N/A      N/A
...
```

If all DCSSs that you intend to access are located above the guest storage, you do not need to take any action.

Procedure

If any DCSS that you intend to access with your guest machine overlaps with the guest storage, redefine the guest storage. Define two or more discontiguous storage extents such that the storage gap with the lowest address range covers the address ranges of all your DCSSs.

Note:

- You cannot place a DCSS into a storage gap other than the storage gap with the lowest address range.
- A z/VM guest that was defined with one or more storage gaps cannot access a DCSS above the guest storage.

From a CMS session, use the DEF STORE command to define your guest storage as discontiguous storage extents. Ensure that the storage gap between the extents covers all your DCSSs' address ranges. Issue a command of this form:

```
DEF STOR CONFIG 0.<storage_gap_begin> <storage_gap_end>.<storage above gap>
```

where:

<storage_gap_begin>
: is the lower limit of the storage gap. This limit must be at or below the lowest address of the DCSS with the lowest address range.

  Because the lower address ranges are needed for memory management functions, make the lower limit at least 128 MB. The lower limit for the DCSS increases with the total memory size. Although 128 MB is not an exact value, it is an approximation that is sufficient for most cases.

<storage_gap_end>
: is the upper limit of the storage gap. The upper limit must be above the upper limit of the DCSS with the highest address range.

<storage above gap>
: is the amount of storage above the storage gap. The total guest storage is *<storage_gap_begin>* + *<storage above gap>*.

All values can be suffixed with M to provide the values in megabyte. See *z/VM: CP Commands and Utilities Reference*, SC24-6268 for more information about the DEF STORE command.

Example

To make a DCSS that starts at 144 MB and ends at 152 MB accessible to a z/VM guest with 512 MB guest storage:

```
DEF STORE CONFIG 0.140M 160M.372M
```

This specification is one example of how a suitable storage gap can be defined. In this example, the storage gap covers 140 - 160 MB and, thus, the entire DCSS range. The total guest storage is 140 MB + 372 MB = 512 MB.

Working with DCSS devices

Typical tasks for working with DCSS devices include mapping DCSS representations in z/VM and Linux, adding and removing DCSSs, and accessing and updating DCSS contents.

Adding a DCSS device

Storage gaps or overlapping storage ranges can prevent you from adding a DCSS.

Before you begin

- You must have set up one or more DCSSs on z/VM and know their names on z/VM.
- If you use the watchdog device driver, turn off the watchdog before adding a DCSS device. Adding a DCSS device can result in a watchdog timeout if the watchdog is active.
- You cannot concurrently access overlapping DCSSs.
- You cannot access a DCSS that overlaps with your z/VM guest virtual storage (see "Avoiding overlaps with your guest storage" on page 439).
- On z/VM guest virtual machines with one or more storage gaps, you cannot add a DCSS that is above the guest storage.
- On z/VM guest virtual machines with multiple storage gaps, you cannot add a DCSS unless it fits in the storage gap with the lowest address range.

Procedure

To add a DCSS device, enter a command of this form:

```
# echo <dcss-list> > /sys/devices/dcssblk/add
```

<dcss-list>
: the name, as defined on z/VM, of a single DCSS or a colon (:) separated list of names of DCSSs to be mapped to a single DCSS device. You can map a set of DCSSs to a single DCSS device if the DCSSs in the set form a contiguous memory space. You can specify the DCSSs in any order. The name of the first DCSS you specify is used to represent the device under `/sys/devices/dcssblk`.

Examples

To add a DCSS called "MYDCSS" enter:

```
# echo MYDCSS > /sys/devices/dcssblk/add
```

To add three contiguous DCSSs "MYDCSS1", "MYDCSS2", and "MYDCSS3" as a single device, enter:

```
# echo MYDCSS2:MYDCSS1:MYDCSS3 > /sys/devices/dcssblk/add
```

In sysfs, the resulting device is represented as `/sys/devices/dcssblk/MYDCSS2`.

Listing the DCSSs that map to a particular device

Read the `seglist` sysfs attribute to find out how DCSS devices in Linux map to DCSSs as defined in z/VM.

Procedure

To list the DCSSs that map to a DCSS device, issue a command of this form:

```
# cat /sys/devices/dcssblk/<dcss-name>/seglist
```

where *<dcss-name>* is the DCSS name that represents the DCSS device.

Examples

In this example, DCSS device `MYDCSS` maps to a single DCSS, "MYDCSS".

```
# cat /sys/devices/dcssblk/MYDCSS/seglist
MYDCSS
```

In this example, DCSS device `MYDCSS2` maps to three contiguous DCSSs, "MYDCSS1", "MYDCSS2", and "MYDCSS3".

```
# cat /sys/devices/dcssblk/MYDCSS2/seglist
MYDCSS2
MYDCSS1
MYDCSS3
```

Finding the minor number for a DCSS device

When you add a DCSS device, a minor number is assigned to it.

About this task

Unless you use dynamically created device nodes as provided by udev, you might need to know the minor device number that was assigned to the DCSS (see “DCSS naming scheme” on page 437).

When you add a DCSS device, a directory of this form is created in sysfs:

```
/sys/devices/dcssblk/<dcss-name>
```

where *<dcss-name>* is the DCSS name that represents the DCSS device.

This directory contains a symbolic link, block, that helps you to find out the device name and minor number. The link is of the form `../../../block/dcssblk<n>`, where `dcssblk<n>` is the device name and *<n>* is the minor number.

Example

To find out the minor number that is assigned to a DCSS device that is represented by the directory `/sys/devices/dcssblk/MYDCSS` issue:

```
# readlink /sys/devices/dcssblk/MYDCSS/block
../../../block/dcssblk0
```

In the example, the assigned minor number is `0`.

Setting the access mode

You might want to access the DCSS device with write access to change the content of the DCSS or set of DCSSs that map to the device.

About this task

There are two possible write access modes to the DCSS device:

shared
: In the shared mode, changes to DCSSs are immediately visible to all z/VM guests that access them. Shared is the default.

  Note: Writing to a shared DCSS device bears the same risks as writing to a shared disk.

exclusive-writable
: In the exclusive-writable mode you write to private copies of DCSSs. A private copy is writable, even if the original DCSS is read-only. Changes that you make to a private copy are invisible to other guests until you save the changes (see “Saving updates to a DCSS or set of DCSSs” on page 443).

  After saving the changes to a DCSS, all guests that open the DCSS access the changed copy. z/VM retains a copy of the original DCSS for those guests that continue accessing it, until the last guest stops using it.

  To access a DCSS in the exclusive-writable mode the maximum definable storage size of your z/VM virtual machine must be above the upper limit of the DCSS. Alternatively, suitable authorizations must be in place (see “Accessing a DCSS in exclusive-writable mode” on page 437).

For either access mode the changes are volatile until they are saved (see “Saving updates to a DCSS or set of DCSSs” on page 443).

Procedure

Issue a command of this form:

```
# echo <flag> > /sys/devices/dcssblk/<dcss-name>/shared
```

where *<dcss-name>* is the DCSS name that represents the DCSS device.

You can read the shared attribute to find out the current access mode.

Example

To find out the current access mode of a DCSS device represented by the DCSS name "MYDCSS":

```
# cat /sys/devices/dcssblk/MYDCSS/shared
1
```

1 means that the current access mode is shared. To set the access mode to exclusive-writable issue:

```
# echo 0 > /sys/devices/dcssblk/MYDCSS/shared
```

Saving updates to a DCSS or set of DCSSs

Use the `save` sysfs attribute to save DCSSs that were defined without optional properties.

Before you begin

- Saving a DCSS as described in this section results in a default DCSS, without optional properties. For DCSSs that were defined with options (see "DCSS options" on page 438), see "Workaround for saving DCSSs with optional properties" on page 443.
- If you use the watchdog device driver, turn off the watchdog before saving updates to DCSSs. Saving updates to DCSSs can result in a watchdog timeout if the watchdog is active.
- Do not place save requests before you have accessed the DCSS device.

Procedure

Issue a command of this form:

```
# echo 1 > /sys/devices/dcssblk/<dcss-name>/save
```

where *<dcss-name>* is the DCSS name that represents the DCSS device.

Saving is delayed until you close the device.

You can check if a save request is waiting to be performed by reading the contents of the `save` attribute.

You can cancel a save request by writing `0` to the `save` attribute.

Example

To check if a save request exists for a DCSS device that is represented by the DCSS name "MYDCSS":

```
# cat /sys/devices/dcssblk/MYDCSS/save
0
```

The `0` means that no save request exists. To place a save request issue:

```
# echo 1 > /sys/devices/dcssblk/MYDCSS/save
```

To purge an existing save request issue:

```
# echo 0 > /sys/devices/dcssblk/MYDCSS/save
```

Workaround for saving DCSSs with optional properties

If you need a DCSS that is defined with special options, you must use a workaround to save the DCSSs.

Before you begin

Important: This section applies to DCSSs with special options only. The workaround in this section is error-prone and requires utmost care. Erroneous parameter values for the described CP commands can render a DCSS unusable. Use this workaround only if you really need a DCSS with special options.

Procedure

Perform the following steps to save a DCSS with optional properties:

1. Unmount the DCSS.

   Example: Enter this command to unmount a DCSS with the device node `/dev/dcssblk0`:

```
# umount /dev/dcssblk0
```

2. Use the CP DEFSEG command to newly define the DCSS with the required properties.

   Example: Enter this command to newly define a DCSS, `mydcss`, with the range `80000-9ffff`, segment type `sr`, and the `loadnshr` operand:

   ```
   # vmcp defseg mydcss 80000-9ffff sr loadnshr
   ```

   Note: If your DCSS device maps to multiple DCSSs as defined to z/VM, you must perform this step for each DCSS. Be sure to specify the command correctly with the correct address ranges and segment types. Incorrect specifications can render the DCSS unusable.
3. Use the CP SAVESEG command to save the DCSS.

   Example: Enter this command to save a DCSS `mydcss`:

   ```
   # vmcp saveseg mydcss
   ```

   Note: If your DCSS device maps to multiple DCSSs as defined to z/VM, you must perform this step for each DCSS. Omitting this step for individual DCSSs can render the DCSS device unusable.

Reference

See *z/VM: CP Commands and Utilities Reference*, SC24-6268 for details about the DEFSEG and SAVESEG CP commands.

Removing a DCSS device

Use the `remove` sysfs attribute to remove a DCSS from Linux.

Before you begin

A DCSS device can only be removed when it is not in use.

Procedure

You can remove the DCSS or set of DCSSs that are represented by a DCSS device from your Linux system by issuing a command of this form:

```
# echo <dcss-name> > /sys/devices/dcssblk/remove
```

where *<dcss-name>* is the DCSS name that represents the DCSS device.

Example

To remove a DCSS device that is represented by the DCSS name "MYDCSS" issue:

```
# echo MYDCSS > /sys/devices/dcssblk/remove
```

What to do next

If you have created your own device nodes, you can keep the nodes for reuse. Be aware that the major number of the device might change when you unload and reload the DCSS device driver. When the major number of your device has changed, existing nodes become unusable.

Scenario: Changing the contents of a DCSS

Before you can change the contents of a DCSS, you must add the DCSS to Linux, access it in a writable mode, and mount the file system on it.

About this task

The scenario that follows is based on these assumptions:

- The Linux instance runs as a z/VM guest with class E user privileges.
- A DCSS is set up and can be accessed in exclusive-writable mode by the Linux instance.
- The DCSS does not overlap with the guest's main storage.
- There is only a single DCSS named "MYDCSS".
- The DCSS block device driver is set up and ready to be used.

The description in this scenario can readily be extended to changing the content of a set of DCSSs that form a contiguous memory space. The only change to the procedure would be mapping the DCSSs in the set to a single DCSS device in step “1” on page 445. The assumptions about the set of DCSSs would be:

- The contiguous memory space that is formed by the set does not overlap with the guest storage.
- Only the DCSSs in the set are added to the Linux instance.

Procedure

Perform the following steps to change the contents of a DCSS:

1. Add the DCSS to the block device driver.

```
# echo MYDCSS > /sys/devices/dcssblk/add
```

2. Ensure that there is a device node for the DCSS block device.

   If it is not created for you, for example by udev, create it yourself.

   a) Find out the major number that is used for DCSS block devices. Read `/proc/devices`:

```
# cat /proc/devices
...
Block devices
...
254 dcssblk
...
```

   The major number in the example is 254.

   b) Find out the minor number that is used for MYDCSS.

   If MYDCSS is the first DCSS to be added, the minor number is 0. To be sure, you can read a symbolic link that is created when the DCSS is added.

```
# readlink /sys/devices/dcssblk/MYDCSS/block
../../../block/dcssblk0
```

   The trailing 0 in the standard device name dcssblk0 indicates that the minor number is, indeed, 0.

   c) Create the node with the **mknod** command:

```
# mknod /dev/dcssblk0 b 254 0
```

3. Set the access mode to exclusive-write.

```
# echo 0 > /sys/devices/dcssblk/MYDCSS/shared
```

4. Mount the file system in the DCSS on a spare mount point.

```
# mount /dev/dcssblk0 /mnt
```

5. Update the data in the DCSS.
6. Create a save request to save the changes.

```
# echo 1 > /sys/devices/dcssblk/MYDCSS/save
```

7. Unmount the file system.

```
# umount /mnt
```

   The changes to the DCSS are now saved. When the last z/VM guest stops accessing the old version of the DCSS, the old version is discarded. Each guest that opens the DCSS accesses the updated copy.

8. Remove the device.

```
# echo MYDCSS > /sys/devices/dcssblk/remove
```

9. Optional: If you have created your own device node, you can clean it up.

```
# rm -f /dev/dcssblk0
```

Chapter 41. z/VM CP interface device driver

z/VM only: The z/VM CP interface device driver applies to Linux on z/VM only.

Using the z/VM CP interface device driver (vmcp), you can send control program (CP) commands to the z/VM hypervisor and display the response.

The vmcp device driver works only for Linux on z/VM.

What you should know about the z/VM CP interface

The z/VM CP interface driver (vmcp) uses the CP diagnose X'08' to send commands to CP and to receive responses. The behavior is similar but not identical to #CP on a 3270 or 3215 console.

Using the z/VM CP interface

There are two ways of using the z/VM CP interface device driver:

- Through the `/dev/vmcp` device node
- Through a user space tool (see “vmcp - Send CP commands to the z/VM hypervisor” on page 745)

You must load the vmcp module before you can use vmcp. If your Linux guest runs under z/VM, you can configure the startup scripts to load the vmcp kernel module automatically during boot. See the section on persistent module loading in *Configuring basic system settings* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_basic_system_settings/) for information about how to do this.

The vmcp device driver works only for Linux on z/VM and cannot be loaded if the Linux system runs in an LPAR.

Differences between vmcp and a 3270 or 3215 console

Most CP commands behave identically with vmcp and on a 3270 or 3215 console. However, some commands show a different behavior:

- Diagnose X'08' (see *z/VM: CP Programming Services*, SC24-6272) requires you to specify a response buffer with the command. Because the response size is not known in advance, the default response buffer of vmcp might be too small and the response truncated.
- On a 3270 or 3215 console, the CP command is executed on virtual CPU 0. The vmcp device driver uses the CPU that is scheduled by the Linux kernel. For CP commands that depend on the CPU number (like trace), specify the CPU, for example: `cpu 3 trace count`.
- Some CP commands do not return specific error or status messages through diagnose X'08'. These messages are only returned on a 3270 or 3215 console. For example, the command `vmcp link user1 1234 123 mw` might return the message `DASD 123 LINKED R/W` in a 3270 or 3215 console. This message is not displayed if the CP command is issued with vmcp. For details, see the z/VM help system or *z/VM: CP Commands and Utilities Reference*, SC24-6268.

Using the device node

You can send a command to z/VM CP by writing to the vmcp device node.

Observe the following rules for writing to the device node:

- Omit the newline character at the end of the command string. For example, use **echo -n** if you are writing directly from a terminal session.
- Write the command in the same case as required on z/VM.
- Escape characters that need escaping in the environment where you issue the command.

Example

The following command attaches a device to your z/VM guest virtual machine. The asterisk (*) is escaped to prevent the command shell from interpreting it.

```
# echo -n ATTACH 1234 \* > /dev/vmcp
```

Application programmers

You can also use the vmcp device node directly from an application by using open, write (to issue the command), read (to get the response), ioctl (to get and set status), and close. The following ioctls are supported:

Table 57. The vmcp ioctls

Name	Code definition	Description
VMCP_GETCODE	_IOR (0x10, 1, int)	Queries the return code of z/VM.
VMCP_SETBUF	_IOW(0x10, 2, int)	Sets the buffer size (the device driver has a default of 4 KB; vmcp calls this ioctl to set it to 8 KB instead).
VMCP_GETSIZE	_IOR(0x10, 3, int)	Queries the size of the response.

Chapter 42. z/VM special messages uevent support

z/VM only: The z/VM CP special messages uevent support applies to Linux on z/VM only.

The smsgiucv_app kernel device driver receives z/VM CP special messages (SMSG) and delivers these messages to user space as udev events (uevents).

The device driver receives only messages that start with APP. The generated uevents contain the message sender and content as environment variables (see Figure 100 on page 449).



Figure 100. CP special messages as uevents in user space

You can restrict the received special messages to a particular z/VM user ID. CP special messages are discarded if the specified sender does not match the sender of the CP special message.

Setting up the CP special message device driver

Configure the CP special message device driver when you load the device driver module.

The z/VM user ID does not require special authorizations to receive CP special messages. CP special messages can be issued from the local z/VM guest virtual machine or from other guest virtual machines. You can issue special messages from Linux or from a CMS or CP session.

Load the device driver module with the **modprobe** command.

smsgiucv_app syntax



Where:

sender = *<user_ID>*
permits CP special messages from the specified z/VM user ID only. CP special messages are discarded if the specified sender does not match the sender of the CP special message. If the **sender=** option is empty or not set, CP special messages are accepted from any z/VM user ID.

Lowercase characters are converted to uppercase.

To receive messages from several user IDs leave the sender= parameter empty, or do not specify it, and then filter with udev rules (see “Example udev rule” on page 451).

To load the smsgiucv_app module automatically at boot time, see the section on persistent module loading in *Configuring basic system settings* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_basic_system_settings/).

Working with CP special messages

You might have to send, access, or respond to CP special messages.

Sending CP special messages

Issue a CP SMSG command from a CP or CMS session or from Linux to send a CP special message.

Procedure

To send a CP special message to LXGUEST1 from Linux, enter a command of the following form:

```
# vmcp SMSG LXGUEST1 APP "<message text>"
```

To send a CP special message to LXGUEST1, enter the following command from a CP or CMS session:

```
#CP SMSG LXGUEST1 APP <message text>
```

The special messages cause uevents to be generated. See “Writing udev rules for handling CP special messages” on page 451 for information about handling the uevents.

Accessing CP special messages through uevent environment variables

A uevent for a CP special message contains environment variables that you can use to access the message.

SMSG_ID
Specifies the message prefix. The SMSG_ID environment variable is always set to APP, which is the prefix that is assigned to the smsgiucv_app device driver.

SMSG_SENDER
Specifies the z/VM user ID that sent the CP special message.

Use SMSG_SENDER in udev rules for filtering the z/VM user ID if you want to accept CP special messages from different senders. All alphabetic characters in the z/VM user ID are uppercase characters.

SMSG_TEXT
Contains the message text of the CP special message. The APP prefix and leading white spaces are removed.

Writing udev rules for handling CP special messages

When using the CP special messages device driver, CP special messages trigger uevents.

change events

The smsgiucv_app device driver generates change uevents for each CP special message that is received.

For example, the special message:

```
#CP SMSG LXGUEST1 APP THIS IS A TEST MESSAGE
```

might trigger the following uevent:

```
UEVENT[1263487666.708881] change /devices/iucv/smsgiucv_app (iucv)
ACTION=change
DEVPATH=/devices/iucv/smsgiucv_app
SUBSYSTEM=iucv
SMSG_SENDER=MAINT
SMSG_ID=APP
SMSG_TEXT=THIS IS A TEST MESSAGE
DRIVER=SMSGIUCV
SEQNUM=1493
```

add and remove events

In addition to the change event for received CP special messages, generic add and remove events are generated when the module is loaded or unloaded, for example:

```
UEVENT[1263487583.511146] add /module/smsgiucv_app (module)
ACTION=add
DEVPATH=/module/smsgiucv_app
SUBSYSTEM=module
SEQNUM=1487

UEVENT[1263487583.514622] add /devices/iucv/smsgiucv_app (iucv)
ACTION=add
DEVPATH=/devices/iucv/smsgiucv_app
SUBSYSTEM=iucv
DRIVER=SMSGIUCV
SEQNUM=1488

UEVENT[1263487628.955149] remove /devices/iucv/smsgiucv_app (iucv)
ACTION=remove
DEVPATH=/devices/iucv/smsgiucv_app
SUBSYSTEM=iucv
SEQNUM=1489

UEVENT[1263487628.957082] remove /module/smsgiucv_app (module)
ACTION=remove
DEVPATH=/module/smsgiucv_app
SUBSYSTEM=module
SEQNUM=1490
```

With the information from the uevents, you can create custom udev rules to trigger actions that depend on the settings of the SMSG_* environment variables (see “Accessing CP special messages through uevent environment variables” on page 450).

For your udev rules, use the add and remove uevents to initialize and clean up resources. To handle CP special messages, write udev rules that match change uevents. For more information about writing udev rules, see the udev man page.

Example udev rule

The udev rules that process CP special messages identify particular messages and define one or more specific actions as a response.

The following example shows how to process CP special messages by using udev rules. The example contains rules for actions, one for all senders and one for the MAINT, OPERATOR, and LNXADM senders only.

The rules are contained in a block that matches uevents from the smsgiucv_app device driver. If there is no match, processing ends:

```
#
# Sample udev rules for processing CP special messages.
#
#
DEVPATH!="*/smsgiucv_app", GOTO="smsgiucv_app_end"

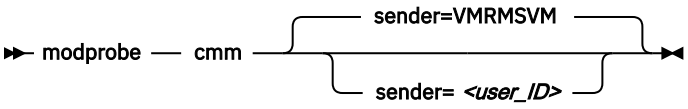
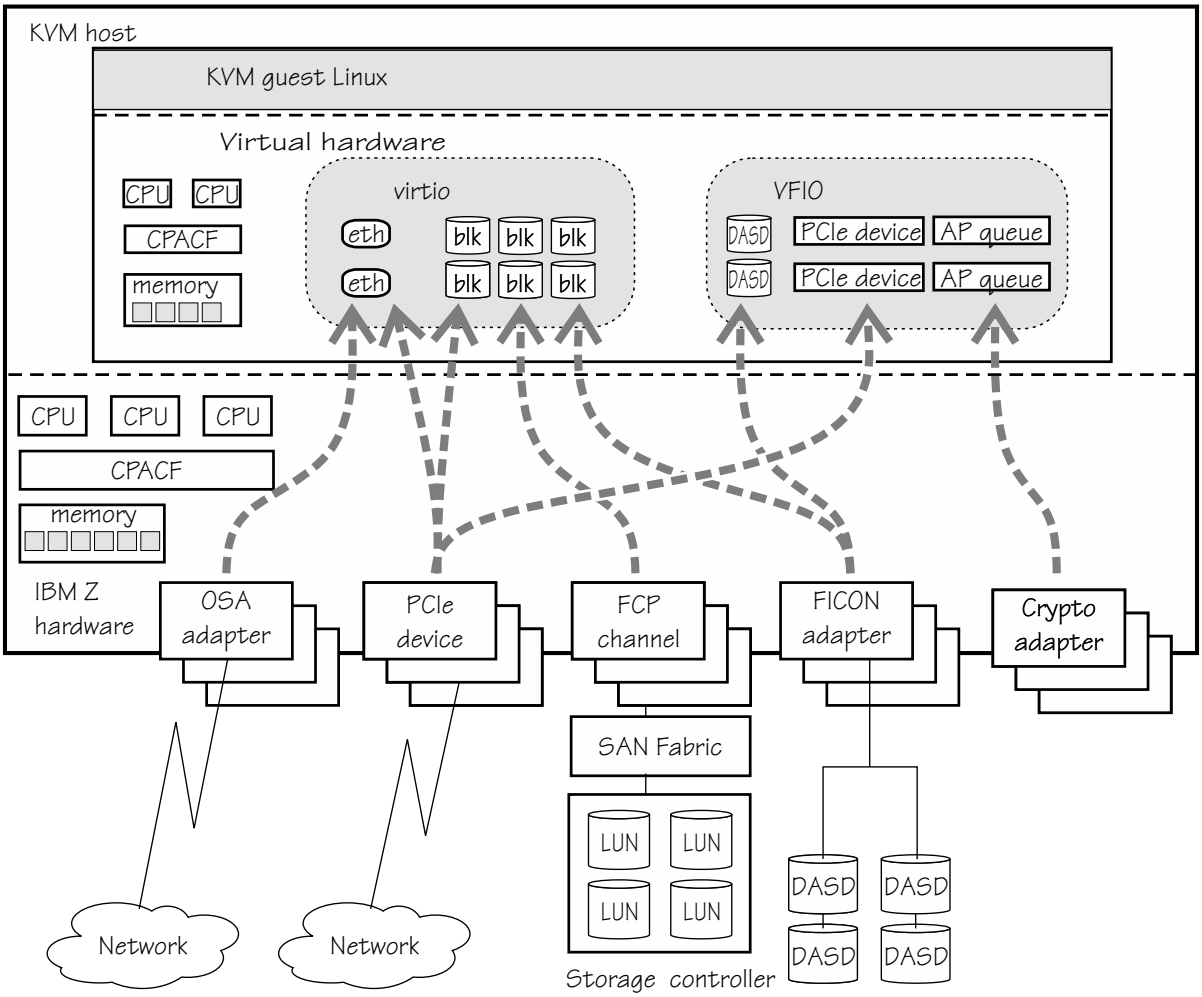
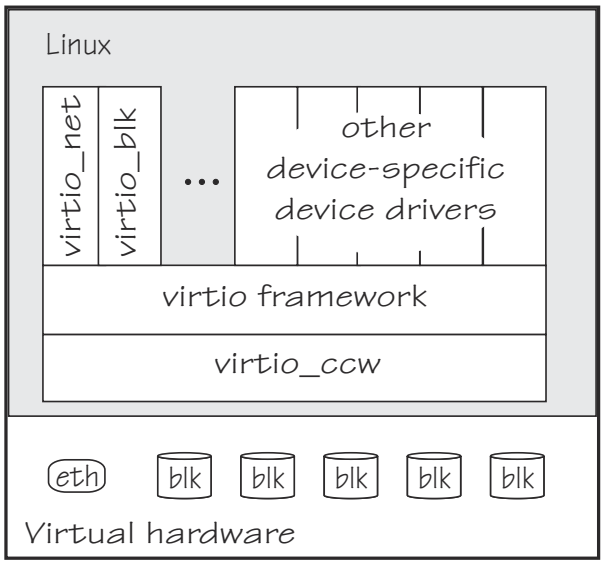
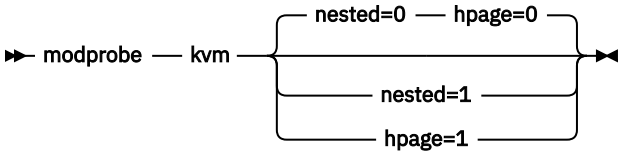
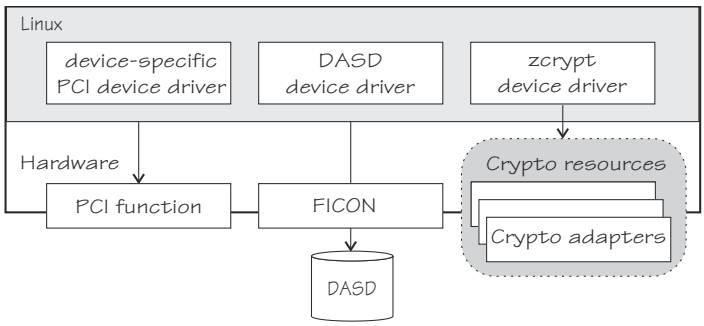
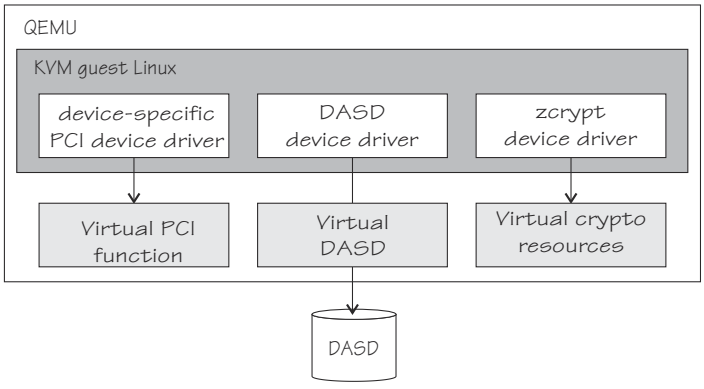
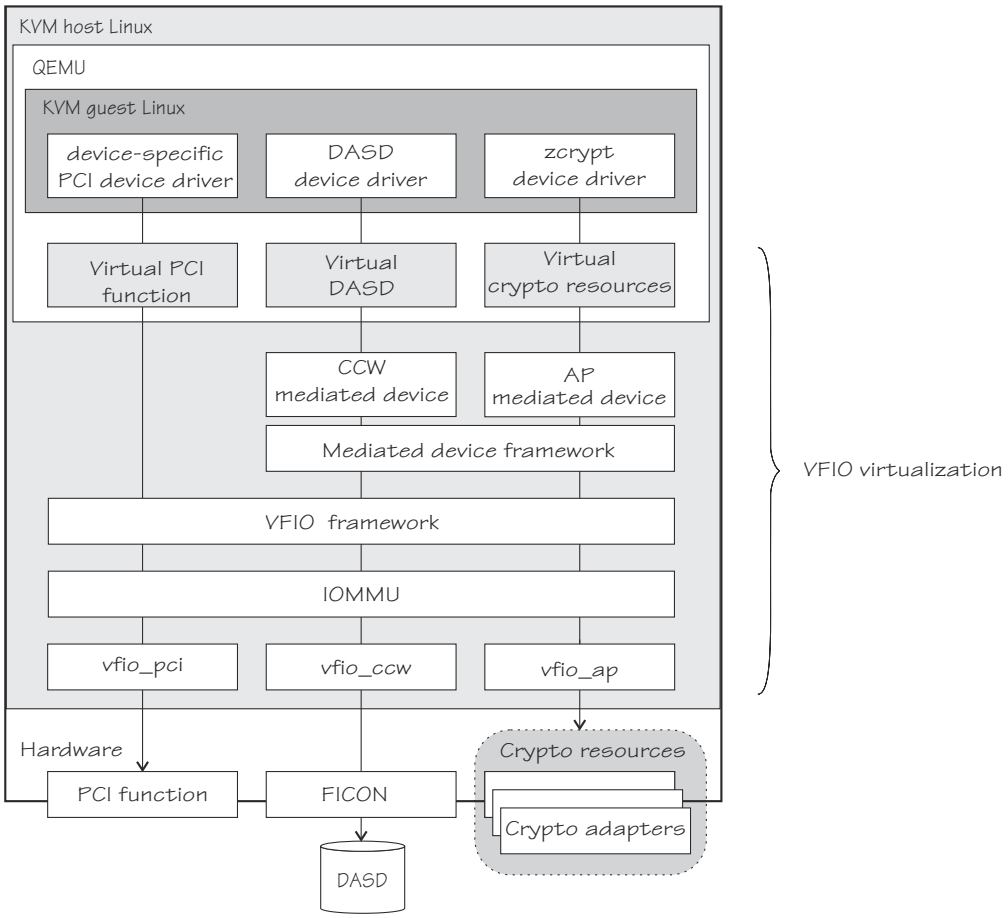
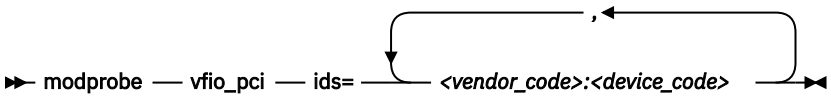
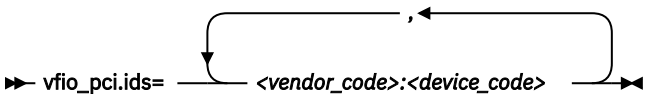
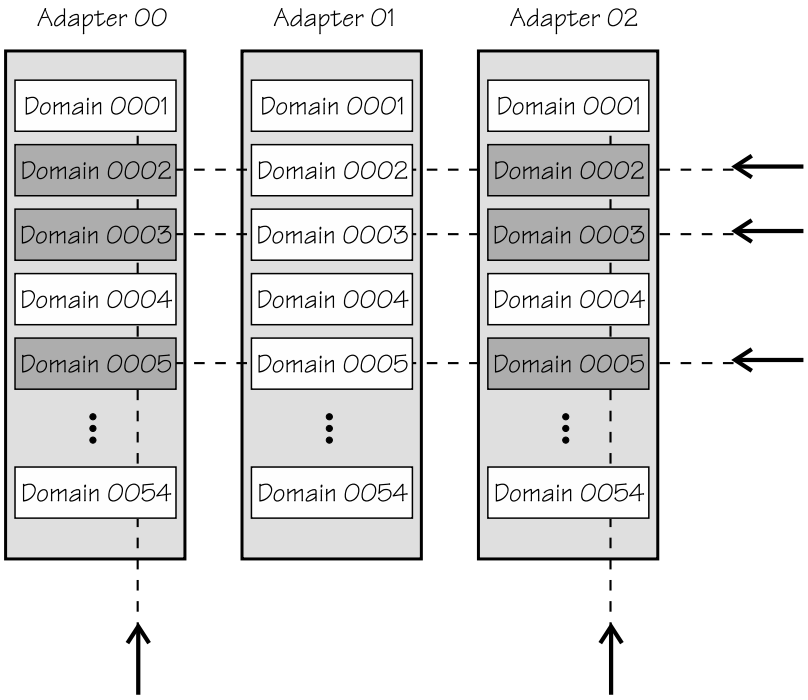
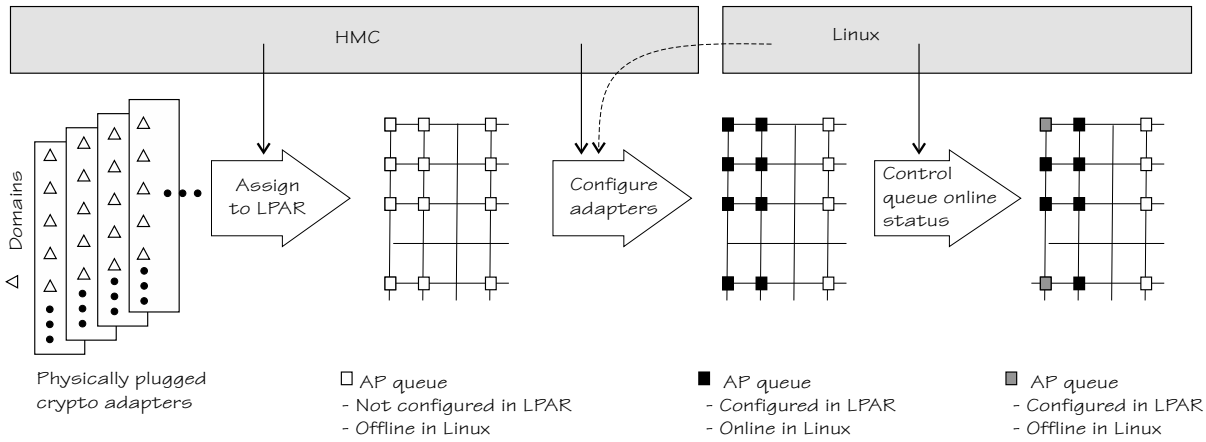
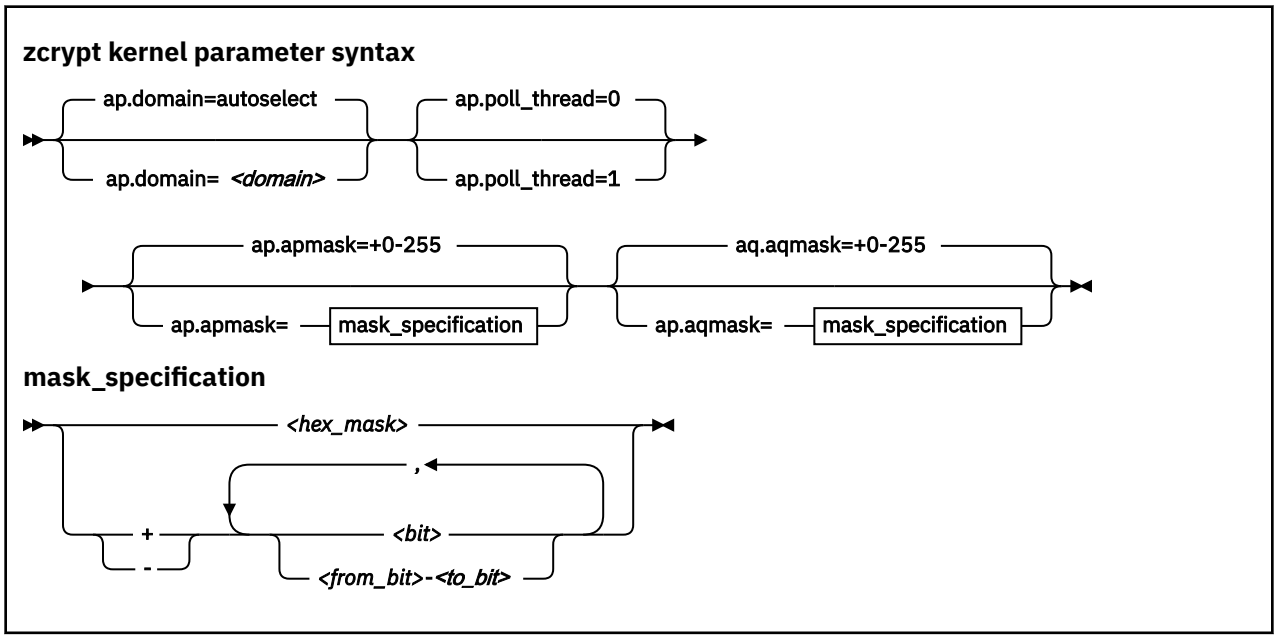
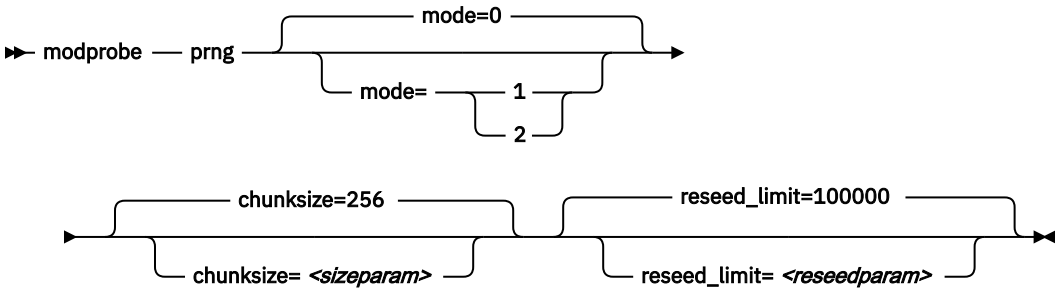
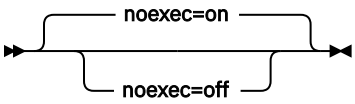
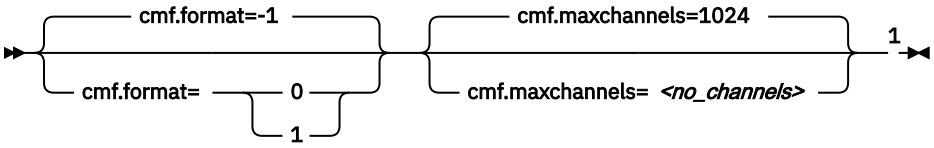
# ---------- Rules for CP messages go here --------

LABEL="smsgiucv_app_end"
```

The example first uses the **vmur** command to load the vmur kernel module. Then the z/VM virtual punch device, 000d, is activated.

```
# --- Initialization ---

# load vmur and set the virtual punch device online
SUBSYSTEM=="module", ACTION=="add", RUN+="/sbin/modprobe --quiet vmur"
SUBSYSTEM=="module", ACTION=="add", RUN+="/sbin/chccwdev -e 000d"
```

The following rule accepts messages from all senders. The message text must match the string UNAME. If it does, the output of the **uname** command (the node name and kernel version of the Linux instance) is sent back to the sender.

```
# --- Rules for all senders ----

# UNAME: tell the sender which kernel is running
ACTION=="change", ENV{SMSG_TEXT}=="UNAME", \
    PROGRAM=="/bin/uname -n -r", \
    RUN+="/sbin/vmcp msg $env{SMSG_SENDER} '$result'"
```

In the following example block rules are defined to accept messages from certain senders only. If no sender matches, processing ends. The message text must match the string DMESG. If it does, the environment variable PATH is set and the output of the **dmesg** command is sent into the z/VM reader of the sender. The name of the spool file is LINUX DMESG.

```
# --- Special rules available for particular z/VM user IDs ---

ENV{SMSG_SENDER}!="MAINT|OPERATOR|LNXADM", GOTO="smsgiucv_app_end"

# DMESG: punch dmesg output to sender
ACTION=="change", ENV{SMSG_TEXT}=="DMESG", \
    ENV{PATH}="/bin:/sbin:/usr/bin:/usr/sbin", \
    RUN+="/bin/sh -c 'dmesg |fold -s -w 74 |vmur punch -r -t -N LINUX.DMESG -u $env{SMSG_SENDER}'"
```

Chapter 43. Cooperative memory management

z/VM only: Cooperative memory management applies to Linux on z/VM only.

Cooperative memory management (CMM, or "cmm1") can reduce the memory that is available to an instance of Linux on z/VM.

CMM allocates pages to page pools that are not available to Linux. A diagnose code indicates to z/VM that the pages in the page pools are out of use. z/VM can then immediately reuse these pages for other z/VM guests.

To set up CMM, you must perform these tasks:

1. Load the cmm module.
2. Set up a resource management tool that controls the page pool. This tool can be the z/VM resource monitor (VMRM) or a third-party systems management tool.

This chapter describes how to set up CMM. For background information about CMM, see "Cooperative memory management background" on page 410.

You can also use the **cpuplugd** command to define rules for cmm behavior, see "Basic configuration file for memory control" on page 609.

For information about setting up the external resource manager, see the chapter on VMRM in *z/VM: Performance*, SC24-6301.

Setting up cooperative memory management

Set up Linux on z/VM to participate in the cooperative memory management by loading the cooperative memory management support module, cmm.

Use the **modprobe** command to load the module. See the **modprobe** man page for command details.

cooperative memory management module parameter syntax



where *<user_ID>* specifies the z/VM guest virtual machine that is permitted to send messages to the module through the special messages interface. The default z/VM user ID is VMRMSVM, which is the default for the VMRM service machine.

To load the cmm module automatically at boot time, see the section on persistent module loading in *Configuring basic system settings* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_basic_system_settings/).

Example

To load the cooperative memory management module and allow the z/VM guest virtual machine TESTID to send messages:

```
# modprobe cmm sender=TESTID
```

Working with cooperative memory management

After it has been set up, CMM works through the resource manager. No further actions are necessary. You might want to read the sizes of the page pools for diagnostic purposes.

To reduce the Linux memory size, CMM allocates pages to page pools that make the pages unusable to Linux. There are two such page pools, a static pool and a timed pool. You can use the procfs interface to read the sizes of the page pools.

Reading the size of the static page pool

You can read the current size of the static page pool from procfs.

Procedure

Issue this command:

```
# cat /proc/sys/vm/cmm_pages
```

Reading the size of the timed page pool

You can read the current size of the timed page pool from procfs.

Procedure

Issue this command:

```
# cat /proc/sys/vm/cmm_timed_pages
```

Part 7. KVM virtual server integration

KVM and LPAR: This part describes both Linux as a KVM guest and Linux as a KVM host.

These device drivers and features help you to effectively run and manage a KVM-based virtual Linux server farm.

Depending on your KVM host and on your virtual server configuration, a particular KVM guest might not provide all of the described features.

Newest version

You can find the newest version of this book at
ibm.com/docs/en/linux-on-systems?topic=linuxone-distributions

Restrictions

For prerequisites and restrictions see the IBM Z architecture specific information in the Red Hat Enterprise Linux 9.1 release notes at
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

Chapter 44. KVM virtualization on IBM Z

KVM only: This chapter applies to Linux on KVM only.

Red Hat Enterprise Linux 9.1 can run in the mainframe environment as virtualized by the KVM hypervisor.

virtio
provides paravirtualized devices, which hide the characteristics of the host devices and are similar across hardware platforms.

VFIO
provides pass-through devices, which preserve the characteristics of the host devices and include devices that are specific to IBM Z.

Omitting all technical detail and without claim to completeness, Figure 101 on page 457 shows an overview of how the KVM hypervisor virtualizes IBM Z resources for Linux on KVM.



Figure 101. KVM virtualization on IBM Z

The KVM hypervisor defines the CPUs, memory, and virtual devices that are available to an instance of Linux on KVM when it is booted. It also specifies the host resources that back these guest resources.

As indicated in Figure 101 on page 457, virtualization options depend on the host device. OSA devices are always virtualized as virtio Ethernet devices. A DASD can be virtualized as a virtio block device or as a VFIO pass-through DASD. Depending on the device type, PCIe devices can be virtualized as virtio block devices, virtio Ethernet devices, or VFIO pass-through PCIe devices.

Pass-through devices block live guest migration. If applicable, dynamically remove all pass-through devices from a virtual server before a live guest migration and dynamically add them after the migration. Dynamically removing or adding devices through the hypervisor results in hotplug events on the guest.

Virtio devices

Device paravirtualization with virtio hides most of the physical device aspects from the guest. A virtio-net network device might be backed on the host, for example, by a physical OSA device, a HiperSockets device, a PCIe-attached Mellanox adapter, or an Open vSwitch configuration. A virtio-blk device might be backed, for example, by a DASD, a SCSI LUN, an NVMe device, or an image file in the host file system.

Both virtio-blk and virtio-net devices use the virtio framework. The virtio CCW transport device driver provides the interface to this framework and uses channel command words (CCW) and a virtual channel subsystem to realize the virtio infrastructure.

Figure 102 on page 458 illustrates the virtio stack for Linux as a KVM guest on IBM Z.



Figure 102. virtio stack

Not all virtio devices are based on host devices. For example, virtual GPUs, virtual keyboard and mouse devices, and virtual SCSI Host Bus Adapters (HBAs) are all provided by the hypervisor.

Virtual SCSI HBAs enable a guest to work with virtual SCSI LUNs within a virtual server. These virtual SCSI LUNs can map to SCSI LUNs or other resources on the host, or they can be provided by the hypervisor. The following examples show common mappings for a virtual SCSI LUN:

- A LUN for a SCSI-attached tape drive on the host.
- A DVD ISO file on the host file system that is mounted on a virtual DVD drive.
- An image file in the host file system.

Another type of virtio device, virtio-fs, can give a KVM guest access to a branch of the host file system.

A Linux instance that is to run as a guest of KVM on IBM Z must support virtio virtualization of the IBM Z environment. In particular, the device drivers for the devices in the virtual channel subsystem must be in place (see Chapter 46, “The virtio CCW transport device driver,” on page 467).

For more information about the virtio framework, see developer.ibm.com/articles/l-virtio.

VFIO pass-through devices

VFIO virtualization is designed to pass devices with their physical attributes through to KVM guests. On the guest, these pass-through devices then can be handled by the same device drivers that would also handle them on the host.

Devices require a special setup on the host to be eligible for VFIO virtualization. In particular, devices must be freed from control of their default device drivers and assigned to an applicable VFIO device driver. For details see, Chapter 48, “Setting up a KVM host for VFIO pass-through,” on page 477.

Linux on KVM versus Linux on z/VM or Linux in LPAR mode

If you are familiar with Linux on z/VM or with Linux in LPAR mode, you will observe some differences when working with Linux on IBM Z as a KVM guest.

Starting and stopping Linux

The KVM hypervisor is the control point for IPL of Linux on KVM. You can initiate a reIPL from a running instance of Linux on KVM.

System dump

As for Linux in LPAR mode and for Linux on z/VM, you can use kdump as a dump tool.

Alternatively, you can initiate a dump on the host. These hypervisor-driven dumps are analogous to using VMDUMP for Linux on z/VM.

You cannot use the stand-alone dump tools to create a dump for Linux on KVM.

For more details, see “Creating a kernel dump of a KVM guest” on page 569.

Responsibilities

Some of the administrative powers and responsibilities for the hardware that backs devices or provides access to devices are offloaded from the guest to the host.

Virtual channel subsystem

The KVM hypervisor provides a virtualized channel subsystem with virtual channel paths to its guests. CHPID `00` on this virtual channel subsystem is shared by all virtio-ccw devices, including virtio-net and virtio-blk devices. See Chapter 45, “The virtual channel subsystem,” on page 463.

Storage devices

Expect to find generic block devices, which can be backed on the KVM host by SCSI disks, DASDs, PCIe-attached NVMe devices, or even files in the host file system.

For these generic block devices, you cannot and need not configure any adapter hardware or physical disk devices. This preparation is done for you by the host.

There are no storage-class memory increments.

Network devices

Expect to find generic virtio network devices, which can be backed on the KVM host by a physical OSA device or a HiperSockets device.

For these generic network devices, you cannot and need not group subchannels into CCW group devices, configure any adapter hardware, or configure the device itself. This setup is done for you by the host.

Linux as a KVM guest on IBM Z versus distributed systems

If you are familiar with Linux as KVM guests on workstations, you will observe some differences when working with Linux as a KVM guest on IBM Z.

Virtual channel subsystem

Linux as a KVM guest on IBM Z uses a virtual IBM Z channel subsystem to access CCW devices.

The virtio CCW transport device driver handles all I/O to virtio storage and network devices through the same virtual channel path, CHPID `00`, on this channel subsystem. Regular Linux on IBM Z device drivers use different channel paths to access VFIO CCW devices.

Cryptographic support

Linux as a KVM guest on IBM Z can use the IBM Z CP Assist for Cryptographic Function (CPACF). If configured for your KVM virtual server, you can also use IBM Z cryptographic adapters (see Part 8, "Security," on page 487).

Absence of common workstation devices

Do not expect to find all device types that are common on workstations. For example, you will not find USB devices.

Live guest migration

In a live guest migration, the system programmer relocates a KVM virtual server with a running Linux instance from one KVM host to another without significantly disrupting operations.

Live guest migrations can help, for example, to avoid downtime during maintenance activities. A live guest migration can succeed only if both KVM hosts have access to equivalent resources. The hosts can but need not run on the same IBM Z or LinuxONE hardware. The system programmer, who also initiates the migration, ensures that all preconditions are met.

If live migration is used at your installation, be sure not to block the migration. In particular:

- The virtual server configuration must not include any vfio_ap devices, which provide access to AP queues on cryptographic adapters.
- PCI passthrough devices must be detached before live migration.
- All tape device nodes must be closed and online tape drives must be unloaded.
- Any virtio-fs file systems, which are shared with the KVM host, must be unmounted.
- No program must be in a prolonged uninterruptible sleep state. Programs can assume this state while waiting for an outstanding I/O request to complete. Most I/O requests complete fast and do not compromise live guest migration. An example of an I/O request that can take too long to complete is rewinding a tape.

Linux as an IBM Secure Execution host or guest

With IBM Secure Execution for Linux, you can run encrypted Linux images on a public, private, or hybrid cloud with their in-use memory protected.

IBM Secure Execution for Linux was introduced with IBM z15 and LinuxONE III.

Both KVM hosts and KVM guests must be set up to support IBM Secure Execution mode. This setup includes two kernel parameters, one for hosts and one for guests.

prot_virt=
: By default, KVM hosts do not support guests in IBM Secure Execution mode. To support such guests, KVM hosts must boot in LPAR mode with the kernel parameter specification `prot_virt=1`.

  KVM hosts that successfully start with support for IBM Secure Execution for Linux issue a kernel message like this: `prot_virt: Reserving <amount>MB as ultravisor base storage`.

swiotlb=
: KVM guests in IBM Secure Execution mode require *bounce buffers* for their virtio devices. Use the `swiotlb=` kernel parameter to assign 2 KB memory blocks for these bounce buffers. A suitable setting for most cases is `swiotlb=262144`, which corresponds to 512 MB.

For details about setting up KVM hosts and guest, see *Introducing IBM Secure Execution for Linux*, SC34-7721.

Indicators for IBM Secure Execution mode

Two read-only sysfs attributes indicate whether a running Linux instance detects an environment of a KVM guest in IBM Secure Execution mode or of a KVM host that can run such guests.

/sys/firmware/uv/prot_virt_guest
: The value of this attribute is 1 for Linux instances that detect their environment as consistent with that of a secure guest. For other instances, the value is 0 or the attribute does not exist.

/sys/firmware/uv/prot_virt_host
: The value of this attribute is 1 for Linux instances that detect their environment as consistent with that of a secure host. For other instances, the value is 0. If the attribute does not exist, the Linux instance is not a KVM host in an environment that supports IBM Secure Execution for Linux.

Note: These values are indications, but do not prove that the Linux instance is a secure guest or host in the context of IBM Secure Execution for Linux. Use these indications for technical evaluations in trusted environments, but do not base security-related decisions on them.

The following example shows a Linux instance that runs as a KVM guest in IBM Secure Execution mode, but is not a KVM host that can run such guests.

```
# cat /sys/firmware/uv/prot_virt_guest
1
# cat /sys/firmware/uv/prot_virt_host
0
```

Chapter 45. The virtual channel subsystem

KVM only: The virtual channel subsystem is specific to Linux on KVM.

The KVM hypervisor provides a virtual channel subsystem to its guests.

This virtual channel subsystem connects paravirtualized CCW devices to the virtual server. In the virtual channel subsystem:

- All paravirtualized CCW devices have control unit type 3832/<*nn*>, where <*nn*> is a two-digit hexadecimal number that indicates the device type.
- All paravirtualized CCW devices use the same virtual channel path with CHPID `00`. The availability of all paravirtualized CCW devices depends on this channel path being operational.

For general information about the channel subsystem, see *z/Architecture Principles of Operation*, SA22-7832.

Listing devices with lscss

The particulars of the channel subsystem view of a guest become visible when you list devices with **`lscss`**.

Example

```
# lscss
Device   Subchan. DevType CU Type Use PIM PAM POM CHPIDs
------------------------------------------------------------------------
0.0.0042 0.0.0000 0000/00 3832/01 yes 80  80  ff  00000000 00000000
0.0.0815 0.0.0001 0000/00 3832/02 yes 80  80  ff  00000000 00000000
0.0.9999 0.0.0002 0000/00 3832/03 yes 80  80  ff  00000000 00000000
0.1.abcd 0.1.0000 0000/00 3832/05 yes 80  80  ff  00000000 00000000
...
0.1.6196 0.1.36e2 3390/0c 3990/e9 yes fc  fc  ff  32333435 40410000
...
```

As illustrated in the example, the output, typically, shows numerous paravirtualized CCW devices. The output columns `DevType`, `PIM`, `PAM`, `POM`, and `CHPIDs` show identical values for these devices. These values result from the virtualization and carry no information that is characteristic for a particular device.

The output of the example also includes a passthrough DASD, with device type 3390/0c, for which all fields provide device-characteristic information. This device is accessed through channel paths other than the one dedicated to virtio devices, and all fields provide device-characteristic information.

The following columns contain meaningful device information for paravirtualized CCW devices:

`Device`
: is the device bus-ID that uniquely identifies a device to the guest and to the KVM hypervisor.

  Use device bus-IDs to identify devices to the KVM hypervisor administrator. The KVM hypervisor defines these bus-IDs with prefix `fe` instead of `0`. For example, `0.0.0042` on the guest is specified as `fe.0.0042` in the virtual server configuration on the KVM hypervisor.

  Device bus-IDs are persistent across reboots and change only if the device definitions are changed in the KVM hypervisor.

`Subchan.`
: shows the current assignment of a subchannel to the device.

  In contrast to the persistent device bus-IDs, subchannel assignments to devices might change across reboots or as a result of hotplug events.

`CU Type`
: has a two-digit suffix that identifies the device type.

For example, `01` in `3832/01` identifies a network device and `02` in `3832/02` identifies a block device. For more information, see "Types of paravirtualized CCW devices" on page 464.

Use
indicates whether the device is online.

Types of paravirtualized CCW devices

For Linux as a KVM guest on IBM Z, paravirtualized CCW devices can represent various real and virtual devices, including block devices, network devices, and devices that are attached through a virtual SCSI HBA.

Table 58 on page 464 explains the values that are shown in the `CU Type` column of the **`lscss`** command. Which of these devices are present on a particular KVM guest depends on the virtual server configuration on the KVM hypervisor.

Table 58. Types of paravirtualized CCW devices

CU Type/Model	Explanation
`3832/01`	Network device The corresponding device bus-ID represents a network interface on the guest. The details of the interface are hidden by the KVM hypervisor. On the KVM hypervisor, this interface might be based on a MacVTap interface or a virtual switch. Network devices are handled by the `virtio_net` device driver module. See "Virtual network devices" on page 470 for details.
`3832/02`	Block device The corresponding device bus-ID represents a persistent storage space to the guest. The details of the block device are hidden by the KVM hypervisor. To the KVM hypervisor, this storage space might be a SCSI LUN or a DASD, but it might also be a file in the file system of the host or any other block device. Block devices are handled by the `virtio_blk` device driver module. See "Virtual block devices" on page 468 for details.
`3832/03`	Character device for console output. See "Using virsh on a KVM host" on page 40.
`3832/04`	Random number generator device Depending on the configuration of your virtual server by the KVM hypervisor, this device might be backed by IBM Z cryptographic hardware. This device provides sufficient random numbers of good quality only if the random device of KVM host does so. In particular, this devices provides true random numbers only if it is backed by a true random number generator on the KVM host. This device is provided by the `virtio_rng` device driver module.
`3832/05`	Balloon device for memory management. The preferred memory management technology is Collaborative Memory Management Assist (CMMA). See "cmma - Reduce hypervisor paging I/O overhead" on page 784.

Table 58. Types of paravirtualized CCW devices (continued)	
CU Type/Model	**Explanation**
3832/08	Virtual SCSI HBA SCSI devices can be attached through a virtual SCSI host bus adapter (HBA) and are then handled by the `virtio_scsi` device driver module. For example, the following devices are attached through a virtual SCSI HBA: • SCSI tapes (see "Virtual SCSI-attached tape devices" on page 470) • Virtual CD/DVD drives (see "Virtual SCSI-attached CD/DVD drives" on page 472) Host devices need not necessarily be attached through a virtual SCSI HBA. For example, SCSI-attached disks are usually virtualized as block devices and handled by the `virtio_blk` device driver module.
3832/10	Virtual graphics processing unit (GPU). Supports remote access to graphical user interfaces. GPU devices are handled by the `virtio_gpu` device driver module.
3832/12	Virtual human interface devices, like a virtual keyboard and a virtual mouse, as remote input devices for graphical user interfaces. These devices are handled by the `virtio_input` device driver module.
3832/13	Devices for communications between applications on the guest and on the host. For example, hypervisor services can use these devices to communicate with guest agents. These devices are handled by the `virtio_vsock` device driver module.
3832/1a	Device that provides access to a branch of the host file system.

Listing channel paths with lschp

Linux as a KVM guest on IBM Z includes a channel path, with CHPID `00`, for paravirtualized CCW devices.

The virtual channel subsystem always provides the same single channel path for all paravirtualized CCW devices to the guest. The following sample output for **`lschp`** shows this channel path:

```
# lschp
CHPID Vary Cfg. Type Cmg Shared PCHID
=========================================
0.00  1    -    32   -   0      -
```

Attention: Setting this channel path logically offline would make all paravirtualized CCW devices inaccessible to the guest. As a consequence, the system is likely to crash.

Depending on the presence of passthrough CCW devices, there can also be output lines for other devices.

Chapter 46. The virtio CCW transport device driver

KVM only: The virtio CCW transport device driver applies to Linux on KVM only.

The virtio CCW transport device driver handles the virtual channel command word (CCW) devices that are provided by the KVM hypervisor.

Virtio CCW devices are accessed through a virtual channel subsystem, see Chapter 45, "The virtual channel subsystem," on page 463.

The virtio CCW transport device driver is part of a module stack that also includes device drivers for handling particular device types. Distributions can handle these modules differently:

- The modules might be compiled into the kernel image of the distribution.
- The distribution might include separately compiled modules that are usually loaded automatically as they are required.

If the distribution includes a separate module that is not loaded automatically, you must load it before you can work with the corresponding devices. Loading a supporting module with the **modprobe** command automatically loads the base module if needed.

Virtio devices

The KVM hypervisor hides some of the specifics of the devices it virtualizes. For example, the hypervisor can virtualize both disk devices and plain files in the host file system as block devices. The KVM guest cannot differentiate block devices according to their nature on the host.

As a user of Linux on KVM, you must work with the virtual devices at the abstraction level with which they are presented. You cannot perform all actions against virtual devices that you can perform against real devices.

Setting CCW devices offline or online

By default, all virtio CCW devices are online after an instance of Linux as a KVM guest on IBM Z is booted.

About this task

If the KVM hypervisor defines unnecessary devices to your Linux instance, you can set them offline.

Tip: You can also use the `cio_ignore=` kernel parameter to prevent unnecessary devices from being sensed in the first place (see "cio_ignore - List devices to be ignored" on page 780).

Procedure

Use the **chzdev** or **chccwdev** command to set CCW devices offline or online.

Alternatively, write `0` to the device's sysfs online attribute to set it offline or `1` to set it online. In contrast to the commands, writing to the sysfs attribute does not trigger a cio_settle for you.

Use the `cio_ignore=` kernel parameter to persistently set a CCW device offline. Setting a device offline with **chzdev** and the `--persistent` option does not prevent the device from being set online with the next start of the virtual server.

For example, to set a device with bus ID `0.0.0815` offline, issue:

```
# chzdev -d 0.0.0815
```

To set this device back online, issue:

```
# chzdev -e 0.0.0815
```

To set the device offline by writing to sysfs, issue:

```
# echo 0 > /sys/bus/ccw/drivers/virtio_ccw/0.0.0815/online
```

Virtual block devices

On Linux as a KVM guest on IBM Z, you can use generic virtual block devices, for example, paravirtualized DASDs or SCSI LUNs.

These virtual block devices are handled by the `virtio_blk` device driver module. This module is loaded automatically during the boot process.

A virtual block device might be backed by a disk device, but it might also be backed by a file on the hypervisor. Do not perform operations that require knowledge of the specific hardware that backs a virtual block device. For example, do not attempt to run a low-level formatting operation on a virtual block device.

Block device naming-scheme

Applications access block devices through device nodes. The virtio-blk device driver uses 16 device nodes for each block device: one for the block device itself and 15 for partitions.

The standard device nodes are of the form:

- /dev/vd<*x*> for the block device
- /dev/vd<*x*><*n*> for partitions

where

<x>
represents one or more alphabetic characters; vd<*x*> matches the device name that is used by the virtio-blk device driver.

<n>
is an integer in the range 1-15.

All of these nodes use the same major number. You can find the major number by issuing the following command:

```
# cat /proc/devices | grep virtblk
```

Table 59. Naming scheme for virtio block devices

Name that is used by the device driver	Standard device node	Minor number	Description
vda vda1 vda2 ... vda15	/dev/vda /dev/vda1 /dev/vda2 ... /dev/vda15	0 1 2 ... 15	First block device and up to 15 partitions
vdb vdb1 vdb2 ... vdb15	/dev/vdb /dev/vdb1 /dev/vdb2 ... /dev/vdb15	16 17 18 ... 31	Second block device and up to 15 partitions

Table 59. Naming scheme for virtio block devices (continued)

Name that is used by the device driver	Standard device node	Minor number	Description
vd<*x*> vd<*x*>1 vd<*x*>2 ... vd<*x*>15	/dev/vd<*x*> /dev/vd<*x*>1 /dev/vd<*x*>2 ... /dev/vd<*x*>15	(<m>-1)×16 (<m>-1)×16+1 (<m>-1)×16+2 ... (<m>-1)×16+15	<*m*>-th block device with up to 15 partitions

With 1,048,576 (20-bit) available minor numbers, the virtio-blk device driver can address 65,536 block devices and their partitions. For the first 26 devices, <*x*> is one alphabetic character (vda-vdz). The next devices use first two (vdaa-vdzz) and then more alphabetic characters.

The mapping of standard device nodes to bus-IDs can change when Linux is rebooted or when hotplug events occur. Your distribution might provide udev rules that create other nodes to attain a persistent mapping between device nodes and bus-IDs.

Mapping block devices to CCW devices

Each virtual block device corresponds to an online CCW device.

Other than the standard device nodes, udev-created device nodes can be based on the bus ID of the CCW device and so provide a persistent mapping of node to CCW device. Preferably, use such persistent device nodes when working with virtual block devices.

Use the information that follows if you need to find out the current mapping between standard device nodes and CCW devices. For example, you might need this mapping for diagnostic explorations.

To list the device nodes for your block devices, issue:

```
# ls /sys/block
```

The command output is a list of symbolic links that match the device names of the block devices.

Example:

```
# ls /sys/block
vda     vdb     vdc
```

These links contain several attributes, including another symbolic link, `device`. To find the bus ID for a particular block device, issue a command according to the following example:

Example:

```
# ls -1 /sys/block/vdb/device/../.. | head -1
0.0.1111
```

Tip: For an overview of the mapping, issue this command:

```
# ls -d /sys/devices/css0/*/*/virtio*/block/*
```

Example:

```
# ls -d /sys/devices/css0/*/*/virtio*/block/*
/sys/devices/css0/0.0.0000/0.0.10b1/virtio3/block/vda
/sys/devices/css0/0.0.0001/0.0.1111/virtio4/block/vdb
/sys/devices/css0/0.0.0002/0.0.11ab/virtio5/block/vdc
```

You can pipe the output to **awk** to obtain a more compact view:

```
# ls -d /sys/devices/css0/*/*/virtio*/block/* | awk -F "/" '{print $9 "\t" $6}'
vda     0.0.10b1
vdb     0.0.1111
vdc     0.0.11ab
```

Partitioning virtual block devices

How to partition a block device depends on how the device is backed on the host, DASD or other.

Before you begin: Block devices that are backed by a DASD must first be formatted with **dasdfmt** on the host. Use the **fdasd -i** or **parted print** command to find out if your block device is backed by a DASD.

DASD backed block devices
Use the **fdasd** command to create up to 3 partitions. For details, see "fdasd – Partition a DASD" on page 632 or the **fdasd** man page

All other block devices
Use the common code **fdisk** command to create up to 15 partitions. For details, see the **fdisk** man page.

Alternatively, you can use the **parted** command to create partitions. The **parted** command can handle both DASD-backed and other block devices. For details, see the **parted** man page.

The partitions of a block device are represented as subdirectories of the device representation in `/sys/block`. For example, you can list the existing partitions of a block device `/sys/block/vda` by issuing:

```
# ls /sys/block/vda
```

Virtual network devices

On Linux as a KVM guest on IBM Z, you use generic network devices and predictable interface names for Ethernet interfaces.

Read `/sys/class/net` to obtain a list of interfaces. Each list item corresponds to an online CCW network device.

Example:

```
# ls /sys/class/net
encf500     enced0
```

According to convention, the device bus IDs of the associated CCW devices are 0.0.f500 and 0.0.0ed0 (see "Predictable network interface names" on page 4).

Use **ip** or an equivalent command to activate an interface.

Example:

```
# ip addr 192.0.2.5 dev encf500 peer 192.0.2.6
```

Virtual SCSI-attached tape devices

The representation of virtual SCSI-attached tape and medium changer devices on Linux as a KVM guest on IBM Z depends on your device driver.

st

The `st` device driver for SCSI tape drives is included in the Linux kernel source from kernel.org. Red Hat Enterprise Linux supplies it as a separate module.

For each device, `st` provides device nodes of the form `/dev/st<i><x>` and `/dev/nst<i><x>` where the latter is for non-rewinding devices, where

<x>
is an alphabetic character that specifies a tape property, for example, compression or encryption.

<i>
identifies an individual device.

The identifier, *<i>*, is assigned when Linux is booted or when a device is set online. As a result, there is no fixed mapping between a physical tape device and the tape device nodes. For details, see the `st` man page.

ch

The `ch` device driver for SCSI medium changers is included in the Linux kernel source from kernel.org. Red Hat Enterprise Linux supplies it as a separate module.

For each device, `ch` provides device nodes of the form `/dev/sch`*<i>* where *<i>* identifies an individual device.

The identifier, *<i>*, is assigned when Linux is booted or when a device is set online. As a result, there is no fixed mapping between physical media changer devices and the media changer device nodes. For details, see `Documentation/scsi/scsi-changer.txt` in the Linux source code.

lin_tape

The `lin_tape` device driver is available from the IBM Fix Central site at www.ibm.com/support/fixcentral. For details about downloading the device driver, see Technote 1428656.

The device nodes that it provides include characteristics of the physical tape drive or medium changer and are persistent across reboots and after setting a tape device offline and back online. For details, see *IBM Tape Device Drivers Installation and User's Guide*, GC27-2130.

Listing your tape devices

Use the **`lsscsi`** command with the `-v` option to list all your SCSI-attached devices, including SCSI-attached tape and medium changer devices. You can also use the **`lstape`** command to list tape and medium changer devices.

Example:

```
# lsscsi -v
[0:0:0:0]   tape       IBM      ULT3580-TD6     H990  /dev/st0
  dir: /sys/bus/scsi/devices/0:0:0:0 \
 [/sys/devices/css0/0.0.0000/0.0.0003/virtio2/host0/target0:0:0/0:0:0:0]
[0:0:0:1]   mediumx    IBM      3573-TL         E.80  /dev/sch0
  dir: /sys/bus/scsi/devices/0:0:0:1 \
 [/sys/devices/css0/0.0.0000/0.0.0003/virtio2/host0/target0:0:0/0:0:0:1]
[0:0:1:0]   tape       IBM      ULT3580-TD6     H990  /dev/st1
  dir: /sys/bus/scsi/devices/0:0:1:0 \
 [/sys/devices/css0/0.0.0000/0.0.0003/virtio2/host0/target0:0:1/0:0:1:0]
[0:0:1:1]   mediumx    IBM      3573-TL         E.80  /dev/sch1
  dir: /sys/bus/scsi/devices/0:0:1:1 \
 [/sys/devices/css0/0.0.0000/0.0.0003/virtio2/host0/target0:0:1/0:0:1:1]
[1:0:0:0]   tape       IBM      ULT3580-TD6     H990  /dev/st3
  dir: /sys/bus/scsi/devices/1:0:0:0 \
 [/sys/devices/css0/0.0.0001/0.0.0004/virtio5/host1/target1:0:0/1:0:0:0]
[1:0:0:1]   mediumx    IBM      3573-TL         E.80  /dev/sch3
  dir: /sys/bus/scsi/devices/1:0:0:1 \
 [/sys/devices/css0/0.0.0001/0.0.0004/virtio5/host1/target1:0:0/1:0:0:1]
[1:0:1:0]   tape       IBM      ULT3580-TD6     H990  /dev/st2
  dir: /sys/bus/scsi/devices/1:0:1:0 \
 [/sys/devices/css0/0.0.0001/0.0.0004/virtio5/host1/target1:0:1/1:0:1:0]
[1:0:1:1]   mediumx    IBM      3573-TL         E.80  /dev/sch2
  dir: /sys/bus/scsi/devices/1:0:1:1 \
 [/sys/devices/css0/0.0.0001/0.0.0004/virtio5/host1/target1:0:1/1:0:1:1]
```

The output includes the device node as used by the `st` or `ch` device driver and the SCSI device name of the form *<scsi_host_no>*:0:*<scsi_id>*:*<scsi_lun>*, `0:0:0:0` for `/dev/st0` in the example.

If the devices are handled by `lin_tape` instead of `st` or `ch`, **`lsscsi`** cannot determine the device node name and displays "-" instead.

The sysfs path in the output includes two bus IDs:

- The first bus ID, from left to right, applies to the subchannel
- The second bus ID applies to the virtual SCSI host bus adapter (HBA)

The two bus IDs can but do not need to be the same. In the example, the HBA device bus-ID for `/dev/st0` is `0.0.0003`.

The following **`lstape`** output for the same device setup assumes that the devices are managed by the `lin_tape` device driver instead of `st` and `ch`. The **`lstape`** output also shows the generic device name, `sg<x>`, that is assigned by the SCSI generic device driver, `sg`.

```
# lstape --verbose
FICON/ESCON tapes  (found 0):
TapeNo  BusID       CuType/Model  DevType/Model    BlkSize State    Op      MedState
SCSI tape devices (found 8):
Generic Device         Target         Vendor   Model            Type       State
        HBA            WWPN                    Serial
sg0     IBMtape0       0:0:0:0        IBM      ULT3580-TD6      tapedrv    running
        0.0.0003       N/A                     10WT037733
sg1     IBMchanger0    0:0:0:1        IBM      3573-TL          changer    running
        0.0.0003       N/A                     00L4U78W6497_LL0
sg3     IBMtape1       0:0:1:0        IBM      ULT3580-TD6      tapedrv    running
        0.0.0003       N/A                     10WT037701
sg2     IBMchanger1    0:0:1:1        IBM      3573-TL          changer    running
        0.0.0003       N/A                     00L4U78W6497_LL0
sg6     IBMtape3       1:0:0:0        IBM      ULT3580-TD6      tapedrv    running
        0.0.0004       N/A                     10WT037733
sg7     IBMchanger3    1:0:0:1        IBM      3573-TL          changer    running
        0.0.0004       N/A                     00L4U78W6497_LL0
sg4     IBMtape2       1:0:1:0        IBM      ULT3580-TD6      tapedrv    running
        0.0.0004       N/A                     10WT037701
sg5     IBMchanger2    1:0:1:1        IBM      3573-TL          changer    running
        0.0.0004       N/A                     00L4U78W6497_LL0
```

Use the SCSI device name and the device bus-ID to communicate about the devices with the hypervisor administrator.

Virtual SCSI-attached CD/DVD drives

The KVM hypervisor might provide virtual SCSI-attached CD/DVD drives to your KVM guest.

Virtual SCSI-attached CD/DVD drives have device nodes of the form `/dev/sr<n>`, where *<n>* is an integer that identifies an individual device. The node for the first drive is `/dev/sr0`.

Issue the following command to list all device nodes for CD/DVD drives:

```
# ls /dev/sr*
```

You can also use the **`lsscsi`** command to list all your SCSI-attached devices, including SCSI-attached CD/DVD drives.

```
# lsscsi
[0:0:0:0]    cd/dvd  QEMU     QEMU CD-ROM      2.3.  /dev/sr0
```

You can use the **`isoinfo`** command with the -i option to find out if a drive contains media.

Example:

```
# isoinfo -i /dev/sr0
```

This command returns an error if no media is present.

You can use the **`mount`** command to mount the content of media in the drive on the file system.

Example:

```
# mount /dev/sr0 /mnt/media
```

Your distribution might provide a udev rule to mount the media content for you.

Unmount the content of the media to release it.

Example:

```
# unmount /dev/sr0
```

You depend on the KVM hypervisor to eject and insert media.

Host file-system branches as virtual file systems

A special type of virtio device can give a KVM guest access to a branch of the host file system.

The virtio device specifications in the virtual server configuration include the path to the host branch and a mount tag that identifies the file system. You use this tag to mount the branch on your KVM guest with a command of this form:

```
# mount --types virtiofs <mount_tag> <mount point>
```

Example: The following command mounts the branch of the file system that is configured with a tag my_shared_fs at `/mnt/shared/` on the guest.

```
# mount --types virtiofs my_shared_fs /mnt/shared/
```

You can persistently mount the file system branch through the following entry in `/etc/fstab`.

```
my_shared_fs /mnt/shared defaults 0 0
```

Chapter 47. Setting up Red Hat Enterprise Linux 9.1 as a KVM host

LPAR or KVM: KVM hosts can run in LPAR mode or they can be nested hosts that run as KVM guests.

A KVM host has hardware and user space requirements.

Hardware

The real or virtual hardware must provide the Start Interpretive Execution (SIE) feature (see “Check whether the Linux instance can be a hypervisor” on page 567).

User space

You must install the packages for QEMU and libvirt.

Module parameters

Configure the KVM host support with module parameters.

kvm module parameter syntax



nested=
If set to "1", passes the SIE capability on to its guests. Thus, the guests can be hosts for higher-level guests.

Nested KVM hosts are intended for test environments and not for production systems.

hpage=
If set to "1", supports guest-configurations with huge-page memory backing.

Enough 1 MB huge pages must be set up to satisfy the needs of the guests. For details about setting up huge pages, see Chapter 26, “Huge-page support,” on page 369.

Consider making this setting persistent, for example, through a modprobe configuration file or by specifying `kvm.hpage=1` on the kernel parameter line.

Nested hosts cannot back their guests with huge pages. You cannot set both `nested=` and `hpage=` to `1`.

Making parameter settings persistent across re-boots

To make a module parameter persistent, create a file called `/etc/modprobe.d/kvm.conf` with, for example, the following content:

```
options kvm hpage=1
```

Unload and reload the kvm module to make the changes take effect.

Chapter 48. Setting up a KVM host for VFIO pass-through

LPAR or KVM: KVM hosts can run in LPAR mode or they can be nested hosts that run as KVM guests.

KVM hosts can use the Virtual Function I/O (VFIO) framework and the VFIO mediated device framework to pass host devices with their attributes through to their KVM guests.

For general information about VFIO and VFIO mediated devices, see `Documentation/vfio.txt` and `Documentation/vfio-mediated-device.txt` in the Linux kernel source. You can also find this information by searching for "vfio" at www.kernel.org/doc/html/latest/search.html.

What you should know about VFIO

Depending on the device type, Linux handles devices with specific device drivers.



Figure 103. Device drivers on Linux

For Linux on KVM, QEMU provides virtual VFIO pass-through devices that preserve the host-device attributes. Therefore, Linux on KVM accesses a pass-through device with the same device driver that the host would use to access the corresponding host resource. For example, Linux in LPAR mode uses the DASD device driver to access DASDs. Correspondingly, Linux on KVM uses the DASD device driver to access VFIO pass-through DASDs.



Figure 104. Device drivers for VFIO pass-through devices

To avoid contention, a KVM host must relinquish direct control of the host resource that backs a VFIO pass-through device. For these host resources, the VFIO framework substitutes the default device drivers on the KVM host with device-specific VFIO device drivers. These substitute device drivers reserve host resources for guest use and provide access to these resources on behalf of the guest.

Red Hat Enterprise Linux 9.1 as a KVM host on IBM Z supports the following types of pass-through devices:

- PCIe
- CCW (DASD)
- Cryptographic adapter resources (AP queues)



Figure 105. VFIO virtualization

On the host, you must define the resources that back a VFIO pass-through device and associate these resources with the applicable VFIO device driver. The necessary configuration steps depend on the device type. For pass-through DASD and pass-through cryptographic adapter resources, you must create specific VFIO mediated devices. The KVM hypervisor then uses VFIO mediated devices as a source for pass-through devices.

For configuration details, see “Host setup for VFIO pass-through devices” on page 478.

Host setup for VFIO pass-through devices

You must ensure that a VFIO device driver controls the resource on the host.

About this task

The VFIO device driver reserves the pass-through device for KVM guests and accesses the corresponding host resource on behalf of the guest.

Procedure

Although the details differ considerably by device type, the following main steps apply to all VFIO pass-through devices:

1. Free the resource from control of the default device driver.

2. Associate the resource with the corresponding VFIO device driver.

What to do next

After completing the host setup for a VFIO pass-through device, you can configure the device for a KVM guest (see *KVM Virtual Server Management*, SC34-2752).

Setting up PCI devices for VFIO pass-through

To set up a PCI device as a VFIO pass-through device you must enable the vfio_pci device driver to handle the PCI device type, and you must assign the specific device to vfio_pci.

Before you begin: The preferred method for setting up PCI devices is to configure them for automatic management with libvirt, see the information about configuring VFIO pass-through devices in *KVM Virtual Server Management*, SC34-2752. This management includes a dynamic host preparation. The tasks that follow describe a fallback method that applies only to PCI devices that are not managed by libvirt.

Enabling the vfio_pci device driver

For devices that are not managed by libvirt, enable the vfio_pci device driver for specific PCI card types. Proceed according to how your vfio_pci device driver is compiled: as part of the kernel image or as a separate module.

Tip: You must know the device's vendor code and device code. Issue **`lspci -n`** to display this information for your PCIe devices in the format *<vendor_code>*:*<device_code>*.

Module parameter

If the vfio_pci device driver is compiled as a separate module, you can use the `ids=` module parameter to specify the PCIe card types.

vfio_pci module parameter syntax



Example: In this example, a PCIe device with function address `0001:000:000:0`, vendor code `15b3`, and device code `1003` is available on the host. The specification for the `ids=` module parameter makes this card type eligible for the vfio_pci device driver.

```
# lspci -n
0001:000:000:0 15b3:1003
# modprobe vfio_pci ids=15b3:1003
```

Kernel parameter

If the vfio_pci device driver is compiled into the kernel, you can use the `vfio_pci.ids=` kernel parameter to specify the PCIe card types.

vfio_pci kernel parameter syntax



Example: The following specification for the `vfio_pci.ids=` kernel parameter makes the card type eligible for the vfio_pci device driver.

```
vfio_pci.ids=15b3:1003
```

On a running host, you can use the `/sys/bus/pci/drivers/vfio-pci/new_id` sysfs attribute to enable the vfio_pci device driver to control a particular PCIe card type. Write the vendor code and device code, separated by a blank, to the attribute.

Example: This example, makes cards with vendor code `15b3` and device code `1003` eligible for the vfio_pci device driver.

```
# echo 15b3 1003 > /sys/bus/pci/drivers/vfio-pci/new_id
```

After setting up the vfio_pci device driver for one or more PCIe card types, all cards of these types that are freed from their default device driver are assigned to the vfio_pci device driver.

Assigning a PCI device to the vfio_pci device driver

For devices that are not managed by libvirt, write the function address of the PCIe device to the `unbind` attribute of its device driver.

```
# echo <function_address> > /sys/bus/pci/drivers/<pci_device_driver>/unbind
```

Tip: Issue **`lspci -v`** to find out which device driver controls the device of interest.

Example:

```
~]# lspci -v
0001:00:00.0 Ethernet controller: Mellanox Technologies MT27500 Family [ConnectX-3]
Subsystem: Mellanox Technologies Device 048d
Physical Slot: 00000015
...
Kernel driver in use: mlx4_core
Kernel modules: mlx4_core
# echo 0001:00:00.0 > /sys/bus/pci/drivers/mlx4_core/unbind
```

Setting up VFIO pass-through DASDs

Free DASDs from host control and assign them to a VFIO mediated devices.

Before you begin

You must know the subchannel bus ID that maps to the DASD. You can find this information with the **`lscss`** command.

Example: In this example, a DASD with bus ID `0.0.3000` has a subchannel with bus ID `0.0.0004`.

```
# lscss -d 0.0.3000
Device   Subchan. DevType CU Type Use PIM PAM POM CHPIDs
----------------------------------------------------------------------
0.0.3000 0.0.0004 3390/0A 3990/E9 YES C0  C0  FF  34400000 00000000
```

Procedure

1. Ensure that the vfio_ccw device driver and the vfio_mdev modules are loaded. If either module is compiled as a separate module, you might have to load it yourself.

   ```
   # modprobe vfio_ccw
   # modprobe vfio_mdev
   ```

2. Free the DASD and its subchannel from the default device drivers on the host and assign the subchannel to the `vfio_ccw` device driver.

   Using the `driverctl` command

   The preferred method for this reassignment is issuing a **`driverctl`** command of this form:

```
# driverctl --bus css set-override <subchannel_bus_id> vfio_ccw
```

By default, these configuration changes persist across reboots of the KVM host. To apply changes only to the active configuration, specify the `--nosave` option with the command.

Example: The command in this example, frees a DASD that corresponds to subchannel ID `0.0.0004` from the dasd-eckd device driver and persistently reassigns the subchannel to the `vfio_ccw` device driver.

```
# driverctl --bus css set-override 0.0.0004 vfio_ccw
```

Using general Linux commands

As a non-persistent fallback method, you can use general Linux commands.

Write the bus ID of the DASD to the `unbind` attribute of its device driver module. Then, write the bus ID of the subchannel to the `unbind` attribute the subchannel device driver. Finally, assign subchannel to the `vfio_ccw` device driver by writing the subchannel bus-ID to `/sys/bus/css/drivers/vfio_ccw/bind`.

```
# echo <device_bus_id> > /sys/bus/ccw/drivers/dasd-eckd/unbind
# echo <subchannel_bus_id> > /sys/bus/css/devices/<subchannel_bus_id>/driver/unbind
# echo <subchannel_bus_id> > /sys/bus/css/drivers/vfio_ccw/bind
```

Example: The first commands in this example frees a DASD with bus ID `0.0.3000` from the dasd-eckd device driver. The commands that follow reassign the subchannel to the `vfio_ccw` device driver.

```
# echo 0.0.3000 > /sys/bus/ccw/drivers/dasd-eckd/unbind
# echo 0.0.0004 > /sys/bus/css/devices/0.0.0004/driver/unbind
# echo 0.0.0004 > /sys/bus/css/drivers/vfio_ccw/bind
```

3. Create a new VFIO CCW mediated device. You can use libvirt or general Linux commands to create the device.

   Using libvirt

   The preferred method is using libvirt. With libvirt, you can create persistent or transient mediated devices. See *KVM Virtual Server Management*, SC34-2752.

   Using general Linux commands

   Mediated devices that you create with the commands that follow do not persist across reboots.

   Writing a universally unique identifier (UUID) to `/sys/bus/css/devices/<subchannel_bus_id>/mdev_supported_types/vfio_ccw-io/create`, where *<subchannel_bus_id>* is the subchannel of the DASD.

   Tip: Use the **uuidgen** command to generate a UUID.

   Example: This example creates a CCW mediated device for subchannel `0.0.0004`.

   ```
   # uuidgen
   18e124fb-b2fc-47f6-a407-f256b6c49767
   # echo 18e124fb-b2fc-47f6-a407-f256b6c49767 > \
   /sys/bus/css/devices/0.0.0004/mdev_supported_types/vfio_ccw-io/create
   ```

4. Optional: Display subchannel information for the mediated device, by issuing **lscss** with the `--vfio` option.

   Example:

   ```
   $ lscss --vfio
   MDEV                                 Subchan.  PIM PAM POM   CHPIDs
   ----------------------------------------------------------------------------
   18e124fb-b2fc-47f6-a407-f256b6c49767  0.0.0004  c0  c0  ff 0  5020000 00000000
   ```

Setting up cryptographic adapter resources for VFIO pass-through

Free cryptographic adapter resources from host control and assign them to a VFIO AP mediated device.

AP queues

Cryptographic adapter resources are managed as AP queues (see “Cryptographic domains” on page 492). An AP queue corresponds to a specific cryptographic domain on a specific cryptographic adapter.

Persistence across host reboots

You can set up your cryptographic resources such that, after a host reboot, your mediated devices are ready to be attached to KVM guests, without further configuration steps. This setup requires a persistent assignment of AP queues to the `vfio_ap` device driver and persistent mediated devices with their assignment of AP queues.

Use the **chzdev** command to persistently assign AP queues to the `vfio_ap` device driver. Use the **nodedev-define** command to make a mediated device and its assignment of AP queues persistent.

Procedure

1. Ensure that the vfio_ap device driver is loaded. If it is compiled as a separate module, you might have to load it yourself. The following command loads vfio_ap and any additional modules it requires.

```
# modprobe vfio_ap
```

2. Assign AP queues to the `vfio_ap` device driver.

   Two 256-bit masks, one for adapters and the other for domains, rule which AP queues are controlled by the zcrypt device driver. Unless you change the initial setting with the `ap.apmask=` and `ap.aqmask=` kernel parameters, both masks have all bits on by default. With all bits set in the masks, all AP queues that are available to the host can be accessed only by the host itself.

   Setting the bit for a particular adapter to 0 frees all queues of that adapter from the host and makes them available for use by vfio_ap. Similarly, setting the bit for a particular domain to 0 frees all queues of that domain, across all adapters, from the host and makes them available for use by vfio_ap.

   Set bits for adapters or domains or both to `0` to assign queues to the `vfio_ap` device driver.

   Using the chzdev command
   The preferred method for changing the masks is the **chzdev** command with the `--type ap` option. With the **chzdev** command, you can change the masks persistently, for the active configuration, or both. For details, see *KVM Virtual Server Management*, SC34-2752.

   Writing to the masks in sysfs
   As a non-persistent fallback method, you can directly write to the masks in sysfs, see “Freeing AP queues for KVM guests” on page 510.

3. Create and configure a mediated device.

   Using libvirt
   The preferred method for creating and configuring a mediated device is using the **virsh nodedev-define** command and a node-device XML description. With this command, you can create a persistent mediated device, see *KVM Virtual Server Management*, SC34-2752.

   Using general Linux commands and sysfs
   As a non-persistent fallback method, you can create VFIO mediated device for the `vfio_ap` device driver by writing a universally unique identifier (UUID) to `/sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-passthrough/create`.

   You then configure the mediated device by writing to the attributes within the sysfs representations of VFIO AP mediated device. VFIO AP mediated devices are represented as subdirectories of `/sys/devices/vfio_ap/matrix`. The directory names match the UUIDs that identify the mediated devices.

Use the attributes in a device directory to configure the device and to obtain information about the device.

Table 60. sysfs attributes of VFIO mediated devices for cryptographic adapter resources

Attribute	Explanation
assign_adapter	Write an adapter ID to this attribute to assign the adapter to the mediated device. Specify the adapter ID in decimal or hexadecimal notation. For hexadecimal notation, use the prefix "0x". Example: `# echo 0x0a > assign_adapter`
assign_control_domain	Write a domain ID to this attribute to assign the domain as a control domain to the mediated device. Assign a control domain for each usage domain that you assign to the mediated device, so that you can manage your domains from the guest that uses the mediated device. Specify the domain ID in decimal or hexadecimal notation. For hexadecimal notation, use the prefix "0x". Example: `# echo 0x001b > assign_control_domain`
assign_domain	Write a domain ID to this attribute to assign a usage domain to the mediated device. Specify the domain ID in decimal or hexadecimal notation. For hexadecimal notation, use the prefix "0x". Example: `# echo 0x001b > assign_domain`

control_domains	Read this attribute to list the assigned control domains.
	Example:
	`# cat control_domains`
	`001b`

matrix	Read this attribute to list the assigned AP queues that result from the adapter and domain assignments.
	Example:
	`# cat matrix`
	`0a.001b`

mdev_type	Symbolic link that points to the `vfio_ap-passthrough` directory.
remove	Write 1 to this attribute to remove the mediated device. Example: `# echo 1 > remove`
subsystem	Symbolic link that points to the matrix bus.

Table 60. sysfs attributes of VFIO mediated devices for cryptographic adapter resources (continued)

Attribute	Explanation
unassign_adapter	Write an adapter ID to this attribute to remove the adapter from the mediated device. Specify the adapter ID in decimal or hexadecimal notation. For hexadecimal notation, use the prefix "0x". Example: `# echo 0x0a > unassign_adapter`
unassign_control_domain	Write a domain ID to this attribute to remove the domain from the control domains of the mediated device. Specify the domain ID in decimal or hexadecimal notation. For hexadecimal notation, use the prefix "0x". Example: `# echo 0x001b > unassign_control_domain`
unassign_domain	Write a domain ID to this attribute to remove a the domain from the usage domains of the mediated device. Specify the domain ID in decimal or hexadecimal notation. For hexadecimal notation, use the prefix "0x". Example: `# echo 0x001b > unassign_domain`

Scenario: Fallback method for creating and configuring a mediated device

This scenario assumes a KVM host on which 8 AP queues have been freed from control of the zcrypt device driver. A matrix of two of these queues are to be assigned to a mediated device.

Note: The steps that follow describe a fallback method that uses common Linux commands to directly write to sysfs. This method does not create a persistent mediated device. The preferred method for creating and configuring mediated devices is an XML description of the mediated device and the **`virsh nodev-define`** command, see *KVM Virtual Server Management*, SC34-2752.

1. Load the required modules.

```
# modprobe vfio_ap
```

2. List the eligible AP queues. AP queues are eligible only if they are controlled by the vfio_ap device driver.

```
# lszcrypt -V | grep vfio
00.0001     CEX7A   Accelerator  online         0        0     13     08 -MC-A-NF-  vfio_ap
00.0002     CEX7A   Accelerator  online         0        0     13     08 -MC-A-NF-  vfio_ap
00.0004     CEX7A   Accelerator  online         0        0     13     08 -MC-A-NF-  vfio_ap
00.001b     CEX7A   Accelerator  online         0        0     13     08 -MC-A-NF-  vfio_ap
0a.0001     CEX7P   EP11-Coproc  online         0        0     13     08 -----XNF-  vfio_ap
0a.0002     CEX7P   EP11-Coproc  online         0        0     13     08 -----XNF-  vfio_ap
0a.0004     CEX7P   EP11-Coproc  online         0        0     13     08 -----XNF-  vfio_ap
0a.001b     CEX7P   EP11-Coproc  online         0        0     13     08 -----XNF-  vfio_ap
```

The output shows that 8 AP queues are eligible. The eight queues correspond to a matrix of two adapters, `0x00` and `0x0a` and four domains, `0x0001`, `0x0002`, `0x0004`, and `0x001b`. These adapters and domains are the only ones that you can assign to the mediated device.

3. Create a mediated device.

```
# uuidgen
4b0518fd-9237-493f-93c8-c5597f8006a3
# echo 4b0518fd-9237-493f-93c8-c5597f8006a3 \
> /sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-passthrough/create
```

4. Add adapters `0x00` and `0x0a` to the device.

```
# echo 0x00 > /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/assign_adapter
# echo 0x0a > /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/assign_adapter
# cat /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/matrix
00.
0a.
```

Reading the `matrix` attribute does not yield any AP queues. To assign AP queues both adapters and domains must be added.

5. Add domain `0x001b`.

```
# echo 0x001b > /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/assign_domain
# cat /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/matrix
00.001b
0a.001b
# cat /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/control_domains
```

The content of the `matrix` attribute shows that two AP queues are assigned to the mediated device. The two queues correspond to a matrix of two adapters, `0x00` and `0x0a` and one domain, `0x001b`. Reading the `control_domains` attribute shows that no control domain is configured.

6. Add domain `0x001b` to the control domains.

```
# echo 0x001b > /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/
assign_control_domain
# cat /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/control_domains
001b
```

Part 8. Security

These device drivers and features support security aspects of Red Hat Enterprise Linux 9.1 for IBM Z.

Newest version

You can find the newest version of this book at
ibm.com/docs/en/linux-on-systems?topic=linuxone-distributions

Restrictions

For prerequisites and restrictions see the IBM Z architecture specific information in the Red Hat Enterprise Linux 9.1 release notes at
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

Chapter 49. Generic cryptographic device driver

The generic cryptographic device driver (zcrypt) supports cryptographic coprocessor and accelerator hardware. Cryptographic coprocessors provide secure key cryptographic operations for the IBM Common Cryptographic Architecture (CCA). Cryptographic coprocessors also support Enterprise PKCS #11 (EP11). CCA and EP11 coprocessors operate as Hardware Security Modules (HSMs).

Some cryptographic processing in Linux can be offloaded from the CPU and performed by dedicated CCA or EP 11 coprocessors or accelerators. Several of these CCA or EP 11 coprocessors and accelerators are available offering a range of features. The generic cryptographic device driver (zcrypt) is required to use any available cryptographic hardware for processor offload.

Features

The generic cryptographic device driver supports a range of hardware and software functions.

Supported cryptographic adapters

The cryptographic hardware feature might contain one or two cryptographic adapters. Each adapter can be configured either as a CCA coprocessor or as an accelerator. The CEX8S, CEX7S, CEX6S and CEX5S cryptographic adapters can also be configured as EP11 coprocessors.

The following types of cryptographic adapters are supported:

- Crypto Express8S (EP11) Coprocessor (CEX8P)
- Crypto Express8S (CCA) Coprocessor (CEX8C)
- Crypto Express8S Accelerator (CEX8A)
- Crypto Express7S (EP11) Coprocessor (CEX7P)
- Crypto Express7S (CCA) Coprocessor (CEX7C)
- Crypto Express7S Accelerator (CEX7A)
- Crypto Express6S (EP11) Coprocessor (CEX6P)
- Crypto Express6S (CCA) Coprocessor (CEX6C)
- Crypto Express6S Accelerator (CEX6A)
- Crypto Express5S (EP11) Coprocessor (CEX5P)
- Crypto Express5S Accelerator (CEX5A)
- Crypto Express5S (CCA) Coprocessor (CEX5C)

For information about setting up your cryptographic environment on Linux under z/VM, see *z/VM: Secure Configuration Guide*, SG24-6323 and *Security for Linux on System z*, SG24-7728.

Supported facilities

The cryptographic device driver supports several cryptographic accelerators as well as CCA and EP11 coprocessors.

Cryptographic accelerators support clear key cryptographic algorithms. In particular, they provide fast RSA encryption and decryption for key sizes 1024 - 4096-bit.

Cryptographic coprocessors act as a hardware security module (HSM) and provide secure key cryptographic operations for the IBM Common Cryptographic Architecture (CCA) and the Enterprise PKCS#11 feature (EP11).

For more information about CCA, see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294. You can obtain this book at ibm.com/docs/en/linux-on-systems?topic=overview-secure-key-solution-cca-application-programmers-guide.

For more information about EP11, see *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713. You can obtain this publication at ibm.com/docs/en/linux-on-systems?topic=security-cryptographic-hardware-support.

Cryptographic coprocessors also provide clear key RSA operations for 1024-bit, 2048-bit, and 4096-bit keys, and a true random number generator for `/dev/hwrng`. The EP11 coprocessor supports only secure key operations.

Hardware and software prerequisites

Support for the Crypto Express features depends on the IBM Z hardware model.

Table 61 on page 490 lists the support for the cryptographic adapters.

Table 61. Support for cryptographic adapters by mainframe model

Cryptographic adapters	**Mainframe support**
CEX8A, CEX8C, and CEX8P	IBM z16
CEX7A, CEX7C, and CEX7P	IBM z16, z15 and LinuxONE III
CEX6A, CEX6C, and CEX6P	IBM z16, z15, z14, z14 ZR1, LinuxONE III, and LinuxONE II
CEX5A, CEX5C, and CEX5P	z15, z14, z13, z13s, LinuxONE III, LinuxONE II, and LinuxONE

Crypto Express8S adapters can handle larger request and reply sizes than earlier cryptographic adapters. The software that manages Crypto Express8S adapters must support these larger request and reply sizes.

Table 62 on page 490 lists the required software by function.

Table 62. Required software

Software required	**Function that is supported by the software**
The CCA library	For the secure key cryptographic functions on CCA coprocessors. For information about cryptographic CCA coprocessors, coexistence of adapter versions, and how to use CCA functions, see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294. You can obtain this publication at ibm.com/docs/en/linux-on-systems?topic=overview-secure-key-solution-cca-application-programmers-guide.
The EP11 library	For the secure key cryptographic functions on EP11 coprocessors. See *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713. You can obtain this publication at ibm.com/docs/en/linux-on-systems?topic=overview-exploiting-enterprise-pkcs-11-using-opencryptoki.
The libica library	For the clear key cryptographic functions. See *libica Programmer's Reference*, SC34-2602. You can obtain this publication at ibm.com/docs/en/linux-on-systems?topic=overview-libica-programmers-reference.
APAR VM65942	To support z14 hardware and the CEX6S adapter for Linux on z/VM. To support ECC with a shared cryptographic adapter in coprocessor mode for Linux on z/VM 6.4.
APARs VM66428 and VM66206	To support z15 hardware and the CEX7S adapter for Linux on z/VM.

You can download the CCA library and the EP11 library from the IBM cryptographic coprocessor web page at

```
www.ibm.com/security/cryptocards
```

Note: The CCA library works with 64-bit applications only.

What you should know about the cryptographic device driver

Your use of the cryptographic device driver and the cryptographic hardware might require additional software. There are special considerations for Linux on z/VM, for performance, and for specific cryptographic operations.

Functions provided by the cryptographic device driver

The functions that are provided by the cryptographic device driver depend on whether the device driver finds a cryptographic accelerator, a CCA coprocessor, or an EP11 coprocessor.

For both accelerators and CCA coprocessors, the device driver provides Rivest-Shamir-Adleman (RSA) encryption and RSA decryption functions using clear keys. RSA operations are supported in both the modulus-exponent and the Chinese-Remainder Theorem (CRT) variants for any modulus in the range 57 - 4096 bit.

For CCA coprocessors, the device driver also provides a function to pass CCA requests to the cryptographic coprocessor and an access to the true random number generator of the CCA coprocessor.

For EP11 coprocessors, the device driver provides functionality to pass EP11 requests to the cryptographic coprocessor.

Adapter discovery

The cryptographic device driver provides two misc device nodes, one for cryptographic requests, and one for a device from which random numbers can be read.

Cryptographic adapters are detected automatically when the module is loaded. They are reprobed periodically, and following any hardware problem.

Upon detection of a cryptographic adapter, the device driver presents a Linux misc device, `z90crypt`, to user space. A user space process can open the misc device to submit cryptographic requests to the adapter through IOCTLs.

If at least one of the detected cryptographic adapters is a coprocessor, an additional misc device, `hwrng`, is created from which random numbers can be read.

You can set cryptographic adapters online or offline in the device driver. The cryptographic device driver ignores adapters that are configured offline even if the hardware is detected. The online or offline configuration is independent of the hardware configuration.

Request processing

Cryptographic adapters process requests asynchronously.

The device driver detects request completion either by standard polling, a special high-frequency polling thread, or by hardware interrupts. If hardware interrupt support is available, the device driver does not use polling to detect request completion.

All requests to either of the two misc devices are routed to a cryptographic adapter using a crypto request scheduling function that, for each adapter, takes into account:

- The supported functions
- The number of pending requests
- A speed rating

The cryptographic device driver supports multiple cryptographic domains simultaneously, see “Cryptographic domains” on page 492.

The device driver uses at least one domain for all adapters. If none is given, the kernel selects a default domain. Alternatively, you can select the default domain using a kernel parameter (see “Kernel parameters” on page 495).

Cryptographic domains

Crypto Express hardware adapters are divided into multiple domains, also called *cryptographic domains* or *AP domains*.

Each domain acts as an independent cryptographic device with its own state, including its own master key. Two domains in the same Crypto Express adapter are completely isolated and cannot access each other's states. The maximum number of domains depends on the mainframe model and is the same for all Crypto Express adapters in that mainframe. For example, a z13 supports up to 85 domains (with hexadecimal domain IDs 0000 to 0054).

Cryptographic devices on LPARs

When you assign adapters and domains to an LPAR on the HMC or SE, you indirectly assign virtual cryptographic devices.



For example, assigning adapter ID 00 and 02 as well as domains 0002, 0003, and 0005 to an LPAR implicitly assigns six virtual cryptographic devices to the LPAR: (00,0002), (00,0003), (00,0005), (02,0002), (02,0003), and (02,0005).

You can choose between two types of access to a cryptographic domain:

To use cryptographic functions.
A domain that is assigned to an LPAR for usage access is called a *usage domain* of that LPAR on the HMC or SE.

To manage or control the domain, including the management of the master keys.
A domain that is assigned to an LPAR for management (control) access is called a *control domain* of that LPAR on the HMC or SE.

Every usage domain of an LPAR must also be a control domain of that LPAR.

The list of usage domains and the list of adapter IDs define the list of virtual cryptographic devices that are assigned to an LPAR. For example, if 00 is an adapter ID and 0002 is a usage domain ID, then the virtual cryptographic device (00,0002) is assigned to the LPAR.

Cryptographic devices on z/VM

In z/VM, the virtual cryptographic devices available to a guest are defined by using the CRYPTO directory statement:

- The `CRYPTO APDEDICATE` statement assigns domain IDs and adapter IDs to the guest. This assignment implicitly defines a list of dedicated virtual cryptographic devices. All virtual cryptographic devices that are determined by an ID from the adapter list of that guest and an ID from the domain list of that guest are assigned to the guest.
- The `CRYPTO APVIRT` statement assigns one virtual cryptographic device that can be shared among multiple guests with a guest-specific virtualized adapter ID and a virtualized domain ID.

Virtual cryptographic devices in z/VM can be either shared or dedicated, but not both.

Linux on z/VM with access to a shared cryptographic accelerator can either observe an accelerator or a CCA coprocessor.

For shared cryptographic CCA coprocessors, the following functions are available to the Linux instance:

- Random number functions
- Clear-key RSA functions
- If APAR VM65942 has been installed: Clear-key ECC functions

Other requests are rejected by z/VM. For more information about supported functions, see the z/VM publications.

Cryptographic devices on Linux

In Linux, virtual cryptographic devices are called *AP queues*. The name of an AP queue consists of two parts, the adapter ID and the domain ID, both in hexadecimal notation. For example, if cryptographic adapters with the IDs 00 and 02 are selected, and the domain IDs 0002, 0003 and 0005 have been configured on the cryptographic adapter, then the following AP queues are defined to Linux:

```
/sys/devices/ap/card00/00.0002
/sys/devices/ap/card00/00.0003
/sys/devices/ap/card00/00.0005
/sys/devices/ap/card02/02.0002
/sys/devices/ap/card02/02.0003
/sys/devices/ap/card02/02.0005
```

Cryptographic devices and KVM

A KVM host can pass AP queues on to its guests. Before an AP queue can be configured for a KVM guest, two steps are required on the host.

1. The AP queue must be freed from control of the zcrypt device driver.
2. The AP queue must be configured for a VFIO mediated device that is controlled by the vfio_ap device driver.

For more information, see “Setting up cryptographic adapter resources for VFIO pass-through” on page 482.

AP queue status overview

Multiple configuration steps are required to make AP queues available to user space programs on Linux.

Linux reflects the configuration progress in two configuration states of an AP queue:

LPAR configuration status
a hardware status that can either be "`configured`" or "`not configured`".

Online status
a state, "`online`" or "`offline`", that is controlled by the zcrypt device driver and that is maintained by the Linux kernel.

Changing the online status of an AP queue does not affect its LPAR configuration status. In contrast, changing the LPAR configuration status can affect a queue's online status. For example, changing the LPAR configuration status from "`not configured`" to "`configured`" also changes its online status from offline to online. AP queues that are not configured are always offline. Configured AP queues can be online or offline, but an AP queue that is not configured cannot be online.

An AP queue is available to cryptographic applications on Linux if its LPAR configuration status is "`configured`" and its on line status is "`online`".



Figure 106. AP queue configuration steps and status

Figure 106 on page 494 illustrates the steps for making AP queues available and how the steps affect the configuration states.

Assigning adapters and domains to the LPAR
You use the HMC to assign cryptographic adapters and domains to an LPAR. Implicitly, this assignment provides a matrix of AP queues to the LPAR. These queues can now be detected by operating systems in the LPAR.

Setting the LPAR configuration status of adapters
Before AP queues can be used by an operating system, the LPAR configuration status of the corresponding adapter must be set to "`configured`". AP queues inherit the LPAR configuration status of the associated adapter.

Depending on the version, your HMC interface might show AP queues in the state "`not configured`" as "`Candidate`" and AP queues in the state "`configured`" as "`Candidate and Online`". Do not confuse this online designation with the online status in Linux.

Although the LPAR configuration status is controlled by the HMC, you can trigger this setting from Linux, see “Setting the LPAR configuration status” on page 505.

Controlling the online status of AP queues in Linux
An AP queue can be used by a cryptographic application if it is online within Linux. Initially, all AP queues of an AP adapter are online when the LPAR configuration status of the adapter becomes "`configured`".

For information about setting AP queues offline and back online, see “Setting devices online or offline” on page 506.

Setting up the cryptographic device driver

Configure the cryptographic device driver through the `ap.domain=` and the `ap.poll_thread=` kernel parameters. You might also have to set up libraries.

The cryptographic device driver consists of multiple, separate modules:

zcrypt
Cryptographic Coprocessor interface, Cryptographic Coprocessor message type 6, Cryptographic Coprocessor message type 50. Support for message type 6 includes secure key and RNG requests. Support for message type 50 includes RSA requests for both modulus-exponent and Chinese-Remainder Theorem variants.

zcrypt_cex4
device driver for CEX7S, CEX6S, and CEX5S adapters.

For information about setting up cryptographic hardware on your mainframe, see *zSeries Crypto Guide Update*, SG24-6870.

Kernel parameters

You can configure the cryptographic device driver by adding parameters to the kernel parameter line.



where

<domain>
is an integer that identifies the default cryptographic domain for the Linux instance. You define cryptographic domains in the LPAR activation profile on the HMC or SE.

The default value (**`ap.domain=autoselect`**) causes the device driver to choose one of the available domains automatically.

Important: Be sure to enter an existing domain. The Trusted Key Entry (TKE) workstation does not find the cryptographic adapters if a non-existing domain is entered here. All CCA applications use the default domain, and do not work correctly if the specified domain does not exist.

<poll_thread>
is an integer argument and enables a polling thread to tune cryptographic performance. Valid values are 1 (enabled) or 0 (disabled, this value is the default). For details, see "Setting the polling thread" on page 507.

Note: If you are running Linux in an LPAR, AP interrupts are used instead of the polling thread. The polling thread is disabled when AP interrupts are available. See “Using AP adapter interrupts” on page 508.

ap.apmask= and ap.aqmask=

are two 256-bit masks that specify which AP queues are controlled by the zcrypt device driver and which are available to alternative device drivers, like vfio_ap.

If your Linux instance is a KVM host, the vfio_ap device driver controls AP queues on behalf of KVM guests. While vfio_ap is the only eligible alternative device driver, freeing an AP queue from zcrypt makes it available to vfio_ap. For more information, see “Setting up cryptographic adapter resources for VFIO pass-through” on page 482.

Each bit of the `ap.apmask=` mask addresses a cryptographic adapter. The leftmost bit corresponds to the adapter with ID `0x00`. Generally, the bit number in the mask corresponds to the decimal value of the adapter ID. If an adapter bit is set to 0, all AP queues for this adapter are available to alternative device drivers, across all domains.

Each bit of the `ap.aqmask=` mask addresses a cryptographic domain. The leftmost bit corresponds to domain `0x0000`. Generally, the bit number in the mask corresponds to the decimal value of the domain ID. If a domain bit is set to 0, all AP queues with this domain are available to alternative device drivers, across all adapters.

AP queues are controlled by the zcrypt device driver if both the bit for its adapter and for its domain are set to 1. An AP queue is available to alternative device drivers if the bit for its adapter, or its domain, or both are set to 0.

By default, all bits in both masks are 1. Therefore, zcrypt is the default device driver for all AP queues.

<hex_mask>
specifies a replacement for the default mask. Valid values are `0x` followed by 1 - 64 hexadecimal digits. If fewer than 64 digits are specified, the specification is padded with 0s on the right. The value is big-endian. The hexadecimal representation is mapped to the 256-digit binary mask.

<bit>
specifies an individual bit number. 0 specifies the leftmost bit. With the plus sign (+) prefix, the bit is set to 1. With the minus sign (-) prefix, the bit is set to 0.

<from_bit>-<to_bit>
specifies the range of bits from bit number *<from_bit>* to bit number *<to_bit>*. With the plus sign (+) prefix, all bits in the range are set to 1. With the minus sign (-) prefix, all bits in the range are set to 0.

Bit numbers can be in decimal or hexadecimal notation. Hexadecimal numbers must be prefixed with `0x`. You can specify a comma-separated list of bits and ranges. Such lists are processed left to right.

The hexadecimal representations of the masks are available in sysfs. Using sysfs, you can change the masks on a running Linux instance, see “Freeing AP queues for KVM guests” on page 510.

Examples

- The following kernel parameter line specification makes the zcrypt device driver operate within the default cryptographic domain "7" with `ap.poll_thread=` enabled:

```
ap.domain=7 ap.poll_thread=1
```

- The following specification makes all AP queues on adapter `0x00` and any adapters with IDs greater than `0x07` and all AP queues with domain ID, `0x0007`, `0x0008`, `0x0009`, `0x000a`, and `0x000b` available to the vfio_ap device driver.

```
ap.apmask=0x7F ap.aqmask=-7-11
```

The following specification with hexadecimal notation for the queue range is equivalent:

```
ap.apmask=0x7F ap.aqmask=-0x7-0xb
```

Accessing cryptographic devices

The cryptographic device driver registers as a misc device and provides a default device node to user space.

In Red Hat Enterprise Linux 9.1, udev creates the device node `/dev/z90crypt` for you. The device node is assigned to the miscellaneous devices.

The `/dev/z90crypt` device node provides unrestricted user space access to a device that represents all AP queues that are available to the Linux instance.

You can create customized device nodes that represent subsets of AP queues and functions, see "Creating customized device nodes" on page 497.

Creating customized device nodes

The cryptographic device driver can provide and maintain multiple zcrypt device nodes. These nodes can be restricted in terms of cryptographic adapters, domains, and available IOCTLs.

About this task

You can create device nodes with access to a subset of the AP queues that are available to the Linux instance and that can perform a subset of the functions. Such a device node can be used for access control to cryptographic resources:

- Selective assignment of device nodes to Linux containers.
- Linux file permissions for the device nodes can be used to restrict the access for users and groups.

Procedure

1. Create a new device node by issuing a **zcryptctl** command of this form:

   ```
   # zcryptctl create <name>
   ```

   where *<name>* is a unique device name. A device node `/dev/`*<name>* and a device directory `/sys/devices/virtual/zcrypt/`*<name>* are created in sysfs.

   For more information about **zcryptctl**, see "zcryptctl - Control access to AP queues and functions" on page 771.

   Example:

   ```
   # zcryptctl create node_1
   ```

   The example creates a device node `/dev/node_1` and a device directory `/sys/devices/virtual/zcrypt/node_1` in sysfs.
2. Set the adapters for the new device node. Issue a **zcryptctl** command of this form:

   ```
   # zcryptctl addap <name> <adapter_id_1>,<adapter_id_2>,<adapter_id_3>,...
   ```

   where *<adapter_id_n>* specifies an adapter to which you want this node to have access. You can use the hexadecimal adapter IDs or their equivalent decimal values. Hexadecimal specifications must be prefixed with `0x`.

   Example:

   ```
   # zcryptctl addap node_1 0x05,0x06,0x07,0x0a
   ```

Using decimal notation this command would be:

```
# zcryptctl addap node_1 5,6,7,10
```

3. Set the domains for the new device node. Issue a **zcryptctl** command of this form:

```
# zcryptctl adddom <name> <dom_1>,<dom_2>,<dom_3>,...
```

where *<dom_n>* is a domain to which you want this node to have access. You can use the hexadecimal domain IDs or their equivalent decimal values. Hexadecimal specifications must be prefixed with `0x`.

Example:

```
# zcryptctl adddom node_1 0x0006
```

4. Set the IOCTLs for the new device node. Issue a **zcryptctl** command of this form:

```
# zcryptctl addioctl <name> <ioctl_1>,<ioctl_2>,<ioctl_3>,...
```

Set IOCTLs according to the functions you want to support. The following table lists the IOCTLs that are required by the CCA, EP11, and libica library.

Table 63. IOCTLs required by cryptographic libraries

Library	Functions	Required IOCTLs
CCA	Secure key cryptographic functions on CCA coprocessors.	ZSECSENDCPRB
EP11	Secure key cryptographic functions on EP11 coprocessors.	ZSENDEP11CPRB
libica	Clear key cryptographic functions.	ICARSAMODEXPO, ICARSACRT, ZSECSENDCPRB

The available IOCTLs are listed in `arch/s390/include/uapi/asm/zcrypt.h` in the Linux source tree.

Example:

```
# zcryptctl addioctl node_1 ZSECSENDCPRB
```

5. Optional: Secure the device node through suitable settings for the file owner and group, and through access permissions for user, group, and others.

Results

Changes to the masks are instantly applied and affect all applications with an open file descriptor for this zcrypt node immediately.

Example

To create and define a zcrypt device node for CCA requests on adapters `0x05`, `0x06`, `0x07`, `0x0a` and domain `0x0006` using the **zcryptctl** command:

```
# zcryptctl create node_1
# zcryptctl addap node_1 0x05,0x06,0x07,0x0a
# zcryptctl adddom node_1 0x0006
# zcryptctl adioctl node_1 ZSECSENDCPRB
```

It is equivalent to using the **zcryptctl config** command with the following configuration file entry:

```
# node 1 for CCA requests on domain 6 - hexadecimal notation
node = node_1
aps = 0x05,0x06,0x07,0x0a
doms = 0x0006
ioctls = ZSECSENDCPRB
```

The following equivalent configuration file uses decimal notation for adapters and domains:

```
# node 1 for CCA requests on domain 6 - decimal notation
node = node_1
aps = 5,6,7,10
doms = 6
ioctls = ZSECSENDCPRB
```

Alternatively, you can use sysfs attributes to obtain the same results:

```
# echo node_1 > /sys/class/zcrypt/create
# echo +0x05,+0x06,+0x07,+0x0a > /sys/devices/virtual/zcrypt/node_1/apmask
# echo +0x0006 > /sys/devices/virtual/zcrypt/node_1/aqmask
# echo +0x81 > /sys/devices/virtual/zcrypt/node_1/ioctlmask
```

The `apmask` and `aqmask` attributes in the node directory follow the same syntax as the `apmask` and `aqmask` attributes at `/sys/bus/ap` (see "Freeing AP queues for KVM guests" on page 510). Relative values require a plus (+) or minus (-) prefix, can use decimal or hexadecimal notation, and can address individual bits or ranges. You can also specify the complete hexadecimal mask as an absolute value. The sysfs interface requires numeric values for the IOCTLs as listed in `arch/s390/include/uapi/asm/zcrypt.h`.

What to do next

You can delete the device node with `zcryptctl destroy <name>`.

Displaying information about the AP bus

Use the **lszcrypt -b** command to display status information about the AP bus; alternatively, you can use sysfs.

About this task

For information about **lszcrypt -b**, see "lszcrypt - Display cryptographic devices" on page 682.

The AP bus is represented in sysfs as a directory of the form

```
/sys/bus/ap
```

The sysfs directory contains a number of attributes with information about the AP bus.

Table 64. AP bus attributes

Attribute	Explanation
ap_domain	Read-write attribute that represents the default domain selected by the kernel. Alternatively, you can select the domain using a kernel parameter, or a module parameter during module load. See "Kernel parameters" on page 495.
ap_max_domain_id	Read-only attribute that represents the largest possible domain ID. Domain IDs can range from 0 to this number, which depends on the mainframe model.

Table 64. AP bus attributes (continued)

Attribute	Explanation
ap_control_domain_mask	Read-only attribute that represents the installed control domain facilities as a 32-byte field in hexadecimal notation. A maximun number of 256 domains can be addressed. Each bit position represents a dedicated control domain.
ap_usage_domain_mask	Read-only attribute that represents the installed usage domain facilities as a 32-byte field in hexadecimal notation. A maximum number of 256 domains can be addressed. Each bit position represents a usage domain.
ap_interrupts	Read-only attribute that indicates whether interrupt handling for the AP bus is enabled.
apmask	Read-write attribute that represents up to 256 cryptographic adapters. The attribute is a 64-digit hexadecimal representation of the 256-digit binary mask. In combination with the `aqmask` attribute, it marks a set of AP queues that are reserved for device drivers other than zcrypt. See "Freeing AP queues for KVM guests" on page 510.
aqmask	Read-write attribute that represents 256 cryptographic domains. The attribute is a 64-digit hexadecimal representation of the 256-digit binary mask. In combination with the `apmask` attribute, it marks a set of AP queues that are reserved for device drivers other than zcrypt. See "Freeing AP queues for KVM guests" on page 510.
bindings	Read-only attribute that shows which ratio of the available AP queues are bound to a device driver. The information has this format: *<nr_of_bound_ap_queues>/<total_nr_of_ap_queues>*. If all available AP queues are bound to a device driver, this ratio is appended with the string `(complete)`. For example, with 8 AP queues, all of which are bound to a device driver, the attribute reads: `8/8 (complete)`.
config_time	Read-write attribute that represents a time interval in seconds used to detect new crypto devices.
poll_thread	Read-write attribute that indicates whether polling for the AP bus is enabled.
poll_timeout	Read-write attribute that represents the time interval of the poll thread in nanoseconds.
scans	Read-only attribute that shows the number of AP bus scans for AP adapters and AP queues that were completed since bus initialization. A value of 1 or greater indicates that the initial AP bus scan is complete.

Example

```
# lszcrypt -b
ap_domain=0x6
ap_max_domain_id=0x54
ap_interrupts are enabled
config_time=30 (seconds)
poll_thread is disabled
poll_timeout=250000 (nanoseconds)
```

Working with cryptographic devices

Typically, cryptographic devices are not directly accessed by users but through user programs. Some tasks can be performed through the sysfs interface.

Displaying information about cryptographic devices and AP queues

Use the **lszcrypt** command to display status information about your cryptographic devices and AP queues; alternatively, you can use sysfs.

About this task

For information about **lszcrypt**, see "lszcrypt - Display cryptographic devices" on page 682.

Each cryptographic adapter is represented in a sysfs directory of the form

```
/sys/bus/ap/devices/card<XX>
```

where *<XX>* is the two-digit hexadecimal device index for each device. For example, device 0x1a can be found under `/sys/bus/ap/devices/card1a`. The sysfs directory contains a number of attributes with information about the AP queue.

Table 65. Cryptographic adapter attributes

Attribute	Explanation
ap_functions	Read-only attribute that represents the function facilities that are installed on this device.
API_ordinalnr	EP11 coprocessors only: Read-only attribute that displays the EP11 adapter's firmware API ordinal number.
chkstop	The adapter is in checkstop state. No requests will be sent to the adapter.
config	Read-write attribute that represents the LPAR configuration status for this adapter, see "Setting the LPAR configuration status" on page 505.
depth	Read-only attribute that represents the input queue length for this device.
FW_version	EP11 coprocessors only: Read-only attribute that shows the major and minor firmware version in the format: *<major_version>.<minor_version>*

Table 65. Cryptographic adapter attributes (continued)

Attribute	Explanation
hwtype	Read-only attribute that represents the numeric hardware type for this device. The following values are defined: **11** CEX5A, CEX5C, or CEX5P adapters. **12** CEX6A, CEX6C, or CEX6P adapters. **13** CEX7A, CEX7C, or CEX7P adapters. **14** CEX8A, CEX8C, or CEX8P adapters. The `hwtype` attribute shows the hardware type as interpreted by the device driver. See also the `raw_hwtype` attribute.
raw_hwtype	Read-only attribute that represents the original hardware type of the cryptographic adapter.
max_msg_size	Read-only attribute that shows the upper limit in bytes that can be used by the AP bus, the zcrypt device driver, and user space applications for requests and replies sent to and received from this adapter.
modalias	Read-only attribute that represents an internally used device bus-ID.
online	Read-write attribute that shows whether the device is online (1) or offline (0).
op_modes	EP11 coprocessors only: Read-only attribute that shows the adapter's enabled modes of operation. Enabled modes are always listed on a single line, with multiple modes separated by spaces. The line is empty if no known mode is enabled. For more information about the modes, see *Enterprise PKCS#11 (EP11) Library structure*. To find this document go to www.ibm.com/common/ssi and search with "WXRDPRAN" in the keyword search field.
pendingq_count	Read-only attribute that represents the number of requests in the hardware queue.
request_count	Read-only attribute that represents the number of requests that are already processed by this device.
requestq_count	Read-only attribute that represents the number of outstanding requests (not including the requests in the hardware queue).
serialnr	For CCA and EP11 coprocessors only: Read-only attribute that shows the adapter serial number. The serial number is a unique ASCII string of 8 characters for CCA coprocessors and 16-characters for EP11 coprocessors.
type	Read-only attribute with a name for the device type. The following types are defined: • CEX5A, CEX5C, CEX5P • CEX6A, CEX6C, CEX6P • CEX7A, CEX7C, CEX7P • CEX8A, CEX8C, CEX8P

Each AP queue is independently configurable and represented in a subdirectory of the cryptographic device it belongs to:

```
/sys/bus/ap/devices/card<XX>/<XX>.<YYYY>
```

where *<XX>* is the adapter ID of the cryptographic device and *<YYYY>* is the domain. For example, a cryptographic device with adapter ID 1a might have domains 5 (0005), 31 (001f), and 77 (004d) configured. The cryptographic device together with its AP queues would be represented in sysfs as:

```
/sys/devices/ap/card1a
/sys/devices/ap/card1a/1a.0005
/sys/devices/ap/card1a/1a.001f
/sys/devices/ap/card1a/1a.004d
```

Actions that you take on the cryptographic device also apply to its associated AP queues. Attributes like type and hwtype are inherited by the AP queues. The sysfs directory contains a number of attributes with information about the AP queues.

Table 66. Attributes of the AP queues

Attribute	Explanation
chkstop	The AP queue is in checkstop state. No requests will be sent to the AP queue. The queue is reset when it exits the checkstop state.
config	Read-only attribute that shows the LPAR configuration status of the AP queue, as "`configured`" (1) or "`not configured`" (0). The configuration status of an AP queue matches the configuration status of its cryptographic adapter.
online	Read-write attribute that shows whether the AP queue is online (1) or offline (0).
interrupt	Read-only attribute that represents the interrupt state (enabled or disabled) of the AP queue, and hence the request queue.
mkvps	Read-only attribute with multiple lines of information about the master key states and verification patterns for CCA or EP11 coprocessors. See “Investigating master key states and verification patterns” on page 503.
op_modes	EP11 coprocessors only: Read-only attribute that shows the adapter's enabled modes of operation. Enabled modes are always listed on a single line, with multiple modes separated by spaces. The line is empty if no known mode is enabled. For more information about the modes, see *Enterprise PKCS#11 (EP11) Library structure*. To find this document go to www.ibm.com/common/ssi and search with "WXRDPRAN" in the keyword search field.
reset	Read-only attribute that indicate the state of pending resets of the AP queues, and hence the request queue.
pendingq_count	Read-only attribute that represents the number of requests in the hardware queue.
request_count	Read-only attribute that represents the number of requests that are already processed by this AP queue.
requestq_count	Read-only attribute that represents the number of outstanding requests (not including the requests in the hardware queue).

Investigating master key states and verification patterns

For information about the master keys on an AP queue and the keys' verification patterns read the queues' `mkvps` sysfs attribute.

In sysfs, AP queues are represented as subdirectories of the cryptographic adapter to which they belong. The paths to the `mkvps` sysfs attribute with the master key states and verification patterns have the following format:

```
/sys/bus/ap/devices/card<XX>/<XX>.<YYYY>/mkvps
```

where *<XX>* is the adapter ID of the cryptographic device and *<YYYY>* is the domain ID. For example, the `mkvps` attribute for an AP queue `01.002a` is at `/sys/bus/ap/devices/card01/01.002a/mkvps`.

The read-only `mkvps` attribute holds multiple lines of information about the master key states and verification patterns. If no valid state information is available, dashes (-) are shown instead of both the state and the verification pattern.

CCA coprocessors

For CCA coprocessors, the `mkvps` attribute shows the state of the AES and APKA key registers (see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294). The information has this format:

```
AES NEW: <new_aes_mk_state> <new_aes_mkvp>
AES CUR: <cur_aes_mk_state> <cur_aes_mkvp>
AES OLD: <old_aes_mk_state> <old_aes_mkvp>
APKA NEW: <new_apka_mk_state> <new_apka_mkvp>
APKA CUR: <cur_apka_mk_state> <cur_apka_mkvp>
APKA OLD: <old_apka_mk_state> <old_apka_mkvp>
```

Where:

<new_aes_mk_state>
: is the key state of the new AES master key, which can be one of the following values: `empty`, `partial`, or `full`.

<cur_aes_mk_state>* and *<old_aes_mk_state>
: are the key states of the current and old AES master key, which can be one of the following values: `valid` or `invalid`.

<new_apka_mk_state>
: is the key state of the new APKA master key, which can be one of the following values: `empty`, `partial`, or `full`.

<cur_apka_mk_state>* and *<old_apka_mk_state>
: are the key states of the current and old APKA master key, which can be one of the following values: `valid` or `invalid`.

<*_*_mkvp>
: *<new_aes_mkvp>*, *<cur_aes_mkvp>*, *<old_aes_mkvp>*, *<new_apka_mkvp>*, *<cur_apka_mkvp>*, and *<old_apka_mkvp>* are all 8-byte hexadecimal master key verification patterns, with a leading `0x`.

  Useful verification patterns are present only for key states `full` and `valid`. For other states, `0x0000000000000000` is shown instead.

The following example shows the information for an AP queue in CCA coprocessor mode:

```
# cat /sys/devices/ap/card01/01.002a/mkvps
AES NEW: empty 0x0000000000000000
AES CUR: valid 0x7d10d17bc8a409c4
AES OLD: invalid 0x0000000000000000
APKA NEW: empty 0x0000000000000000
APKA CUR: valid 0x82a5e2cd5030d5ec
APKA OLD: invalid 0x0000000000000000
```

EP11 coprocessors

For EP11 coprocessors, the information has this format:

```
MK CUR: <cur_mk_state> <cur_mkvp>
MK NEW: <new_mk_state> <new_mkvp>
```

Where:

<new_mk_state>
is the key state of the new master key, which can be one of the following values: `empty`, `uncommitted`, or `committed`.

<cur_mk_state>
is the key state of the current master key, which can be one of the following values: `valid` or `invalid`.

<new_mkvp>* and *<cur_mkvp>
are 32-byte hexadecimal master key verification patterns with a 0x prefix.

Useful verification patterns are present only for key states `committed` and `valid`.

Setting the LPAR configuration status

You can use the HMC or SE to control which of the cryptographic devices that the hardware definition assigns to an LPAR are also available to operating systems within that LPAR. From a Linux instance in LPAR mode, you can control device availability by setting the device configuration status for the LPAR in which it runs.

About this task

The LPAR configuration status of a cryptographic device persists across reboots and also applies if a different operating system is IPLed in the LPAR. Use the **`chzcrypt`** command to set the LPAR configuration status of cryptographic devices.

Cryptographic devices with an LPAR configuration status "`configured`" are available to the operating systems that runs in the LPAR, devices with an LPAR configuration status "`not configured`" are not available. The configuration status of a cryptographic device extends to all associated AP queues.

For more information about the status of AP queues, see “AP queue status overview” on page 493.

Procedure

- Preferably, use the **`chzcrypt`** command with the **`--config-on`** option to configure devices and the associated AP queues for an LPAR, or use the **`--config-off`** option to change the state to "`not configured`" in the LPAR.
  Examples:
  - To configure cryptographic devices (in decimal notation) 0, 1, 4, 5, and 12 on, issue:

    ```
    # chzcrypt --config-on 0 1 4 5 12
    ```

  - To configure all available cryptographic devices and their associated AP queues, issue:

    ```
    # chzcrypt --config-on -a
    ```

  For more information about **`chzcrypt`**, see “chzcrypt - Modify the cryptographic configuration” on page 587.
- Alternatively, write `1` to the `config` sysfs attribute of a cryptographic device to configure the device, or write `0` to set the device status to "`not configured`".
  Examples:
  - To configure a cryptographic device with adapter ID 0x04 for the LPAR, issue:

    ```
    # echo 1 > /sys/bus/ap/devices/card04/config
    ```

– To set the LPAR configuration state of a cryptographic device with adapter ID 0x04 to "`not configured`", issue:

```
# echo 0 > /sys/bus/ap/devices/card04/config
```

– To check the LPAR configuration state of the cryptographic device with adapter ID 0x04, issue:

```
# cat /sys/bus/ap/devices/card04/config
0
```

The value is `1` if the device is configured or `0` otherwise.

Alternatively, use the **`lszcrypt`** command to display the status.

```
# lszcrypt
CARD.DOMAIN TYPE  MODE        STATUS    REQUESTS
-----------------------------------------------
...
04          CEX7A Accelerator deconfig         0
04.0011     CEX7A Accelerator deconfig         0
04.0036     CEX7A Accelerator deconfig         0
...
```

Status `deconfig` indicates that the device is not configured for the LPAR.

What to do next

For configured cryptographic devices, you can now set the device and any associated AP queues online in Linux, see “Setting devices online or offline” on page 506.

Setting devices online or offline

Use the **`chzcrypt`** command to set cryptographic devices and AP queues online or offline.

Before you begin

A cryptographic device must be configured for the LPAR before you can set the device or any associated AP queues online in Linux, see “Setting the LPAR configuration status” on page 505. For information about the dependencies between the LPAR configuration status and the online status of an AP queue, see “AP queue status overview” on page 493.

Procedure

- Preferably, use the **`chzcrypt`** command with the **`-e`** option to set cryptographic devices and AP queues online, or use the **`-d`** option to set them offline.
  For example:

  – To set cryptographic devices (in decimal notation) 0, 1, 4, 5, and 12 online issue:

  ```
  # chzcrypt -e 0 1 4 5 12
  ```

  – To set all available cryptographic devices offline issue:

  ```
  # chzcrypt -d -a
  ```

  For more information about **`chzcrypt`**, see “chzcrypt - Modify the cryptographic configuration” on page 587.
- Alternatively, write `1` to the `online` sysfs attribute of a cryptographic device to set the device online, or write `0` to set the device offline.
  For example:

  – To set a cryptographic device with device ID 0x3e online issue:

```
# echo 1 > /sys/bus/ap/devices/card3e/online
```

– To set a cryptographic device with device ID 0x3e offline issue:

```
# echo 0 > /sys/bus/ap/devices/card3e/online
```

– To check the online status of the cryptographic device with device ID 0x3e issue:

```
# cat /sys/bus/ap/devices/card3e/online
```

The value is `1` if the device is online or `0` otherwise.

Setting the polling thread

For Linux on z/VM, enabling the polling thread can improve cryptographic performance.

About this task

Linux in LPAR mode supports interrupts that indicate the completion of cryptographic requests. See “Using AP adapter interrupts” on page 508. If AP interrupts are available, it is not possible to activate the polling thread.

Depending on the workload, enabling the polling thread can increase cryptographic performance. For Linux on z/VM, the polling thread is deactivated by default.

The cryptographic device driver can run with or without the polling thread. When it runs with the polling thread, one processor constantly polls the cryptographic cards for finished cryptographic requests while requests are being processed. The polling thread sleeps when no cryptographic requests are being processed. This mode uses the cryptographic cards as much as possible, at the cost of blocking one processor during cryptographic operations.

Without the polling thread, the cryptographic cards are polled at a much lower rate. The lower rate results in higher latency, and reduced throughput for cryptographic requests, but without a noticeable processor load.

Procedure

- Use the **chzcrypt** command to set the polling thread.

  Examples:

  – To activate the polling thread issue:

  ```
  # chzcrypt -p
  ```

  – To deactivate the polling thread issue:

  ```
  # chzcrypt -n
  ```

  For more information about **chzcrypt**, see “chzcrypt - Modify the cryptographic configuration” on page 587.
- Alternatively, you can set the polling thread through the `poll_thread` sysfs attribute.

  This read-write attribute can be found at the AP bus level.

  Examples:

  – To activate a polling thread for a device 0x3e issue:

  ```
  echo 1 > /sys/bus/ap/devices/card3e/poll_thread
  ```

  – To deactivate a polling thread for a cryptographic device with bus device-ID 0x3e issue:

```
echo 0 > /sys/bus/ap/devices/card3e/poll_thread
```

Using AP adapter interrupts

To improve cryptographic performance for Linux instances that run in LPAR mode, use AP interrupts.

About this task

Using AP interrupts instead of the polling thread frees one processor while cryptographic requests are processed.

During initialization, the zcrypt device driver checks whether AP adapter interrupts are supported by the hardware. If so, polling is disabled and the interrupt mechanism is automatically used.

To query whether AP adapter interrupts are used, read the sysfs attribute `interrupt` of the device. Another interrupt attribute at the AP bus level, `/sys/bus/ap/ap_interrupts`, indicates that the AP bus is able to handle interrupts.

Example

To read the interrupt attribute for a device 0x3e with domain 0xf, issue:

```
# cat  /sys/bus/ap/devices/card3e/3e.000f/interrupt
```

If interrupts are used, the attribute shows "Interrupts enabled", otherwise "Interrupts disabled".

Setting the polling interval

Request polling is supported at nanosecond intervals.

Procedure

- Use the **`lszcrypt`** and **`chzcrypt`** commands to read and set the polling time.

  Examples:

  – To find out the current polling time, issue:

  ```
  # lszcrypt -b
  ...
  poll_timeout=250000 (nanoseconds)
  ```

  – To set the polling time to 1 microsecond, issue:

  ```
  # chzcrypt -t 1000
  ```

  For more information about **`lszcrypt`** and **`chzcrypt`**, see “lszcrypt - Display cryptographic devices” on page 682 and “chzcrypt - Modify the cryptographic configuration” on page 587.
- Alternatively, you can set the polling time through the `poll_timeout` sysfs attribute. This read-write attribute can be found at the AP bus level.

  Examples:

  – To read the poll_timeout attribute for the ap bus issue:

  ```
  # cat  /sys/bus/ap/poll_timeout
  ```

  – To set the poll_timeout attribute for the ap bus to poll, for example, every microsecond, issue:

  ```
  # echo 1000 > /sys/bus/ap/poll_timeout
  ```

Dynamically adding and removing cryptographic adapters

You can add or remove cryptographic adapters from a running Linux instance.

Before you begin

For z/VM or KVM guests, your hypervisor version must support dynamic adding and removing of cryptographic adapters.

About this task

Linux attempts to detect new cryptographic adapters and set them online every time a configuration timer expires. Read or modify the expiration time with the **lszcrypt** and **chzcrypt** commands.

For more information about **lszcrypt** and **chzcrypt**, see “lszcrypt - Display cryptographic devices” on page 682 and “chzcrypt - Modify the cryptographic configuration” on page 587.

Adding or removing of cryptographic adapters to or from an LPAR is transparent to applications that use clear key functions. If a cryptographic adapter is removed while cryptographic requests are being processed, the device driver automatically resubmits lost requests to the remaining adapters. Special handling is required for secure key.

Secure key requests are submitted to a dedicated cryptographic coprocessor. If this coprocessor is removed or lost, new requests cannot be submitted to a different coprocessor. Therefore, dynamically adding and removing adapters with a secure key application requires support within the application. For more information about secure key cryptography, see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294. You can obtain this publication at

```
ibm.com/docs/en/linux-on-systems?topic=overview-secure-key-solution-cca-application-programmers-guide
```

Alternatively, you can read or set the configuration timer through the `config_time` sysfs attribute. This read-write attribute can be found at the AP bus level. Valid values for the `config_time` sysfs attribute are in the range 5 - 120 seconds.

For the secure key cryptographic functions on EP11 coprocessors, see *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713. You can obtain it at ibm.com/docs/en/linux-on-systems?topic=overview-exploiting-enterprise-pkcs-11-using-opencryptoki

Procedure

You can work with cryptographic adapters in the following ways:

- Add or remove cryptographic adapters by using the SE or HMC.

  After the configuration timer expires, the cryptographic adapter is added to or removed from Linux, and the corresponding sysfs entries are created or deleted.
- Enable or disable a cryptographic adapter by using the **chzcrypt** command.

  The cryptographic adapter is only set online or offline in sysfs. The sysfs entries for the cryptographic adapter are retained. Use the **lszcrypt** command to check the results of the **chzcrypt** command.

Examples

- To use the **lszcrypt** and **chzcrypt** commands to find out the current configuration timer setting, issue:

```
# lszcrypt -b
...
config_time=30 (seconds)
...
```

  In the example, the timer is set to 30 seconds.
- To set the configuration timer to 60 seconds, issue:

```
# chzcrypt -c 60
```

To use sysfs to find out the current configuration timer setting, issue:

- To read the configuration timer setting, issue:

```
# cat  /sys/bus/ap/config_time
```

- To set the configuration timer to 60 seconds, issue:

```
# echo 60 > /sys/bus/ap/config_time
```

Freeing AP queues for KVM guests

By default, the zcrypt device driver controls all AP queues on a Linux instance. Free AP queues from zcrypt to make them available to alternative device drivers, such as vfio_ap. The vfio_ap device driver controls AP queues on behalf of KVM guests. While vfio_ap is the only eligible alternative device driver, freeing an AP queue from zcrypt makes it available to vfio_ap.

Before you begin

Free AP queues only if your Linux instance is a KVM host that needs to provide these AP queues to its KVM guests.

Attention:

Do not change the settings for adapters or domains that are in use or reserved for another exploiter. In particular, do not mark adapters or queues for the zcrypt device driver while they are assigned to a KVM guest.

About this task

Two masks rule which AP queues are controlled by the zcrypt device driver and which are available to alternative device drivers, such as vfio_ap.

Adapter mask
: The adapter mask is a 256-bit value, each bit representing a cryptographic adapter. The leftmost bit represents the adapter with ID `0x00`. In sysfs, the mask is available as the value of attribute `/sys/bus/ap/apmask`. If an adapter bit is set to 0, all AP queues on this adapter are available for alternative device drivers.

Domain mask
: The domain mask is a 256-bit value, each bit representing a cryptographic domain. The leftmost bit represents the domain with ID `0x0000`. In sysfs, the mask is available as the value of attribute `/sys/bus/ap/aqmask`. If a domain bit is set to 0, all AP queues with this domain are available for alternative device drivers, across adapters.

The sysfs representation of both masks is a big-endian, 64-bit, hexadecimal value. For example:

- For an adapter mask `0x8000...`, the bit for adapter `0x00` is 1 and all others are 0.
- For a adapter mask `0xFF00...`, bits for adapters `0x00` to `0x07` are 1 and all others are 0.

`zcrypt` handles all AP queues for which both the adapter bit and the domain bit are set to 1. The default for both masks is 1 for all bits. Hence, the default value for both masks in sysfs is `0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff` and `zcrypt` is the default device driver for all AP queues. To free an AP queue for alternative device drivers, the corresponding adapter bit, or the corresponding domain bit, or both must be set to 0.

You can use module parameters (see “Kernel parameters” on page 495) to set the mask.

Procedure

Use the following methods to change a mask on a running Linux instance.

- Write a new mask value to the sysfs attribute.

  You can write a 1 - 64-digit hexadecimal number to the respective sysfs attribute to replace the mask. If fewer than 64 digits are specified, the number is padded with 0s on the right.

  Examples:

  – To set the bit for the adapters with ID `0x00` and `0x01` to 0 and all other bits to 1, issue the following command:

  ```
  # echo 0x3fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff > /sys/bus/ap/apmask
  ```

  – To set the bit for the domains with ID `0x0000` and `0x0001` to 1 and all other bits to 0, issue the following command:

  ```
  # echo 0xc > /sys/bus/ap/aqmask
  ```

- Set an individual bit value.

  You can set an individual bit by specifying the bit-number, counting from the left, with one of the following prefixes:

  -
  The minus sign (-) sets the bit to 0.

  +
  The plus sign (+) sets the bit to 1.

  Bit numbers can be in decimal or hexadecimal notation. Hexadecimal numbers must be prefixed with `0x`.

  Examples:

  – To set the bit for the adapter with ID `0x01` to 0, issue the following command:

  ```
  # echo -1 > /sys/bus/ap/apmask
  ```

  – To set the bit for the domain with ID `0x000a` to 1, issue the following command:

  ```
  # echo +10 > /sys/bus/ap/aqmask
  ```

  The following equivalent command uses hexadecimal notation:

  ```
  # echo +0xa > /sys/bus/ap/aqmask
  ```

  Note: Do not omit the leading plus (+) or minus (-) sign. Plain numbers that can be interpreted as hexadecimal values replace the entire mask, even if they are specified without the 0x prefix.

- Set a range of bit values.

  Using the same prefixes as for individual bits, you can specify a range of bit-numbers to set all bits in the range to the same value. Specify a range by specifying the bit numbers of the first and last bit in the range, separated by a hyphen (-).

  Examples:

  – To set the bit for the adapters with IDs `0x00`, `0x01`, `0x02`, and `0x03` to 0, issue the following command:

  ```
  # echo -0-3 > /sys/bus/ap/apmask
  ```

  – To set the bit for the domains with ID `0x0008`, `0x0009`, `0x000a`, `0x000b`, and `0x000c` to 1, issue the following command:

```
# echo +8-12 > /sys/bus/ap/aqmask
```

The following equivalent command uses hexadecimal notation:

```
# echo +0x8-0xc > /sys/bus/ap/aqmask
```

– To set all domains to 0 issue:

```
# echo -0-255 > /sys/bus/ap/aqmask
```

This command is equivalent to the following hexadecimal notation:

```
# echo -0x0-0xff > /sys/bus/ap/aqmask
```

The same results can be achieved by replacing the entire mask with 0s with the following command:

```
# echo 0x0 > /sys/bus/ap/aqmask
```

- Set multiple bits and bit ranges with a single command.

  You can set multiple bits and bit ranges with the same command, by using a comma-separated list of specifications for individual bits and ranges. The expressions are processed individually from left to right.

  Examples:

  – To set the bit for the adapters with IDs `0x00` to 0, the range from `0x01` to `0x03` to 1, and `0x0a` to 1 issue:

```
# echo -0,+1-3,+10 > /sys/bus/ap/apmask
```

The following equivalent command uses hexadecimal notation:

```
# echo -0x0,+0x1-0x3,+0xa > /sys/bus/ap/apmask
```

– This command is equivalent to the following sequence of commands:

```
# echo -0 > /sys/bus/ap/apmask
# echo +1-3 > /sys/bus/ap/apmask
# echo +10 > /sys/bus/ap/apmask
```

– To first set the bits for all domains to zero, then set the range `0x0010` to `0x001f` to 1, and then set `0x0014` to 0, issue the following command:

```
# echo -0-255,+16-31,-20 > /sys/bus/ap/aqmask
```

The following equivalent command uses hexadecimal notation:

```
# echo -0x0-0xff,+0x10-1f,-0x14 > /sys/bus/ap/aqmask
```

This command is equivalent to the following sequence of commands:

```
# echo -0-255 > /sys/bus/ap/aqmask
# echo +16-31 > /sys/bus/ap/aqmask
# echo -20 > /sys/bus/ap/aqmask
```

The resulting bit mask can be read from the sysfs attribute:

```
# cat /sys/bus/ap/aqmask
0x0000f7ff00000000000000000000000000000000000000000000000000000000
```

Results

You can check your results by reading the masks from the sysfs attributes. In the following example, adapter `0x01`, and domains `0x0001` and `0x0002`, across all adapters, are ignored by the `zcrypt` device driver and thus, free for alternative device drivers.

```
# cat /sys/bus/ap/apmask
0xbfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
# cat /sys/bus/ap/aqmask
0x9fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

External programming interfaces

Applications can directly access the cryptographic device driver through an API.

Programmers: This information is intended for those who want to program against the cryptographic device driver or against the available cryptographic libraries.

For information about the library APIs, see the following files:

- The libica library `/usr/include/ica_api.h`
- The openCryptoki library `/usr/include/opencryptoki/pkcs11.h`
- The CCA library `/opt/IBM/CCA/include/csulincl.h`
- The EP11 library `/usr/include/ep11-host-devel/ep11.h` and `ep11adm.h`

`ep11.h` and `ep11adm.h` are included in the EP11 devel package. `pkcs11.h` is included in the opencryptoki devel package. `ica_api.h` is provided by the libica-devel package. `csulincl.h` is present after the csulcca package is installed. Install the libica-devel-*<version>* RPM. You can obtain the openCryptoki package from sourceforge at: sourceforge.net/projects/opencryptoki

Clear key cryptographic functions

The libica library provides a C API to clear-key cryptographic functions that are supported by IBM Z hardware. You can configure both openCryptoki (by using the icatoken) and openssl (by using the ibmca engine) to use IBM Z clear-key cryptographic hardware support through libica. See *libica Programmer's Reference*, SC34-2602 for details about the libica functions.

If you must circumvent libica and access the cryptographic device driver directly, your user space program must open the `zcrypt` device node, and submit the cryptographic request with an IOCTL. The IOCTL subfunction ICARSAMODEXPO performs RSA modular exponent encryption and decryption. The IOCTL ICARSACRT performs RSA CRT decryption. See the cryptographic device driver header file in the Linux source tree:
`/usr/include/asm/zcrypt.h`

Secure key cryptographic functions

To use secure key cryptographic functions in your user space program, see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294. You can obtain this publication at ibm.com/docs/en/linux-on-systems?topic=overview-secure-key-solution-cca-application-programmers-guide.

To use secure key cryptographic functions in your user space program by accessing an EP11 coprocessor adapter, see *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713. You can obtain it at ibm.com/docs/en/linux-on-systems?topic=security-cryptographic-hardware-support

Reading true random numbers

To read true random numbers, a user space program must open the `hwrng` device and read as many bytes as needed from the device.

Tip: Using the output of the `hwrng` device to periodically reseed a pseudo-random number generator might be an efficient use of the random numbers.

AP bus and zcrypt uevents

The AP bus and the zcrypt device driver generate uevents.

Application programmers: This information is intended for programmers or system administrators who want to act on changes related to the cryptographic adapters. For example, this information is helpful when writing a udev rule to unlock an encrypted device with a key that depends on a specific cryptographic resource.

Table 67 on page 514 summarizes the uevents that are generated by the AP bus, and Table 68 on page 517 those that are generated by the zcrypt device driver. The uevents include one or more zcrypt-specific properties, see “Properties” on page 517.

Table 67. AP bus uevents

Uevent	Example

An ADD uevent is generated for each cryptographic adapter when the AP card device struct is registered at the Linux device model.	`KERNEL[133.856730] add /devices/ap/card00 (ap)`
	`ACTION=add`
	`DEVPATH=/devices/ap/card00`
	`SUBSYSTEM=ap`
	`DEVTYPE=ap_card`
	`DEV_TYPE=000C`
	`MODALIAS=ap:t0C`
	`MODE=accel`
	`SEQNUM=14752`

An ADD uevent is generated for each AP queue when the AP queue device struct is registered at the Linux device model.	`KERNEL[133.856916] add /devices/ap/card00/00.0011 (ap)`
	`ACTION=add`
	`DEVPATH=/devices/ap/card00/00.0011`
	`SUBSYSTEM=ap`
	`DEVTYPE=ap_queue`
	`MODE=accel`
	`SEQNUM=14753`

A BIND uevent is generated for each cryptographic adapter when the AP card device is bound to a device driver.	`KERNEL[133.889983] bind /devices/ap/card00 (ap)`
	`ACTION=bind`
	`DEVPATH=/devices/ap/card00`
	`SUBSYSTEM=ap`
	`DEVTYPE=ap_card`
	`DRIVER=cex4card`
	`DEV_TYPE=000C`
	`MODALIAS=ap:t0C`
	`MODE=accel`
	`SEQNUM=14801`

A BIND uevent is generated for each AP queue when the AP queue device is bound to a device driver.	`KERNEL[133.894060] bind /devices/ap/card00/00.0011 (ap)`
	`ACTION=bind`
	`DEVPATH=/devices/ap/card00/00.0011`
	`SUBSYSTEM=ap`
	`DEVTYPE=ap_queue`
	`DRIVER=cex4queue`
	`MODE=accel`
	`SEQNUM=14818`

Table 67. AP bus uevents (continued)

Uevent	Example

A CHANGE uevent is generated when the first AP bus scan is complete. This uevent indicates that all AP devices have been detected and are represented in sysfs.
The AP devices might still require bindings to the appropriate zcrypt device drivers to become usable to user space.
	`KERNEL[133.888562] change /devices/ap (ap)`
	`ACTION=change`
	`DEVPATH=/devices/ap`
	`SUBSYSTEM=ap`
	`INITSCAN=done`
	`SEQNUM=14800`

A CHANGE uevent is generated when all AP devices are bound to device drivers.
This event can recur, for example, in response to user space actions that change driver bindings through unbind or rebind. The uevent is then generated when all device driver bindings are, again, complete.
The COMPLETECOUNT property shows how often this uevent has occurred.
	`KERNEL[133.899708] change /devices/ap (ap)`
	`ACTION=change`
	`DEVPATH=/devices/ap`
	`SUBSYSTEM=ap`
	`BINDINGS=complete`
	`COMPLETECOUNT=1`
	`SEQNUM=14849`

An UNBIND uevent is generated for each cryptographic adapter when the AP card device is unbound from its device driver.	`KERNEL[1915.351494] unbind /devices/ap/card00 (ap)`
	`ACTION=unbind`
	`DEVPATH=/devices/ap/card00`
	`SUBSYSTEM=ap`
	`DEVTYPE=ap_card`
	`DEV_TYPE=000C`
	`MODE=accel`
	`SEQNUM=14930`

An UNBIND uevent is generated for each AP queue when the AP queue device is unbound from its device driver.	`KERNEL[11648.474687] unbind /devices/ap/card0a/0a.0036 (ap)`
	`ACTION=unbind`
	`DEVPATH=/devices/ap/card0a/0a.0036`
	`SUBSYSTEM=ap`
	`DEVTYPE=ap_queue`
	`MODE=ep11`
	`SEQNUM=14865`

A REMOVE uevent is generated for each cryptographic adapter when the AP card device struct is unregistered at the Linux device model.	`KERNEL[3193.308746] remove /devices/ap/card02 (ap)`
	`ACTION=remove`
	`DEVPATH=/devices/ap/card02`
	`SUBSYSTEM=ap`
	`DEVTYPE=ap_card`
	`DEV_TYPE=000C`
	`MODALIAS=ap:t0C`
	`MODE=cca`
	`SEQNUM=14936`

Table 67. AP bus uevents (continued)

Uevent	Example

A REMOVE uevent is generated for each AP queue when the AP queue device struct is unregistered at the Linux device model.	`KERNEL[3193.308372] remove /devices/ap/card02/02.0036 (ap)`
	`ACTION=remove`
	`DEVPATH=/devices/ap/card02/02.0036`
	`SUBSYSTEM=ap`
	`DEVTYPE=ap_queue`
	`DEV_TYPE=000C`
	`MODALIAS=ap:t0C`
	`MODE=ep11`
	`SEQNUM=14934`

A CHANGE uevent is generated for each cryptographic adapter when the configuration state of the adapter changes.
The configuration state can change when the adapter is switched between "Candidate" and "Candidate and Online" on the SE panel that shows the cryptographic resources of an LPAR.
The configuration state can also be changed from Linux, see “AP queue status overview” on page 493.
The property field CONFIG shows the new configuration state of the adapter (0 or 1).
	`KERNEL[89.321715] change /devices/ap/card0f (ap)`
	`ACTION=change`
	`DEVPATH=/devices/ap/card0f`
	`SUBSYSTEM=ap`
	`CONFIG=1`
	`DEVTYPE=ap_card`
	`DRIVER=cex4card`
	`DEV_TYPE=000D`
	`MODALIAS=ap:t0D`
	`MODE=cca`
	`SEQNUM=14949`

A CHANGE uevent is generated for each AP queue when the configuration state of the adapter changes. The property field CONFIG shows the new configuration state of the queue (0 or 1).	`KERNEL[89.358270] change /devices/ap/card0f/0f.0011 (ap)`
	`ACTION=change`
	`DEVPATH=/devices/ap/card0f/0f.0011`
	`SUBSYSTEM=ap`
	`CONFIG=0`
	`DEVTYPE=ap_queue`
	`DRIVER=cex4queue`
	`MODE=cca`
	`SEQNUM=14950`

Table 68. zcrypt device driver uevents

Uevent	Example

A CHANGE uevent is generated when the online state, within Linux, of a cryptographic adapter changes.
The online state can be changed through the `online` sysfs attribute of the card device.
An adapter can also be set offline in response to failures being detected within the zcrypt device driver.
The property field ONLINE shows the new online state of the adapter (0 or 1).
	`KERNEL[1463.711604] change /devices/ap/card0f (ap)`
	`ACTION=change`
	`DEVPATH=/devices/ap/card0f`
	`SUBSYSTEM=ap`
	`ONLINE=0`
	`DEVTYPE=ap_card`
	`DRIVER=cex4card`
	`DEV_TYPE=000D`
	`MODALIAS=ap:t0D`
	`MODE=cca`
	`SEQNUM=14955`

A CHANGE uevent is generated when the online state, within Linux, of an AP queue changes.
The online state can be changed through the `online` sysfs attribute of the AP queue device.
An AP queue can also be set offline in response to failures being detected within the zcrypt device driver.
The property field ONLINE shows the new online state of the AP queue (0 or 1).
	`KERNEL[1463.712387] change /devices/ap/card0f/0f.0036 (ap)`
	`ACTION=change`
	`DEVPATH=/devices/ap/card0f/0f.0036`
	`SUBSYSTEM=ap`
	`ONLINE=1`
	`DEVTYPE=ap_queue`
	`DRIVER=cex4queue`
	`MODE=cca`
	`SEQNUM=14957`

Properties

The following properties are specific to zcrypt:

COMPLETECOUNT
The bindings complete counter. For the first bindings complete uevent its value is 1. The value is then incremented with each subsequent bindings complete uevent.

CONFIG
The new configuration state, 0 or 1, for an adapter or AP queue.

DEVPATH
The path to the device representation in sysfs. This path does not include the sysfs mount point, which is usually `/sys`.

DEVTYPE
Indicates whether the uevent is for an adapter (`ap_card`) or for an AP queue (`ap_queue`).

DEV_TYPE
The device type as a 4-digit hexadecimal value.

DRIVER
The device driver module that is bound to or unbound from the AP device, for example, cex4card, cex4queue, or vfio_ap.

INITSCAN
Indication that the initial AP bus scan is complete. The value is always "done".

MODALIAS: ap:t*<xx>*
The module alias of the AP device, where *<xx>* is a two-digit hexadecimal value for the mapped device type.

MODE
The mode of operation of the adapter or AP queue:

accel
for cryptographic accelerator mode.

ep11
for EP11 coprocessor mode.

cca
for CCA coprocessor mode.

ONLINE
The online state, within Linux, of the adapter or AP queue.

SUBSYSTEM
Identifier for the AP subsystem. The value is always "ap".

Chapter 50. Pseudorandom number generator device driver

The pseudorandom number generator (PRNG) device driver provides user-space applications with pseudorandom numbers generated by the IBM Z CP Assist for Cryptographic Function (CPACF).

The PRNG device driver supports the Deterministic Random Bit Generator (DRBG) requirements that are defined in NIST Special Publication 800-90/90A. The device driver uses the SHA-512 based DRBG mechanism.

To use the SHA-512 based DRBG, the device driver requires version 5 of the Message Security Assist (MSA), which is available as of the zEC12 and zBC12 with the latest firmware level. During initialization of the prng kernel module the device driver checks for the prerequisite.

If the prerequisites for the SHA-512 based DRBG are not fulfilled, the device driver uses the Triple Data Encryption Standard (TDES) algorithm instead. In TDES mode, the PRNG device driver uses a DRBG in compliance with ANSI X9.17 based on the TDES cipher algorithm. You can force the fallback to TDES mode by using the `mode=1` module parameter.

Terminology hint: Various abbreviations are commonly used for Triple Data Encryption Standard, for example: TDES, triple DES, 3DES, and TDEA.

Setting up the pseudo-random number device driver

In Red Hat Enterprise Linux, the pseudo-random number device driver is compiled as a module. To use it, you must load the device driver module, and optionally make it available to non-root users.

Module parameters

You can load and configure the PRNG device driver if it was compiled as a separate module.

Module parameter syntax



where:

mode=
specifies the mode in which the device driver runs:

0
Default. In this mode, the device driver automatically detects the MSA extension level and feature enablement. The device driver runs in SHA512 mode if the requirements are fulfilled, otherwise it falls back to TDES mode.

1
forces the device driver to run in TDES mode. The device driver starts only if the requirements for TDES mode are fulfilled.

2
forces the device driver to run in SHA512 mode. The device driver starts only if the requirements for SHA512 mode are fulfilled. The device driver does not fall back to TDES mode.

<sizeparam>
adjusts the random-buffer block size that the device driver uses to generate new random bytes. In TDES mode, this value can be in the range 8 - 65536, for SHA512 mode, the rangespieg is 64 - 65536. The default is 256 bytes.

<reseedparam>
adjusts the reseed limit in SHA512 mode. Multiply this value with the chunksize to obtain the reseed boundary in bytes. The value can be in the range 10000 - 100000. The default is 100000. In TDES mode, the reseed limit is a constant value of 4096 bytes.

Controlling access to the device node

Red Hat Enterprise Linux by default assigns access mode 0644 to `/dev/prandom`.

To restrict access to the device node to root users, add the following udev rule. It prevents non-root users from reading random numbers from `/dev/prandom`.

```
KERNEL=="prandom", MODE="0600", OPTIONS="last_rule"
```

If access to the device is restricted to root, add the following udev rule. It automatically extends access to the device to other users.

```
KERNEL=="prandom", MODE="0644", OPTIONS="last_rule"
```

Working with the PRNG device driver

Read random numbers and control the settings of the PRNG device driver.

Tasks include:

- “Reading pseudo-random numbers” on page 520
- “Displaying PRNG information” on page 520
- “Reseeding the PRNG” on page 522
- “Setting the reseed limit” on page 521

Reading pseudo-random numbers

The pseudo-random number device is read-only. Use the read function, cat program, or dd program to obtain random numbers.

Example

In this example `bs` specifies the block size in bytes for transfer, and `count` specifies the number of records with block size. The bytes are written to the output file.

```
dd if=/dev/prandom of=<output file name> bs=<xxxx> count=<nnnn>
```

Displaying PRNG information

Read the attributes of the `prandom` device in sysfs.

About this task

The sysfs representation of a PRNG device is a directory: `/sys/devices/virtual/misc/prandom`. This sysfs directory contains a number of attributes with information about the device.

Table 69. Attributes with PRNG information

Attribute	Explanation
chunksize	The size, in bytes, of the random-data bytes buffer that is used to generate new random numbers. The value can be in the range 64 bytes - 64 KB. The default is 256 bytes. It is rounded up to the next 64-byte boundary and can be adjusted as a module parameter when you start the module.
byte_counter	The number of random bytes generated since the PRNG device driver was started. You can reset this value only by removing and reloading the kernel module, or rebooting Linux (if PRNG was compiled into the kernel). This attribute is read-only.
errorflag	SHA512 mode only: `0` if the PRNG device driver is instantiated and running well. Any other value indicates a problem. If there is an error indication other than 0: • The DRBG does not provide random data bytes to user space • The `read()` function fails • The error code errno is set to EPIPE (broken pipe) This attribute is read-only.
mode	SHA512 if the PRNG device driver runs in SHA512 mode, TDES if the PRNG device driver runs in TDES mode. This attribute is read-only.
reseed	SHA512 mode only: An integer, writable only by root. Write any integer to this attribute to trigger an immediate reseed of the PRNG. See "Reseeding the PRNG" on page 522.
reseed_limit	SHA512 mode only: An integer, writable only by root to query or set the reseed counter limit. Valid values are in the range 10000 - 100000. The default is 100000. See "Setting the reseed limit" on page 521.
strength	SHA512 mode only: A read-only integer that shows the security strength according to NIST SP800-57. Returns the integer value of 256 in SHA512 mode.

Procedure

Issue a command of this form to read an attribute:

```
# cat /sys/devices/virtual/misc/prandom/<attribute>
```

where *<attribute>* is one of the attributes of Table 69 on page 521.

Example

This example shows a `prandom` device that is running in SHA512 mode, set to reseed after 2.56 MB:

```
# cat /sys/devices/virtual/misc/prandom/chunksize
256
# cat /sys/devices/virtual/misc/prandom/mode
2
# cat /sys/devices/virtual/misc/prandom/reseed_limit
10000
```

Setting the reseed limit

The PRNG reseeds after `chunksize` × `reseed_limit` bytes are read. By default, 100000 × 256 ≈ 25.6 MB can be read before an automatic reseed occurs.

Procedure

To set the number of times a `chunksize` amount of random data can be read from the PRNG before reseeding, write the number to the `reseed_limit` attribute.
For example:

```
# echo 10000 > /sys/devices/virtual/misc/prandom/reseed_limit
```

The `reseed_limit` value must be in the range 10000 - 100000.

Reseeding the PRNG

You can force a reseed by writing to the `reseed` attribute.

Procedure

To reseed the PRNG, write an integer to its reseed attribute:

```
# echo 1 > /sys/devices/virtual/misc/prandom/reseed
```

Writing any integer value to this attribute triggers an immediate reseed of the PRNG instance.

Chapter 51. True random-number generator device driver

The true random number generator (TRNG) device driver provides user-space applications with random data generated from the IBM Z hardware CPACF true random source.

The TRNG device is designed to provide high-quality random data for sensitive operations. This random data can be the base for cryptographic key generation or for seeding a pseudorandom number generator. The device is not intended to provide mass random data, for example, for overwriting disks.

Setting up the TRNG device driver

The true random-number generator can be built into the kernel or compiled as a separate module.

The true random-number generator requires Message-Security-Assist Extension 7 (MSA 7), which is available as of the IBM z14. During initialization of the TRNG kernel module, or, if TRNG is compiled into the kernel, during kernel startup, the device driver checks for the prerequisite. If the prerequisite is not fulfilled, the device driver silently exits.

The TRNG device driver module registers itself to the CPU feature MSA. The device driver is then loaded automatically. However, you can activate the TRNG device driver manually with the command:

```
modprobe s390_trng
```

There are no kernel or module parameters for the TRNG device driver.

Device nodes for random data

The true random-number generator device driver provides two interfaces to user space applications: the device node `/dev/trng` for direct access, and the generic device node `/dev/hwrng`.

The `/dev/hwrng` node appears when the TRNG or another source of random data registers with the hwrng device driver. If both the TRNG and a CCA coprocessor are registered, the TRNG takes precedence.

As of the z14, the kernel random device driver also uses the CPACF TRNG true random source through the arch_get_random_seed_* functions. The kernel random device driver provides two device nodes, `/dev/random` and `/dev/urandom`. The arch_get_random_seed_* functions require the CPACF TRNG.

Working with the TRNG device driver

Read random numbers and retrieve the counters of the TRNG device driver.

Tasks include:

- "Reading random numbers" on page 523
- "Displaying TRNG information" on page 524

Reading random numbers

The TRNG device is read-only. Use the read function, cat program, or dd program to obtain random numbers.

Example

In this example `bs` specifies the block size in bytes for transfer, and `count` specifies the number of records with block size. The bytes are written to the output file.

```
dd if=/dev/trng of=<output file name> bs=<xxxx> count=<nnnn>
```

Displaying TRNG information

Read the `byte_counter` attribute of the TRNG device in sysfs.

About this task

The sysfs representation of a TRNG device is a directory: `/sys/devices/virtual/misc/trng`. This sysfs directory contains an attribute, `byte_counter`, with statistical data.

Procedure

Issue this command to read the `byte_counter` attribute:

```
# cat /sys/devices/virtual/misc/trng/byte_counter
```

Example

To see statistics of a TRNG device, issue:

```
# cat /sys/devices/virtual/misc/trng/byte_counter
trng: 6187
hwrng: 528
arch: 1319696
total: 1326411
```

Where:

trng
: shows the number of bytes delivered through the `/dev/trng` device node.

hwrng
: shows the bytes retrieved from the generic hw_rng device driver and contributed to `/dev/hwrng`.

arch
: shows the amount of data that is supplied by the arch random implementation and delivered to the random device driver device nodes `/dev/random` and `/dev/urandom`.

total
: shows the sum of all bytes.

Chapter 52. Protected key device driver

The protected key device driver provides functions for generating and verifying protected keys.

Protected keys are encrypted with wrapping keys that, for Linux in LPAR mode, are specific to the LPAR. For guests of z/VM or KVM, the wrapping key is specific to the guest. Both the wrapping keys and the clear key values of protected keys are invisible to the operating system. Protected keys are designed for accelerated encryption and decryption with CPACF. For more information, see the chapter about protected keys in *z/Architecture Principles of Operation*, SA22-7832.

Pervasive encryption uses protected keys for data-at-rest, see *Pervasive Encryption for Data Volumes*, SC34-2782.

Functions

The device driver provides the following functions to cryptographic applications. The following secure key functions require a Crypto Express adapter:

- Generate a secure key from random data, then generate a protected key from the secure key.

  The secure key must be available to create a new version of the protected key whenever the current protected key is invalidated.
- Generate a secure key from a clear key, then generate a protected key from the secure key.

  The clear key must be in memory when the protected key is generated. Thereafter, the clear key can be deleted.

  The secure key must be available to create a new version of the protected key whenever the current protected key is invalidated.

The following functions do not require a Crypto Express adapter:

- Generate a protected key from a clear key. The clear key must be in memory when the protected key is generated.

  The clear key must also be available to create a new protected key if the existing protected key is invalidated.
- Generate a protected AES key from random data.

  The effective clear key is never exposed in memory.

  Important: The key is volatile and cannot be recreated if lost, for example during a reboot. Use a protected key generated from random data only to protect transient data.

The device driver also provides an in-kernel interface to generate protected keys. This interface is used, for example, by the `paes_s390` module.

Prerequisites

The protected key device driver requires the message-security-assist-extension 3 facility (MSA level 3), which was introduced with z196.

The protected key device driver requires permission for the AES key import functions. To grant this permission, go to the security settings within the profile of the applicable LPAR on the HMC. In the **CPACF Key Management Operations** section, select the **Permit AES Key import functions** option. For z/VM and KVM guests, the LPAR in which the hypervisor runs requires this option.

Secure keys are encoded with a master key that is held in an AP queue. Functions that involve secure keys require an IBM Crypto Express adapter in CCA coprocessor mode with a valid master key. For Linux on z/VM, the adapter must be dedicated to the z/VM guest virtual machine.

Crypto Express adapters can provide the following types of secure keys:

CCA AES data secure key
Requires an IBM CEX4S or later adapter in CCA coprocessor mode.

CCA AES cipher secure key
Requires an IBM CEX6S or later adapter in CCA coprocessor mode.

CCA ECC secure key
Requires an IBM CEX7S or later adapter in CCA coprocessor mode.

EP11 AES secure key
Requires an IBM CEX7S adapter in EP11 coprocessor mode.

EP11 ECC secure key
Requires an IBM CEX7S adapter in EP11 coprocessor mode.

Loading the device driver module

The protected key device driver is compiled as a separate module, pkey. This module is required for working with protected keys.

Load the pkey module with the **modprobe** command. The pkey module has no module parameters.

```
# modprobe pkey
```

Generating volatile protected keys by using the pkey device driver

You can generate protected keys from random data by reading the binary sysfs pkey attributes.

About this task

You do not need a Crypto Express adapter to generate a protected key from random data.

The `/sys/devices/virtual/misc/pkey/protkey` directory contains an attribute for each available key type. Read an attribute to obtain a protected key token.

Procedure

Go to the `protkey` subdirectory. The following attributes are available:

- protkey_aes_128
- protkey_aes_192
- protkey_aes_256
- protkey_aes_128_xts
- protkey_aes_256_xts

When reading from an attribute, you receive exactly one protected-key token. That is, for non-XTS keys, you get 80 bytes. For attributes related to the XTS cipher mode, you get two concatenated protected-key tokens, that is, you get 160 bytes.

Important: Do not use protected keys that are generated from random data to encrypt persistent data.

Alternatively to sysfs, you can use the ioctl calls, see “External programming interfaces ” on page 528.

For secure key tokens, see “Generating secure keys using the pkey device driver” on page 526

Generating secure keys using the pkey device driver

The pkey device driver uses random data from an AP queue to generate secure keys.

Such keys can be used for example, for swap disks where you might want a new key to be generated at every boot. Secure keys for this and other purposes can be read from secure key sysfs attributes.

Alternatively to sysfs, you can use the ioctl calls, see “External programming interfaces ” on page 528.

Procedure

Read the sysfs attribute according to the required the type, length, and cipher mode of the key.

- For a CCA AES data secure key, read from one of the attributes in `/sys/devices/virtual/misc/pkey/ccadata`. The following attributes are available:
  - ccadata_aes_128
  - ccadata_aes_192
  - ccadata_aes_256
  - ccadata_aes_128_xts
  - ccadata_aes_256_xts
- For a CCA AES cipher secure key read from one of the attributes in `/sys/devices/virtual/misc/pkey/ccacipher`. The following attributes are available:
  - ccacipher_aes_128
  - ccacipher_aes_192
  - ccacipher_aes_256
  - ccacipher_aes_128_xts
  - ccacipher_aes_256_xts
- For an EP11 AES secure key read from one of the attributes `/sys/devices/virtual/misc/pkey/ep11`. The following attributes are available:
  - ep11_aes_128
  - ep11_aes_192
  - ep11_aes_256
  - ep11_aes_128_xts
  - ep11_aes_256_xts

Results

Attributes for non-XTS keys yield exactly one secure-key token. Attributes for XTS cipher mode yield two concatenated secure-key tokens. The length of a token also varies by key type and length as summarized in Table 70 on page 527.

Table 70. Length of the attribute values

Key type	Attribute length non-XTS (single key token)	Attribute length XTS (two key tokens)
CCA AES data secure key	64 bytes	128 bytes
CCA AES cipher secure key	136 bytes	272 bytes
EP11 AES secure key	320 bytes	640 bytes

Setting up an encrypted swap disk

You can use a volatile protected key generated by the pkey device driver to encrypt a swap disk.

About this task

Because swap disks are discarded on reboot, volatile encryption keys are an option. You can generate volatile protected keys or secure keys from random data.

Important: Use a protected key based on random data for cases where the key is not needed after a reboot. In particular, do not use such a key with:

- KVM guest migration
- z/VM live guest relocation in a single system image (SSI)
- Suspend and resume

Procedure

1. Add an entry to `/etc/crypttab`. To encrypt the swap device using a protected key, the entry must point to one of the sysfs attributes within the `protkey` directory. Use the attribute for the required key type (see "Generating volatile protected keys by using the pkey device driver" on page 526).
   For example:

```
# <name>   <device>     <password>                                              <options>
  swap     /dev/dasdx   /sys/devices/virtual/misc/pkey/protkey/protkey_aes_256_xts   swap,\
                                          cipher=paes-xts-plain64,size=1280
```

   The swap option causes an **mkswap** to be performed after the dm-crypt device is set up.

   Tip: Consider adding the `sector-size=4096` option to increase the performance of dm-crypt encrypted disks with large block sizes.
2. Add an entry to `/etc/fstab` to use the device-mapper device `swap` as swap device:
   For example:

```
<filesystem>      <dir>      <type>      <options>      <dump>       <pass>
/dev/mapper/swap  none       swap        defaults       0            0
```

3. Ensure that the pkey kernel module is loaded during system startup before `/etc/crypttab` is evaluated.

   Check that a configuration file called `s390-pkey.conf` is in the `/usr/lib/modules-load.d/` directory. The configuration file must contain:

```
pkey
```

   The `modules-load.d` directory causes the modules to be loaded early during startup, before the swap disk is initiated.

Results

During system startup, `/etc/crypttab` is evaluated, and a dm-crypt device is set up in plain mode as a swap device, using protected key AES in XTS cipher mode. The random protected AES key is read from `/sys/devices/virtual/misc/pkey/protkey/protkey_aes_256_xts`. Its size is 2x80 bytes, which is 1280 bits.

Linux now runs with a swap device that is encrypted with a protected key.

External programming interfaces

Applications can use the protected key device driver through ioctls or corresponding kernel APIs.

Programmers: This information is intended for programmers of cryptographic applications who want to use protected keys for accelerated cryptographic operations with CPACF.

Issue ioctls on the misc character device `/dev/pkey` to generate and handle protected keys. The ioctl interface, including the required defines and structure definitions, is described in `arch/s390/include/uapi/asm/pkey.h`. Each ioctl has a matching kernel API that is also described in this file.

Table 71. ioctls of the protected key device driver

Name	Structure passed	Description
PKEY_GENSECK	struct pkey_genseck	Obtain a random CCA AES data secure key from an AP queue. The secure key is encrypted with the master key of the AP queue.
PKEY_GENSECK2	struct pkey_genseck2	Obtain a random secure key from an AP queue. The secure key is encrypted with the master key of the AP queue. Available secure key types are: CCA AES data, CCA AES cipher , and EP11 AES.
PKEY_CLR2SECK	struct pkey_clr2seck	Obtain a secure key from an AP queue. The secure key is generated from a specified clear key and encrypted with the master key of the AP queue.
PKEY_CLR2SECK2	struct pkey_clr2seck2	Obtain a secure key from an AP queue. The secure key is generated from a specified clear key and encrypted with the master key of the AP queue. Available key types are: CCA AES data secure key, CCA AES cipher secure key, and EP11 AES secure key.
PKEY_SEC2PROTK	struct pkey_sec2protk	Obtain a protected CCA AES data key from an AP queue. The protected key is generated from a specified secure key.
PKEY_CLR2PROTK	struct pkey_clr2protk	Obtain a protected key. The protected key is generated from a specified clear key.
PKEY_FINDCARD	struct pkey_findcard	Find an AP queue that holds the applicable master key for a specified secure key.
PKEY_SKEY2PKEY	struct pkey_skey2pkey	Find an AP queue that holds the applicable master key for a specified CCA AES data secure key. Then use that AP queue to obtain a protected key that is generated from the secure key. This ioctl call combines PKEY_FINDCARD and PKEY_SEC2PROTK.
PKEY_GENPROTKEY	struct pkey_genprotk	Generates a volatile protected key using pkey.
PKEY_VERIFYPROTKEY	struct pkey_verifyprotk	Verifies an AES protected key.
PKEY_VERIFYKEY2	struct pkey_verifykey2	Verifies a key blob and returns information about the key. The key can be verified against one specific AP queue. If no AP queue is specified, all available queues are checked and the ID of a queue is returned for which the key is valid. The call ends with an error if the key is not valid for the specified queue or if no queue is available for which the key is valid.
PKEY_KBLOB2PKEY	struct pkey_kblob2pkey	Transforms a key blob of any type into a protected key.

Table 71. ioctls of the protected key device driver (continued)

Name	Structure passed	Description
PKEY_KBLOB2PKEY2	struct pkey_kblob2pkey2	Transforms a key blob into a protected key. The key blob can be of a protected key or of one of the following types: • CCA AES data secure key • CCA AES cipher secure key • EP11 AES secure key
PKEY_KBLOB2PROTK3	struct pkey_kblob2pkey3	Transforms a key blob into a protected key. The key blob can be of a protected key or of one of the following types: • CCA AES data secure key • CCA AES cipher secure key • CCA ECC secure key • EP11 AES secure key • EP11 ECC secure key
PKEY_APQNS4K	struct pkey_apqns4key	Finds all available AP queues for which a specified key blob is valid.
PKEY_APQNS4KT	struct pkey_apqns4keytype	Finds all available AP queues for which a specified key type is valid for a particular master key.

Chapter 53. Hardware-accelerated in-kernel cryptography

The Linux kernel implements cryptographic operations for kernel subsystems like dm-crypt and IPSec. Applications can use these operations through the kernel cryptographic API.

In-kernel cryptographic and checksum operations can be performed by platform-specific implementations instead of the generic implementations within the Linux kernel.
On IBM Z, hardware-accelerated processing is available for some of these operations.

Hardware dependencies and restrictions

The cryptographic operations that can be accelerated by hardware implementations depend on your IBM Z hardware features and mode of operating Red Hat Enterprise Linux 9.1.

The following table provides an overview of the supported cryptographic standards and the earliest mainframe with hardware-acceleration for the associated operations:

Table 72. Hardware prerequisites

Algorithms for	**Hardware-acceleration available as of**
SHA-1	z990
SHA-256	z9®
SHA-512	z10
DES and TDES	z990 for the ECB and CBC modes z196 for the CTR mode
AES	z9 for the ECB and CBC modes with 128-bit keys z10 for the ECB and CBC modes for all AES key sizes z196 for the CTR mode for all AES key sizes z196 for XTS mode for 256-bit and 512-bit keys z14 for GCM for 128-bit, 192-bit, and 256-bit keys
GHASH	z196
PAES	z196 (AES with protected key) modes ECB, CBC, CTR, and XTS
CRC32	z13 for CRC-32 (IEEE 802.3 Ethernet) and CRC-32C (Castagnoli)
ChaCha20	z13
SHA3-256 and SHA3-512	z14
Edwards-curve DSA (Ed25519, Ed448), Elliptic Curve DSA (P-256, P-384, P-521)	z15

CPACF dependencies

Hardware-acceleration for DES, TDES, AES, GHASH, PAES, and SHA requires the Central Processor Assist for Cryptographic Function (CPACF). For information about enabling CPACF, see the documentation for your IBM Z hardware.

Vector Extension Facility dependencies

Hardware-acceleration for CRC32 algorithms and for the ChaCha20 stream cipher require require the Vector Extension Facility. Read the `features` line from `/proc/cpuinfo` to find out whether this facility is available on your hardware.

Example:

```
# grep features /proc/cpuinfo
features        : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te vx sie
```

In the output line, `vx` indicates that the Vector Extension Facility is available.

FIPS restrictions of the hardware capabilities

If the kernel runs in Federal Information Processing Standard (FIPS) mode, only FIPS 140-2 approved algorithms are available. DES, for example, is not approved by FIPS 140-2.

Read `/proc/sys/crypto/fips_enabled` to find out whether your kernel runs in FIPS mode.

Example:

```
# cat /proc/sys/crypto/fips_enabled
0
```

The kernel of the example does not run in FIPS mode. For kernels that run in FIPS mode, the output of the command is 1.

You control the FIPS mode with the fips kernel parameter, see “fips - Run Linux in FIPS mode” on page 785.

For more information about FIPS, see csrc.nist.gov/publications/detail/fips/140/2/final.

Loading the support modules

Load support modules to enable hardware-acceleration for specific cryptographic operations. None of these modules have module parameters.

sha512_s390
: enables hardware-acceleration for SHA-384 and SHA-512 operations. `sha512_s390` requires the `sha_common` module.

sha3_256_s390
: enables hardware-acceleration for SHA3-224 and SHA3-256 operations. `sha3_256_s390` requires the `sha_common` module.

sha3_512_s390
: enables hardware-acceleration for SHA3-384 and SHA3-512 operations. `sha3_512_s390` requires the `sha_common` module.

chacha_s390
: enables hardware-acceleration for the ChaCha20 stream cipher (RFC 7539).

ghash_s390
: enables hardware-acceleration for Galois hashes.

aes_s390
: enables hardware-acceleration for AES encryption and decryption for the following modes of operation:

- ECB, CBC, and CTR for key lengths 128, 192, and 256 bits
- XTS for key lengths 128 and 256 bits

des_s390
enables hardware-acceleration for DES and TDES for the following modes of operation: ECB, CBC, and CTR.

crc32-vx_s390
enables hardware-acceleration for CRC-32 (IEEE 802.3 Ethernet) and CRC-32C (Castagnoli).

paes_s390
enables protected key AES encryption and decryption for the following modes of operation:

- ECB, CBC, and CTR for key lengths 128, 192, and 256 bits
- XTS for key lengths 128 and 256 bits

The `paes_s390` kernel module includes a self test for each cipher that it provides. These self tests run by default. As a prerequisite for a successful self test, at least one of the following conditions must be met:

- The PCKMO instruction is enabled in the profile of the LPAR on which the Linux instance or its hosting hypervisor runs. To enable the PCKMO instruction, select the **Permit AES Key import functions** option in the **CPACF Key Management Operations** section.
- The Linux instance can access a cryptographic adapter in CCA coprocessor mode.
- The Linux instance can access a cryptographic adapter in EP11 coprocessor mode.

The `paes_s390` module requires the pkey device driver, see Chapter 52, “Protected key device driver,” on page 525.

The module also requires a cryptographic adapter for creating and handling secure and protected keys:

- To use CCA AES data or CCA AES cipher secure keys, the module requires a cryptographic adapter in CCA coprocessor mode.
- To use EP11 secure keys, the module requires a cryptographic adapter in EP11 coprocessor mode.

The ciphers in the `paes_s390` module can work with CCA secure data keys and CCA secure cipher keys, for example, keys that are generated by the pkey device driver. XTS requires two secure keys.

Before the `paes_s390` module uses secure keys in a cipher, it transforms them into protected keys. If a protected key becomes invalid, the `paes_s390` module re-generates the protected key from the secure key.

Mainframe hardware prior to z14: To use CPACF for AES-GCM operations, you must load both the `aes_s390` and `ghash_s390` module.

Tip: Load the modules with **`modprobe`**. **`modprobe`** handles dependencies on other modules for you.

Example:

```
# modprobe sha512_s390
```

Confirming hardware support for cryptographic operations

Read `/proc/crypto` to confirm that cryptographic operations are performed with hardware support.

Procedure

Read the `driver` lines from the content of `/proc/crypto`.

Example:

```
# cat /proc/crypto | grep driver
driver       : sha512-s390
driver       : sha224-s390
driver       : sha256-s390
driver       : sha3-512-s390
driver       : sha3-256-s390
driver       : sha1-s390
driver       : chacha20-s390
driver       : ghash-s390
...
driver       : crc32c-vx
driver       : crc32be-vx
driver       : crc32-vx
```

Each line that ends in `-s390` indicates hardware-acceleration for a corresponding algorithm or mode. Lines that end in `-vx` indicate hardware-acceleration for CRC32 checksums.

Chapter 54. Instruction execution protection

The instruction execution protection feature on IBM mainframes protects against data execution, similar to the NX feature on other architectures.

Instruction execution protection prevents stack-overflow exploits and generally makes a system insensitive to buffer-overflow attacks.

Data instruction protection is available on IBM mainframe hardware with the IEP feature. For Linux as a guest of a hypervisor, the hypervisor must support and use the instruction execution protection feature.

Instruction execution protection is available to your Linux instance if the features line in `/proc/cpuinfo` includes `iep`.

Setting up instruction execution protection

By default, Linux uses the instruction execution protection feature if it is available. You can use the `noexec` kernel parameter to disable the feature in Linux.

Kernel parameter for instruction execution protection



If set to on, `noexec` enables instruction execution protection, this is the default. If set to off, `noexec` disables instruction execution protection.

Controlling stack execution protection

To prevent stack-overflow exploits, the stack of a binary or shared library must be marked as not executable.

About this task

Use the **`execstack`** command to set, clear, or query the executable stack flag of ELF binaries and shared libraries (GNU_STACK). The execstack command is available from the prelink package. For details about **`execstack`**, see the man page.

Example

- Set and query the executable stack flag.

```
# execstack -s /usr/bin/find
# execstack -q /usr/bin/find
X /usr/bin/find
```

The leading X at the beginning of the query output line indicates that the stack is executable.

Hint: You can also use the **`readelf`** command to confirm that the stack can be executed.

```
# readelf -a /usr/bin/find | grep GNU_STACK -A 1
GNU_STACK 0x0000000000000000 0x0000000000000000 0x0000000000000000
0x0000000000000000 0x0000000000000000 RWE 8
```

The RWE towards the end of the output line means read/write/execute. You can obtain the **`readelf`** command as part of the `binutils` package. For command details, see the man page.

- Clear and query the executable stack flag.

```
# execstack -c /usr/bin/find
# execstack -q /usr/bin/find
- /usr/bin/find
```

The leading - at the beginning of the query output line indicates that the stack is not executable.

Hint: You can also use the **`readelf`** command to confirm that the stack cannot be executed.

```
# readelf -a /usr/bin/find | grep GNU_STACK -A 1
GNU_STACK 0x0000000000000000 0x0000000000000000 0x0000000000000000
0x0000000000000000 0x0000000000000000 RW  8
```

The RW towards the end of the output line means read/write, but not execute.

Part 9. Performance measurement using hardware facilities

The IBM Z hardware provides performance data that can be accessed by Linux.

Gathering performance data constitutes an additional load on the Linux instance on which the application to be analyzed runs. Hardware support for data gathering can reduce the extra load and can yield more accurate data.

For the performance measurement facilities of z/VM, see “Performance monitoring for z/VM guest virtual machines” on page 409.

Other performance relevant information is provided in the context of the respective device driver or feature. For example, see “Working with DASD statistics in debugfs” on page 158 for DASD performance and “Displaying and resetting QETH performance statistics” on page 274 for qeth group devices.

Newest version

You can find the newest version of this book at
ibm.com/docs/en/linux-on-systems?topic=linuxone-distributions

Restrictions

For prerequisites and restrictions see the IBM Z architecture specific information in the Red Hat Enterprise Linux 9.1 release notes at
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

Chapter 55. Channel measurement facility

LPAR and z/VM: The channel measurement facility is supported for Linux in LPAR mode and for Linux on z/VM.

The IBM Z architecture provides a channel measurement facility to collect statistical data about I/O on the channel subsystem.

Data collection can be enabled for all CCW devices. User space applications can access this data through the sysfs.

The channel measurement facility provides the following features:

- Basic channel measurement format for concurrently collecting data on up to 4096 devices. (Specifying 4096 or more channels causes high memory consumption, and enabling data collection might not succeed.)
- Extended channel measurement format for concurrently collecting data on an unlimited number of devices.
- Data collection for all channel-attached devices, except those using QDIO (that is, except qeth and SCSI-over-Fibre channel attached devices)

Setting up the channel measurement facility

Configure the channel measurement facility by adding parameters to the kernel parameter file.

Channel measurement facility kernel parameters



Notes:

[1] If you specify both parameter=value pairs, separate them with a blank.

where:

cmf.format
defines the format, `0` for basic and `1` for extended, of the channel measurement blocks. The default, `-1`, uses the extended format.

cmf.maxchannels=*<no_channels>*
limits the number of devices for which data measurement can be enabled concurrently with the basic format. The maximum for *<no_channels>* is 4096. A warning is printed if more than 4096 channels are specified. The channel measurement facility might still work; however, specifying more than 4096 channels causes a high memory consumption.

For the extended format, there is no limit and any value you specify is ignored.

Working with the channel measurement facility

Typical tasks that you need to perform when you work with the channel measurement facility is controlling data collection and reading data.

Enabling, resetting, and switching off data collection

Control data collection through the `cmb_enable` sysfs attribute of the device.

Procedure

Use a device's cmb_enable attribute to enable, reset, or switch off data collection.

- To enable data collection, write 1 to the cmb_enable attribute. If data collection was already enabled, writing 1 to the attribute resets all collected data to zero.

  Issue a command of this form:

  ```
  # echo 1 > /sys/bus/ccw/devices/<device_bus_id>/cmb_enable
  ```

  where `/sys/bus/ccw/devices/`*`<device_bus_id>`* represents the device in sysfs.

  When data collection is enabled for a device, a subdirectory `/sys/bus/ccw/devices/`*`<device_bus_id>`*`/cmf` is created that contains several attributes. These attributes contain the collected data (see “Reading data” on page 540).
- To switch off data collection issue a command of this form:

  ```
  # echo 0 > /sys/bus/ccw/devices/<device_bus_id>/cmb_enable
  ```

  When data collection for a device is switched off, the subdirectory `/sys/bus/ccw/devices/`*`<device_bus_id>`*`/cmf` and its content are deleted.

Example

In this example, data collection for a device `/sys/bus/ccw/devices/0.0.b100` is already active and reset:

```
# cat /sys/bus/ccw/devices/0.0.b100/cmb_enable
1
# echo 1 > /sys/bus/ccw/devices/0.0.b100/cmb_enable
```

Reading data

Read the sysfs attributes with collected I/O data, for example with the **`cat`** command.

Procedure

To read one of the attributes, issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/cmf/<attribute>
```

where `/sys/bus/ccw/devices/`*`<device_bus_id>`* is the directory that represents the device, and *<attribute>* the attribute to be read. Table 73 on page 540 summarizes the available attributes.

Table 73. Attributes with collected I/O data

Attribute	Value
ssch_rsch_count	An integer that represents the ssch rsch count value.
sample_count	An integer that represents the sample count value.
avg_device_connect_time	An integer that represents the average device connect time, in nanoseconds, per sample.

Table 73. Attributes with collected I/O data (continued)

Attribute	Value
avg_function_pending_time	An integer that represents the average function pending time, in nanoseconds, per sample.
avg_device_disconnect_time	An integer that represents the average device disconnect time, in nanoseconds, per sample.
avg_control_unit_queuing_time	An integer that represents the average control unit queuing time, in nanoseconds, per sample.
avg_initial_command_response_time	An integer that represents the average initial command response time, in nanoseconds, per sample.
avg_device_active_only_time	An integer that represents the average device active only time, in nanoseconds, per sample.
avg_device_busy_time	An integer that represents the average value device busy time, in nanoseconds, per sample.
avg_utilization	A percent value that represents the fraction of time that was spent in device connect time, plus function pending time, plus device disconnect time during the measurement period.
avg_sample_interval	An integer that represents the average time, in nanoseconds, between two samples during the measurement period. Can be "-1" if no measurement data was collected.
avg_initial_command_response_time	An integer that represents the average time in nanoseconds between the first command of a channel program being sent to the device and the command being accepted. Available in extended format only.
avg_device_busy_time	An integer that represents the average time in nanoseconds of the subchannel being in the "device busy" state when initiating a start or resume function. Available in extended format only.

Example

To read the avg_device_busy_time attribute for a device `/sys/bus/ccw/devices/0.0.b100`:

```
# cat /sys/bus/ccw/devices/0.0.b100/cmf/avg_device_busy_time
21
```

Chapter 56. Using the CPU-measurement facilities

LPAR only: The CPU-measurement facilities apply to Linux in LPAR mode only.

Use the CPU-measurement counter facility and sampling facility to obtain performance data for Linux in LPAR mode.

Counter facility
The hardware counters are grouped into the following counter sets:

- Basic counter set
- Problem-state counter set
- Crypto-activity counter set
- Extended counter set
- MT-diagnostic counter set

A further common counter set, the Coprocessor group counter set, cannot be accessed from Linux on IBM Z.

Sampling facility
The sampling facility includes the following sampling modes:

- Basic-sampling mode
- Diagnostic-sampling mode

The diagnostic-sampling mode is intended for use by IBM support only.

The number and type of individual counters and the details of the sampling facility depend on your IBM Z hardware model. Use the **lscpumf** command to find out what is available for your hardware (see "lscpumf - Display information about the CPU-measurement facilities" on page 655). For details, see *IBM The CPU-Measurement Facility Extended Counters Definition for z10, z196/z114, zEC12/zBC12, z13/z13s, z14, z15 and z16*, SA23-2261.

You can use the **perf** tool on Linux to access the hardware counters and sample data of the CPU-measurement facilities. To use the **perf** tool, you need to install the perf tool package, perf-*<version>*.s390x.rpm, provided with Red Hat Enterprise Linux.

To access counter data, you can also use the **lshwc** command, see "lshwc - Extract CPU Measurement Facilities counter sets" on page 666.

If you want to write your own application for analyzing counter or sample data, you can use the `libpfm4` library. This library is available on sourceforge at perfmon2.sourceforge.net.

If you want to write your own application for analyzing counter data, you can also use an API through ioctls, see "Obtaining counter data through an API" on page 549.

Getting ready for obtaining performance data

Preparations include LPAR authorization, setting buffer limits for sample data, and exploring the available and authorized facilities.

Authorizing an LPAR for CPU-measurement counter sets

The LPAR within which the Linux instance runs must be authorized to use the CPU-measurement counter sets. Use the HMC or SE to authorize the LPAR for teh counter sets you need.

About this task

The details of the steps in this task can differ, depending on your hardware. For more information, see the *Support Element Operations Guide* for your mainframe system.

Procedure

Perform these steps on the HMC or SE to grant authorizations:

1. Navigate to the LPAR for which you want to grant authorizations.
2. Within the LPAR profile, select the **Security** page.
3. Within the counter facility options, select each counter set you want to use. The coprocessor group counter set is not supported by Linux on IBM Z.
4. Click **Save**.

What to do next

Deactivate, activate, and IPL the LPAR to make the authorization take effect. For more information, see the *Support Element Operations Guide* for your mainframe system.

When your Linux instance is available again, you can use the **lscpumf** command to confirm that the authorizations are in place (see "lscpumf - Display information about the CPU-measurement facilities" on page 655).

Setting limits for the sampling facility buffer

Use the **chcpumf** command to set the minimum and maximum buffer size for the CPU-measurement sampling facility. See "chcpumf - Set limits for the CPU measurement sampling facility buffer" on page 579.

Before you begin

For each CPU, the CPU-measurement sampling facility has a buffer for writing sample data. The required buffer size depends on the sampling function and the sampling interval that is used by the perf tool. The sampling facility starts with an initial buffer size that depends on the expected requirements, your IBM Z hardware, and the available hardware resources. During the sampling process, the sampling facility increases the buffer size if required.

The sampling facility is designed for autonomous buffer management, and you do not usually need to intervene. You might want to change the minimum or maximum buffer size, for example, for one of the following reasons:

- There are considerable resource constraints on your system that cause perf sampling to malfunction and sample data to be lost.
- As an expert user of perf and the sampling facility, you want to explore results with particular buffer settings.

Procedure

Use the **chcpumf** command to set the minimum and maximum buffer sizes.

1. Optional: Specify the **lscpumf** command with the `-i` parameter to display the current limits for the buffer size (see "lscpumf - Display information about the CPU-measurement facilities" on page 655).
2. Optional: Specify the **chcpumf** command with the `-m` parameter to set the minimum buffer size.

   Example:

   ```
   # chcpumf -m 500
   ```

   The value that you specify with `-m` is the minimum buffer size in multiples of sample-data-blocks. A sample-data-block occupies approximately 4 KB. The specified minimum value is compared with the initial buffer size that is calculated by the sampling facility. The greater value is then used as the initial size when the sampling facility is started.
3. Optional: Specify the **chcpumf** command with the `-x` parameter to set the maximum buffer size.

   Example:

```
# chcpumf -x 1000
```

The value that you specify with `-x` is the maximum buffer size in multiples of sample-data-blocks. A sample-data-block occupies approximately 4 KB. The specified maximum is the upper limit to which the sampling facility can adjust the buffer.

Tips:

- You can specify both, the minimum and the maximum buffer size with a single command.
- Use the `-V` parameter to to display the minimum and maximum buffer settings that apply as a result of the command.

Example

To change the minimum buffer size to 500 times the size of a sample-data-block and the maximum buffer size to 1000 times the size of a sample-data-block, issue:

```
# chcpumf -V -m 500 -x 1000
Sampling buffer sizes:
    Minimum:    500 sample-data-blocks
    Maximum:   1000 sample-data-blocks
```

Obtaining details about the CPU-measurement facilities

You can obtain version information for the CPU-measurement counter and sampling facility and check which counter sets are authorized on your LPAR.

Procedure

1. Issue the **`lscpumf`** command with the `-i` parameter to display detailed information and debug data about the CPU-measurement facilities.

   Example:

```
# lscpumf -i
CPU-measurement Counter Facility
--------------------------------------------------------------------------
Version: 3.7

Authorized counter sets:
    Basic counter Set
    Crypto-Activity counter Set
    Extended counter Set
    MT-diagnostic counter Set
    Problem-State counter Set

Linux perf event support: Yes (PMU: cpum_cf)

CPU-measurement Sampling Facility
--------------------------------------------------------------------------
Sampling Interval:
     Minimum:      20800 cycles (approx.    250000 Hz)
     Maximum:  170372800 cycles (approx.        30 Hz)

Authorized sampling modes:
     basic:      (sample size:  32 bytes)
     diagnostic: (sample size: 173 bytes)

Linux perf event support: Yes (PMU: cpum_sf)

Current sampling buffer settings for cpum_sf:
    Basic-sampling mode
        Minimum:     15 sample-data-blocks (  64KB)
        Maximum:   8176 sample-data-blocks (  32MB)

    Diagnostic-sampling mode (including basic-sampling)
        Minimum:     90 sample-data-blocks ( 364KB)
        Maximum:  49056 sample-data-blocks ( 192MB)
        Size factor:  6
```

2. Optional: For more detailed information, including debug information, use the magic sysrequest function with character p. This function triggers kernel messages.

   For example, trigger the messages from procfs:

```
# echo p > /proc/sysrq-trigger
```

   Note: If you call magic sysrequest functions with a method other than through the procfs, you might need to activate them first. For more information about the magic sysrequest functions, see “Using the magic sysrequest feature” on page 51.

   Find the messages by issuing the **dmesg** command and looking for output lines that include CPUM_CF or CPUM_SF.

   More information: For details about the information in the messages, see *z/Architecture The Load-Program-Parameter and the CPU-Measurement Facilities*, SA23-2260, and the perf section in *Kernel Messages*, SC34-2599.

Obtaining counter data with the perf tool

You can use the perf tool to work with the CPU-measurement counter facility for authorized LPARs.

- “Reading CPU-measurement counters for an application” on page 546
- “Collecting CPU-measurement sample data” on page 548

Reading CPU-measurement counters for an application

Use the perf tool to read CPU-measurement counters with the scope of an application.

Before you begin

You must know the hexadecimal value of the counter number. You can find the decimal values in *z/Architecture The Load-Program-Parameter and the CPU-Measurement Facilities*, SA23-2260 and in *IBM*

The CPU-Measurement Facility Extended Counters Definition for z10, z196/z114, zEC12/zBC12, z13/z13s, z14, z15 and z16, SA23-2261.

Procedure

Issue a command of this form to read a counter:

- Using symbolic names:

```
# perf stat -e cpum_cf/event=<symbolic_name>/ -- <path_to_app>
```

- Using raw events:

```
# perf stat -e <type>:<counter_number> -- <path_to_app>
```

- Using raw events without specifying the type:

```
# perf stat -e cpum_cf/event=<counter_number>/ -- <path_to_app>
```

Where:

-e cpum_cf/event=*<symbolic_name>*/
specifies a counter through a symbolic name. Symbolic names are lengthy but meaningful and the same for all mainframes models that support the counter.

-e *<type>*:*<counter_number>*
specifies a counter as a raw event. In the specification, *<counter_number>* is a decimal number.

<type> is a decimal number that the kernel assigns to the CPU-measurement facilities device driver. To find that value on a running Linux instance, read the value of `/sys/devices/cpum_cf/type`.

Interface change: As of Red Hat Enterprise Linux 8.3 the notation *<type>*:*<counter_number>* replaces the former r*<hex_counter_number>*, where *<hex_counter_number>* was the counter number in hexadecimal notation.

This specification is short but abstract, the numbers can differ between hardware models, and the value for *<type>* can change across boot cycles.

-e cpum_cf/event=*<counter_number>*/
specifies a counter as a raw event. In the specification, *<counter_number>* is a the same decimal number as in the previous format. This format avoids the type specification.

<path_to_app>
specifies the path to the application to be evaluated. The counters are incremented for all threads that belong to the specified application. If you specify `-a` instead of the double hyphen and path, system-wide counter data is read.

Tip: You can read multiple counters by specifying a comma-separated list of counters. For example, with 5 as the value for *<type>*: `-e 5:32,5:33`.

For more information about the **perf** command, see the **perf** or **perf-stat** man page.

Examples

Issue one of the following commands to read the problem-state cycle count counter (symbolic name PROBLEM_STATE_CPU_CYCLES; decimal value 32) and the problem-state instruction count counter (symbolic name PROBLEM_STATE_INSTRUCTIONS; decimal value 33) for an application `/bin/df`.

- Using symbolic names:

```
# perf stat -e cpum_cf/event=PROBLEM_STATE_CPU_CYCLES/,\
cpum_cf/event=PROBLEM_STATE_INSTRUCTIONS/ -- /bin/df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/dasda1            6967656   3360888   3229780  51% /
none                    942956        88    942868   1% /dev/shm
/dev/dasdb1            6967656   4135792   2471260  63% /root

 Performance counter stats for '/bin/df':

         1,258,624      PROBLEM_STATE_CPU_CYCLES
           341,792      PROBLEM_STATE_INSTRUCTIONS

       0.002676094 seconds time elapsed
```

- Using raw events:

```
# cat /sys/devices/cpum_cf/type
5
# perf stat -e 5:32,5:33 -- /bin/df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/dasda1            6967656   3360884   3229784  51% /
none                    942956        88    942868   1% /dev/shm
/dev/dasdb1            6967656   4135792   2471260  63% /root

 Performance counter stats for '/bin/df':

         1,233,295      5:32
           341,792      5:33

       0.002526281 seconds time elapsed
```

Collecting CPU-measurement sample data

Use the perf tool to read CPU-measurement sample data.

Procedure

Issue a command of this form to read sample data:

```
# perf record -e cpum_sf/event=SF_CYCLES_BASIC/ -- <path_to_app>
```

Where *<path_to_app>* is the path to the application for which you want to collect sample data. If you specify `-a` instead of the double hyphen and path, system-wide sample data is collected.

Instead of the symbolic name, you can also specify the raw event name `rB0000`.

Example

```
# perf record -e cpum_sf/event=SF_CYCLES_BASIC/ -- /bin/df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/dasda1            6967656   3360508   3230160  51% /
none                    942956        88    942868   1% /dev/shm
/dev/dasdb1            6967656   4132924   2474128  63% /root
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.001 MB perf.data (~29 samples) ]
```

What to do next

You can now display the sample data by issuing the following command:

```
# perf report
```

For more information about collecting and displaying sample data with the **perf** command, see the **perf-record** and the **perf-report** man pages.

Hint: You can use the **perf record -F** option to collect sample data at a high frequency or the **perf record -c** option to collect sample data for corresponding short sampling intervals. Specified values

must be supported by both the CPU-measurement sampling facility and perf. Issue **lscpumf -i** to find out the maximum and minimum values for the CPU-measurement sampling facility. If perf fails at a high sampling frequency, you might have to adjust the `kernel.perf_event_max_sample_rate` system control to override default perf limitations.

Obtaining counter data through an API

Applications can access the CPU-measurement counters through ioctls. For example, the **lshwc** tool uses these ioctls.

Application programmers: This information is intended for programmers who want to write applications that extract data from CPU-measurement counters.

Table 74. CPU-measurement counter facility ioctls

Name	Structure passed	Description
S390_HWCTR_START	s390_ctrset_start	Start counter sets on CPUs.
S390_HWCTR_READ	s390_ctrset_read	Read counter sets on CPUs.
S390_HWCTR_STOP	n/a	Stop counter sets on CPUs.

For details about the calls and structures, see the `arch/s390/include/uapi/asm/hwctrset.h` header file in the Linux kernel source.

Issue the ioctls on the misc character device `/dev/hwctr`. You must first open this device node in read/write mode.

Sample program

```
unsigned long cpumask[1];
struct s390_hwctr_start start;

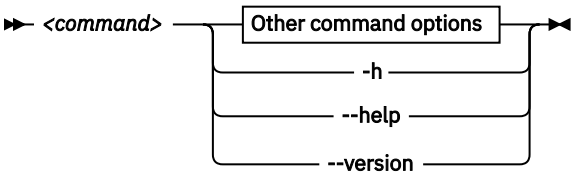
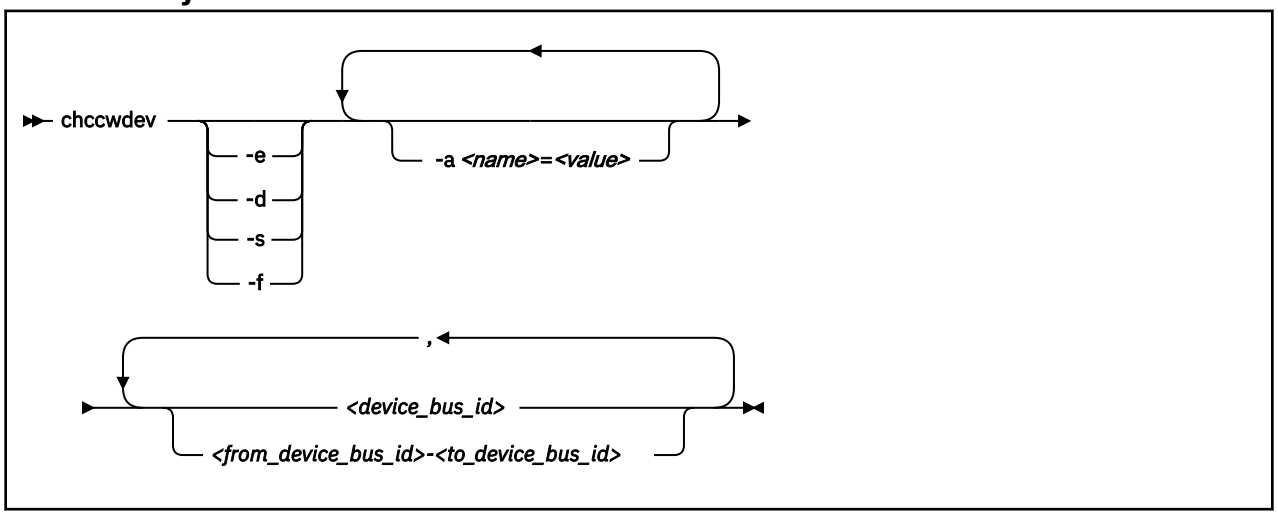
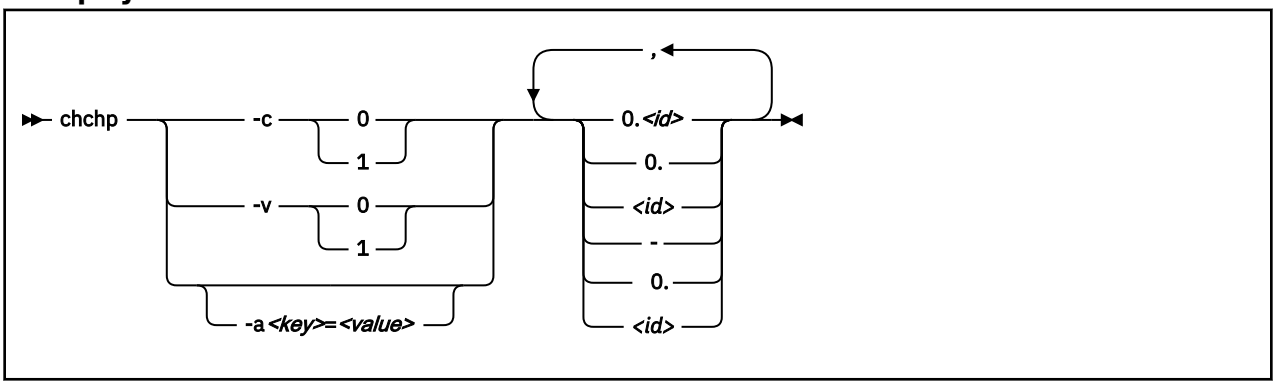
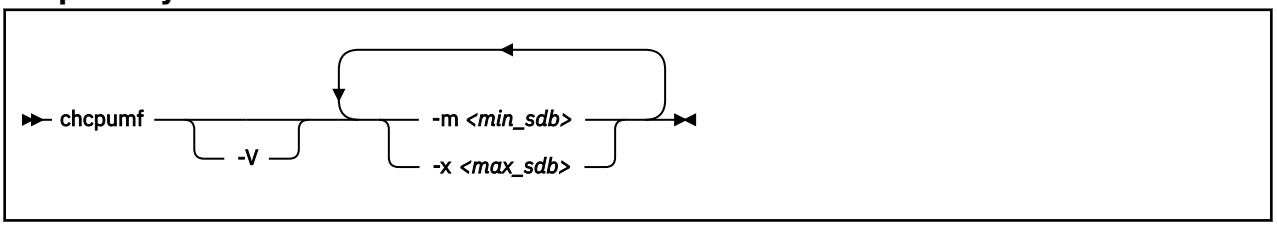
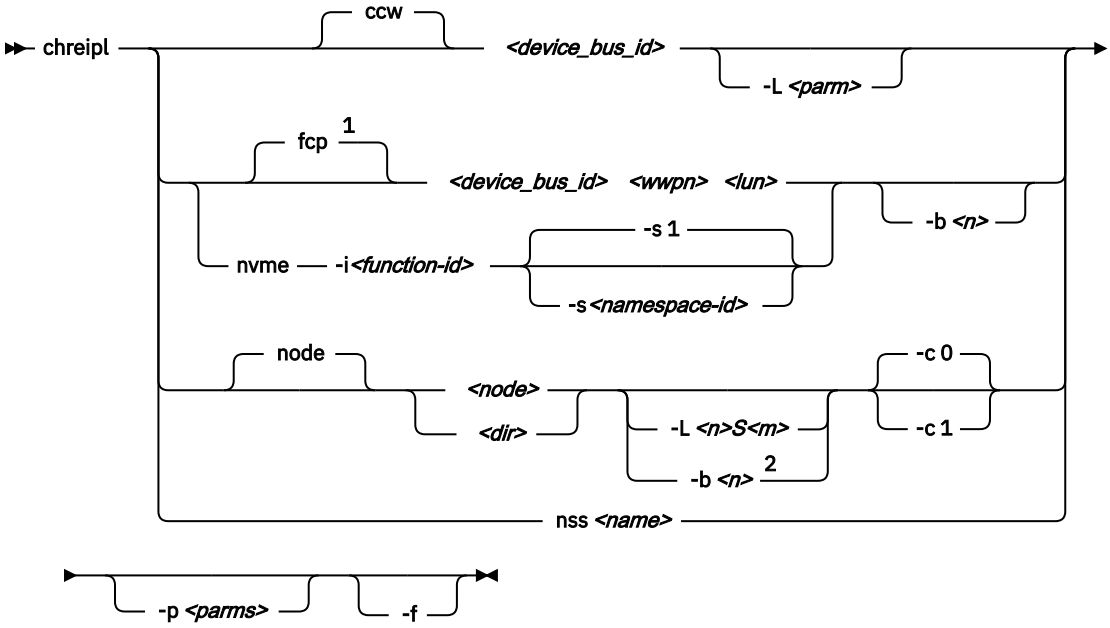
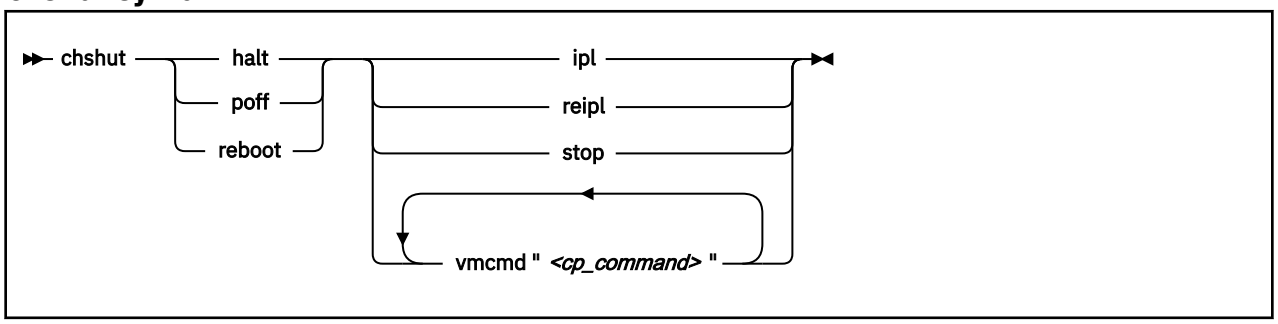
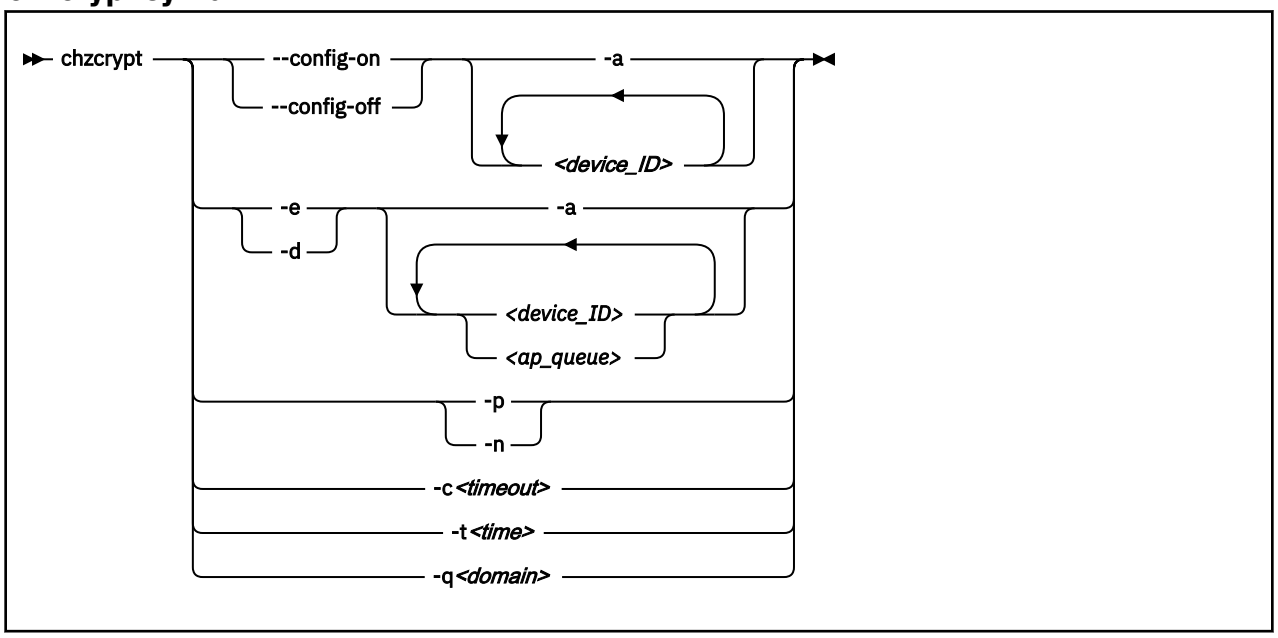
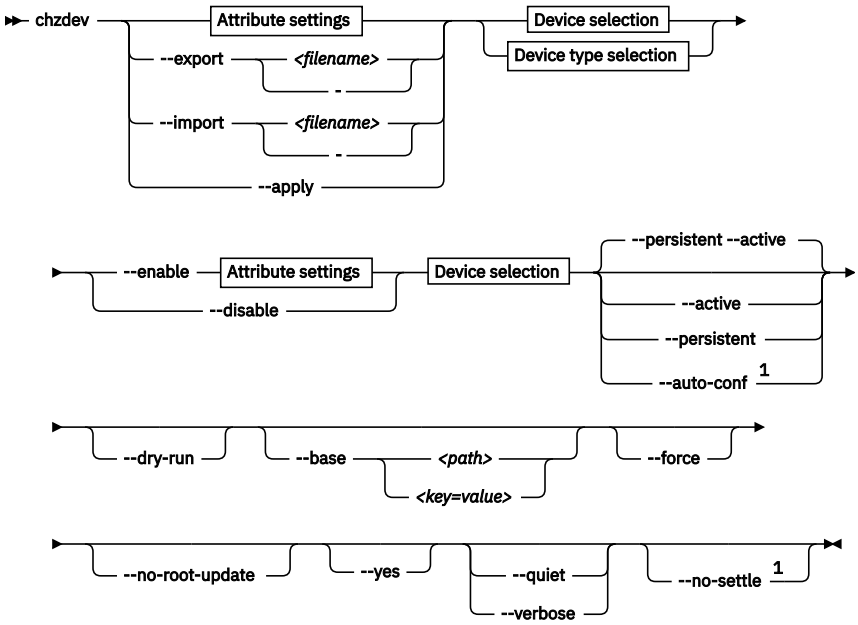
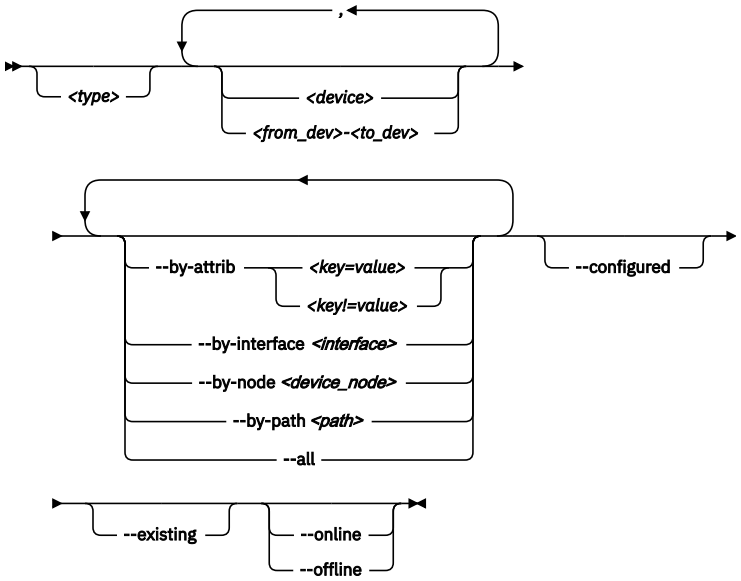
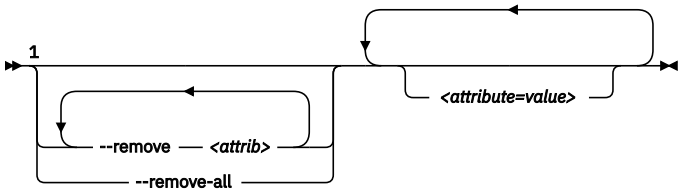
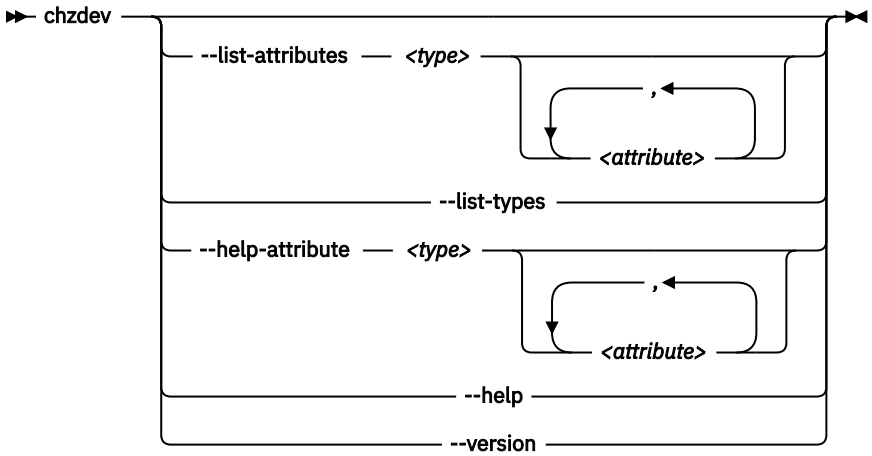
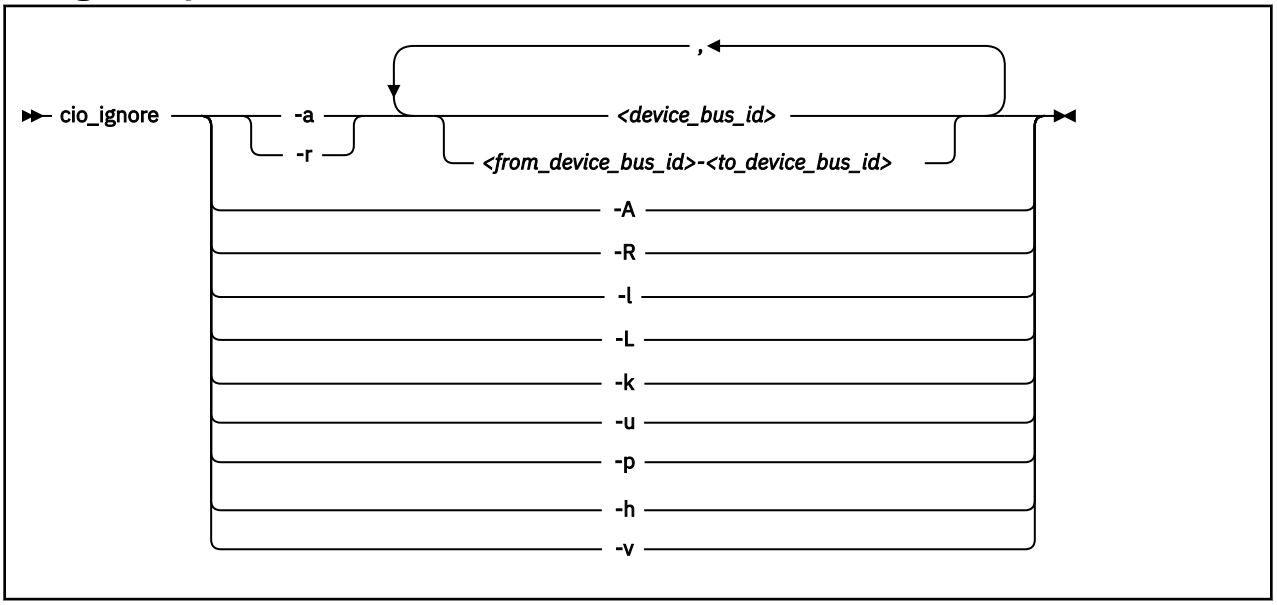
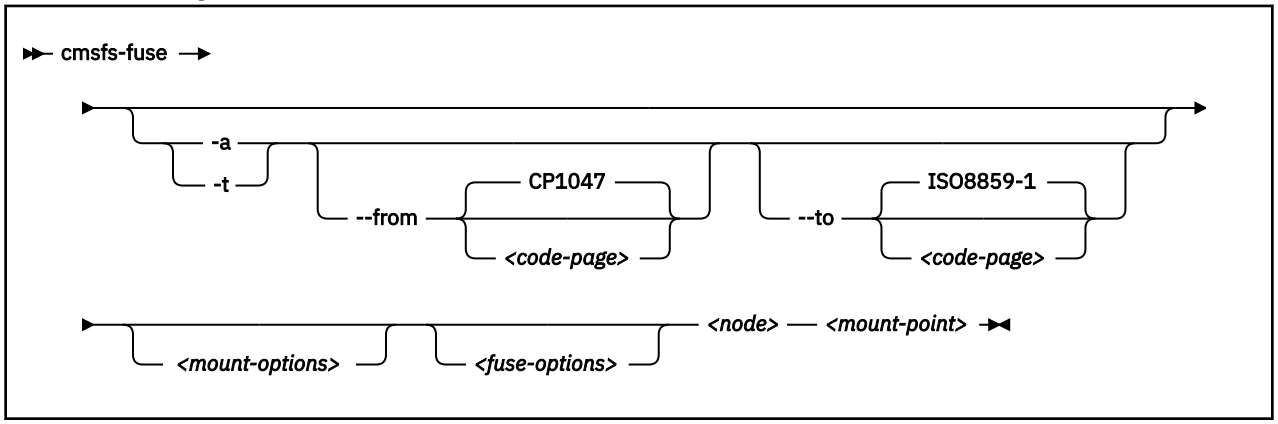
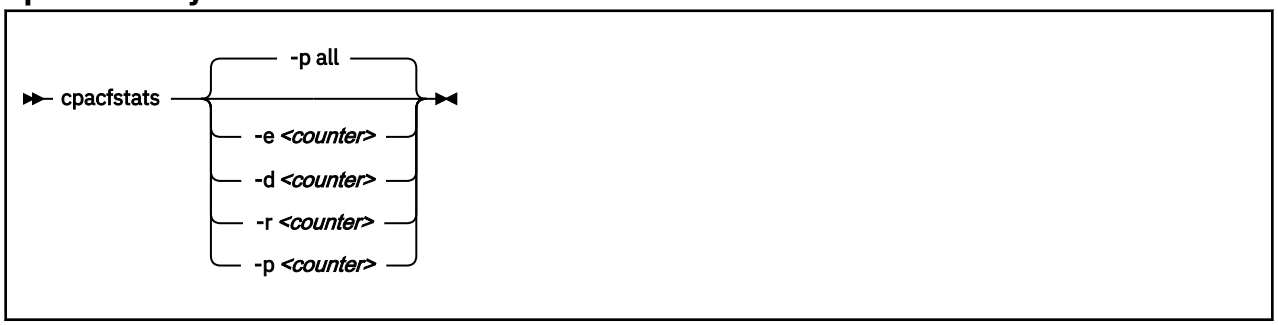
struct s390_hwctr_read read;
int fd, rc;

/* Open device */
fd = open(S390_HWCTR_DEVICE, O_RDWR);

/* Start counter sets */
cpumask[0] = 0xffff; /* Use CPUs 0-15 */
start.version = 1;
start.counter_sets = S390_HWCTR_ALL;
start.cpumask_len = sizeof(cpumask);
start.cpumask = cpumask;
rc = ioctl(fd, S390_HWCTR_START, &start);

/* Read counter sets */
read.data = malloc(start.data_bytes + sizeof(*read));
rc = ioctl(fd, S390_HWCTR_READ, &read);

/* Stop counter sets */
rc = ioctl(fd, S390_HWCTR_STOP, 0);

/* Close device */
close(fd);
```

Part 10. Diagnostics and troubleshooting

These resources are useful when diagnosing and solving problems for Red Hat Enterprise Linux 9.1.

Newest version

You can find the newest version of this book at
ibm.com/docs/en/linux-on-systems?topic=linuxone-distributions

Restrictions

For prerequisites and restrictions see the IBM Z architecture specific information in the Red Hat Enterprise Linux 9.1 release notes at
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

When reporting a problem to IBM support, you might be asked to supply a kernel dump. See *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751 for information about how to create dumps.

Chapter 57. Logging I/O subchannel status information

When investigating I/O subchannels, support specialists might request operation status information for the subchannel.

About this task

The channel subsystem offers a logging facility that creates a set of log entries with such information. You can trigger this logging facility through sysfs.

The log entries are available through the SE Console Actions Work Area with the View Console Logs function. The entries differ dependent on the device and model that is connected to the subchannel. On the SE, the entries are listed with a prefix that identifies the model. The content of the entries is intended for support specialists.

Procedure

To create a log entry, issue a command of this form:

```
# echo 1 > /sys/devices/css0/<subhannel-bus-id>/logging
```

where *<subchannel-bus-id>* is the bus ID of the I/O subchannel that corresponds to the I/O device for which you want to create a log entry.

To find out how your I/O devices map to subchannels you can use, for example, the **lscss** command.

Example

In this example, first the subchannel for an I/O device with bus ID `0.0.3d07` is identified, then logging is initiated.

```
# lscss -d 0.0.3d07
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.3d07 0.0.000c  1732/01 1731/01      80  80  ff   05000000 00000000
# echo 1 > /sys/devices/css0/0.0.000c/logging
```

Chapter 58. Control program identification

For Linux in LPAR mode, you can provide data about the Linux instance to the control program identification (CPI) feature.

The data is used, for example, to represent the Linux instance on the HMC or SE.

You provide data to the CPI feature in two steps:

1. Write values for one or more of the following items to specific sysfs attributes. Use the sysfs attributes in `/sys/firmware/cpi` for testing purposes. Use the sysfs attributes in `/etc/sysconfig/cpi` to make the changes persistent across reboots::
   - The system name
   - The sysplex name (if applicable)
   - The operating system type
   - The operating system level
2. Transfer the data to the SE, see "Sending system data to the SE" on page 557.

Red Hat Enterprise Linux 9.1 provides a systemd service unit, `cpi`, to set the attributes automatically during reboot. To automatically set the attributes during reboot, issue a command as follows:

```
# systemctl enable cpi
```

Specifying a system name

Use the `system_name` attribute in the `/etc/sysconfig/cpi` directory in sysfs to specify a system name for your Linux instance.

About this task

The system name is a string that consists of up to eight characters of the following set: A-Z, 0-9, $, @, #, and blank.

Example

```
# echo LPAR12 > /etc/sysconfig/cpi/system_name
```

To make a change to the currently running system, use:

```
# echo LPAR12 > /sys/firmware/cpi/system_name
```

What to do next

To make the setting take effect, transfer the data from `/sys/firmware/cpi/system_name` to the SE (see "Sending system data to the SE" on page 557).

Specifying a sysplex name

Use the `sysplex_name` attribute in the `/etc/sysconfig/cpi` directory in sysfs to specify a sysplex name.

About this task

The sysplex name is a string that consists of up to eight characters of the following set: A-Z, 0-9, $, @, #, and blank.

Example

```
# echo SYSPLEX1 > /etc/sysconfig/cpi/sysplex_name
```

To make a change to the currently running system, use:

```
# echo LPAR12 > /sys/firmware/cpi/sysplex_name
```

What to do next

To make the setting take effect, transfer the data from `/sys/firmware/cpi/system_name` to the SE (see "Sending system data to the SE" on page 557).

Specifying a system type

Linux uses the `/sys/firmware/cpi/system_type` sysfs attribute to identify itself as a Linux instance.

About this task

Unless your distribution sets this value for you, write LINUX to the attribute.

Example

```
# cat /sys/firmware/cpi/system_type
""
# echo LINUX > /sys/firmware/cpi/system_type
```

What to do next

To make the setting take effect, transfer the data to the SE (see "Sending system data to the SE" on page 557).

Specifying the system level

Linux uses the `/sys/firmware/cpi/system_level` sysfs attribute for the distribution and kernel version.

About this task

Red Hat sets the values for the distribution for you, and you can read the decoded values on the HMC. On machines prior to IBM z16, some values are not available to display.

The 8-byte hexadecimal system-level value has this format:

```
0x<a><b><cc><dd><eeee><ff><gg><hh>
```

where:

<a>
: is one hexadecimal byte. Its most significant bit (Bit 0) indicates hypervisor use.

<b>
: is one digit that indicates the distribution as follows:

  0
  : Generic Linux (0)

  1
  : Red Hat Enterprise Linux

  2
  : SUSE Linux Enterprise Server

3
Canonical Ubuntu

4
Fedora

5
openSUSE Leap

6
Debian GNU/Linux

7
Red Hat Enterprise Linux CoreOS

<cc>
are two digits for a distribution-specific encoding of the major version of the distribution.

<dd>
are two digits for a distribution-specific encoding of the minor version of the distribution.

<eeee>
are four digits for the patch level of the distribution.

<ff>
are two digits for the major version of the kernel.

<gg>
are two digits for the minor version of the kernel.

<hh>
are two digits for the stable version of the kernel.

Example

Red Hat Enterprise Linux 9.1 displays as

```
# cat /sys/firmware/cpi/system_level
0x0109010089050e1b
```

where no hypervisor use is indicated, the distribution is Red Hat Enterprise Linux, the distribution version is 9.1, and the kernel version is 5.14.27-137.

What to do next

To make the setting take effect, transfer the data to the SE (see “Sending system data to the SE” on page 557).

Sending system data to the SE

Use the `set` attribute in the `/sys/firmware/cpi` directory in sysfs to send data to the Support Element (SE).

About this task

To send the data in attributes `sysplex_name`, `system_level`, `system_name`, and `system_type` to the SE, write an arbitrary string to the `set` attribute.

Example

```
# echo 1 > /sys/firmware/cpi/set
```

Chapter 59. Avoiding common pitfalls

Common problems and how to avoid them.

Ensuring correct channel path status

Ensure that you have varied the path offline before you perform a planned task on it.

Before you begin

Do not vary channel paths offline that provide access to vital resources, such as the root file system.

KVM: A KVM guest might have only one channel path through which all CCW devices are accessed.

Tasks that require the channel path to be offline include:

- Pulling out or plugging in a cable on a path.
- Configuring a path off or on at the SE.

To vary the path offline, issue a command of the form:

```
# chchp -v 0 <chpid>
```

where *<chpid>* is the channel path ID.

After the operation completed and the path is available again, vary the path online by using a command of the form:

```
# chchp -v 1 <chpid>
```

Alternatively, you can write on or `off` to the channel path `status` attribute in sysfs to vary the path online or offline.

```
# echo on|off > /sys/devices/css0/chp0.<chpid>/status
```

An unplanned change in path availability can occur due to, for example, unplanned cable pulls or a temporary path malfunction. Then, the PIM/PAM/POM values (as obtained through **`lscss`**) might not be as expected. To update the PIM/PAM/POM values, vary one of the paths that lead to the affected devices.

Example:

```
# chchp -v 0 0.12
# chchp -v 1 0.12
```

Rationale: Linux does not always receive a notification (machine check) when the status of a path changes (especially for a path that comes online again). To make sure Linux has up-to-date information about the usable paths, path verification is triggered through the Linux vary operation.

Determining channel path usage on LPAR

To determine the usage of a specific channel path on LPAR, for example, to check whether traffic is distributed evenly over all channel paths, use the channel path measurement facility.

See “Channel path measurement” on page 14 for details.

Ignore unnecessary I/O devices

A Linux LPAR should contain only those I/O devices that it uses.

A mainframe environment often includes numerous I/O devices. Especially for Linux in LPAR mode, more of these I/O devices might be available to a particular instance of Linux on IBM Z than needed. Limit the I/O devices by:

- Adding only the needed devices to the applicable LPAR in the IOCDS.

  LPAR: For Linux in LPAR mode, also see Chapter 3, “Device auto-configuration for Linux in LPAR mode,” on page 21.

  z/VM and KVM: Making only needed devices available to guests.
- Using the cio_ignore= kernel parameter to ignore all devices that are not currently in use by this instance of Linux on IBM Z.

  If more devices are needed later, they can be dynamically removed from the list of devices to be ignored. Use the cio_ignore kernel parameter or the `/proc/cio_ignore` dynamic control to remove devices, see “cio_ignore - List devices to be ignored” on page 780 and “Changing the exclusion list” on page 781.

Rationale: Numerous unused devices can cause:

- Unnecessary high memory usage due to allocation of device structures.
- Unnecessary high load on status changes because hot-plug handling must be done for every device found.
- Prolonged boot and shutdown time.

Using cio_ignore

With cio_ignore, essential devices might be hidden.

For example, Linux might not boot because the device with the root file system is ignored.

If Linux does not boot under z/VM and does not show any message except:

```
HCPGIR450W CP entered; disabled wait PSW 00020001 80000000 00000000 00144D7A
```

Check if cio_ignore is used and verify that the console device, which is typically device number 0.0.0009, is not ignored.

Excessive z/VM guest swapping

Avoid excessive guest swapping by using the timed page pool size and the static page pool size attributes.

An instance of Linux on z/VM might be swapping and stalling. Setting the timed page pool size and the static page pool size to zero might solve the problem:

```
# echo 0 > /proc/sys/vm/cmm_timed_pages
# echo 0 > /proc/sys/vm/cmm_pages
```

If you see a temporary relief, the guest does not have enough memory. Try increasing the guest memory.

If the problem persists, z/VM might be out of memory.

If you are using cooperative memory management (CMM), unload the cooperative memory management module:

```
# modprobe -r cmm
```

See Chapter 43, “Cooperative memory management,” on page 453 for more details about CMM.

Including service levels of the hardware and the hypervisor

The service levels of the different hardware cards, the LPAR level, and the z/VM service level are valuable information for problem analysis.

If possible, include this information with any problem you report to IBM Support for Linux in LPAR mode or Linux on z/VM.

A `/proc` interface that provides a list of service levels is available. To see the service levels issue:

```
# cat /proc/service_levels
```

Example for a z/VM system with a QETH adapter and an FCP adapter:

```
# cat /proc/service_levels
VM: z/VM Version 7 Release 2.0, service level 2101 (64-bit)
qeth: 0.0.bdf0 firmware level V721
zfcp: 0.0.1916 microcode level 20600123
```

The service level information is automatically included in the DBGINFO file that IBM Support might ask you to generate when analyzing a problem for you.

Booting stops with disabled wait state

An automatic processor type check might stop the boot process with a disabled wait PSW.

On Red Hat Enterprise Linux 9.1, a processor type check is automatically run at every kernel startup. If the check determines that Red Hat Enterprise Linux 9.1 is not compatible with the hardware, it stops the boot process with a disabled wait PSW with an address of zero.

If this problem occurs, ensure that you are running Red Hat Enterprise Linux 9.1 on supported hardware. See the Red Hat Enterprise Linux 9.1 release notes at https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

Auto-configuration overrides persistent configuration

After upgrading Linux, persistent device configurations that were specified with **chzdev** are overridden with the auto-configuration.

Ensure that your device is configured with the `zdev:early=1` attribute. See “Activating a device early during the boot process” on page 595.

For zfcp devices that were configured with a **chzdev** version earlier than 2.5, you must also perform a one-time migration of udev rules. To migrate the rules, issue the following command:

```
# chzdev zfcp-lun --configured --enable --persistent --force
```

For more information about auto-configuration data, see Chapter 3, “Device auto-configuration for Linux in LPAR mode,” on page 21.

Preparing for dump-on-panic

You might want to consider setting up your system to automatically create a memory dump after a kernel panic.

Before you begin: If you set up kdump, a kernel panic or PSW restart automatically triggers a dump. For a KVM guest, kdump is the only option for an automatic dump-on-panic.

Configuring and using dump-on-panic has the following advantages:

- You have a memory dump disk that is prepared ahead of time.

- You do not have to reproduce the problem since a memory dump will be triggered automatically immediately after the failure.

See Chapter 8, “Shutdown actions,” on page 123 for details.

Multipath failover causes kernel panic

In a multipath setup where SCSI disks are attached over multiple paths, failover might trigger a kernel panic.

For zfcp-attached SCSI disks, `fast_io_fail_tmo` should always be configured to 5 seconds and `dev_loss_tmo` should be disabled with value "infinity". This can be configured in the file `/etc/multipath.conf`.

Preventing excessive kernel parameters during installation

The Red Hat Enterprise Linux installation program automatically generates one rd.zfcp boot parameter for each FC-attached SCSI disk that is a dependency required to mount the root file system that it finds. This automation can lead to the kernel command-line parameters exceeding the limit of 895 bytes.

About this task

For your Linux installation, you use kernel parameters in a kernel parameter file. For more details, see “Kernel parameters” on page 95. The kernel parameters are limited to 895 characters.

The same limit applies to the parameters that are generated for the installed system. These parameters include some parameters inherited from the parameter file used during installation. However, it also includes parameters generated during installation, such as rd.zfcp.

If your SAN is set up for extensive path redundancy to dependencies required to mount the root file system, or if the block device dependencies of your root file system require multiple volumes, for example, if you are using LVM, many rd.zfcp parameters might be generated.

The parameters for the installed system might then exceed the limit of 895 bytes. During installation, the writing of the boot record with zipl, and thus the installation, would then fail and could not be booted.

To reduce the number of discoverable paths, only specify a small number of paths and temporarily add the zfcp module parameter `zfcp.allow_lun_scan=0` to the parameter file used during installation. After installation, remove the workaround from the installed system.

Procedure

1. Ascertain the need for the workaround.

   Each fully qualified rd.zfcp parameter consists of 55 bytes including a separating space.

   For example, there might be 16 redundant paths to the dependencies of the root file system discoverable in your SAN. If so, you would need to multiply the number of paths with 55, here `16*55=880`. The resulting boot configuration might contain well over 1000 bytes and thus be over the limit of 895 bytes.

   Likewise, if your system configuration and installer choices cause the installer to generate other large parameters, plus the generated rd.zfcp parameters, the parameters generated for the installed system (see the example parameters in Step 3) might exceed 895 bytes. If you have enough space within the size limit for at least two redundant paths with rd.zfcp, the workaround can help to keep the parameters generated for the installed system small.

   If your system configuration and installer choices cause the installer to generate boot parameters less than or equal to 895 bytes, you do not need this workaround. For instance, with four or potentially eight paths for the root file system dependencies you might be below the size limit.
2. Temporarily add the `zfcp.allow_lun_scan=0` parameter. You need an extra 22 bytes for the parameter.

For the installation, specify the smallest number of paths that captures a maximum of path redundancy. To specify paths during installation, use the anaconda GUI, the kickstart parameter zfcp, or the dracut boot parameter rd.zfcp in the kernel parameters for an installation. See also "Manually configured FCP LUNs and their SCSI devices" on page 202.

The following example kernel parameters for an installation includes the workaround:

```
ro ramdisk_size=40000 cio_ignore=all,!condev
rd.znet=qeth,0.0.0600,0.0.0601,0.0.0602,layer2=1
ip=172.18.182.2::172.18.0.1:15:hostname:enc600:none
nameserver=172.18.0.1
inst.repo=http://server.company.com/install/RHEL/DVD
inst.vnc inst.vncpassword=secret inst.sshd
zfcp.allow_lun_scan=0
```

Subsequently, assume that the following four paths are specified during installation:

```
0.0.1900,0x5005076009009000,0x4078408800000000
0.0.1940,0x5005076009049000,0x4078408800000000
0.0.1980,0x5005076009045000,0x4078408800000000
0.0.19c0,0x5005076009005000,0x4078408800000000
```

3. Install Linux.
   An example kernel parameter line of an installed system could look like this:

```
root=/dev/mapper/rhel_hostname-root
crashkernel=256M
rd.zfcp=0.0.1900,0x5005076009009000,0x4078408800000000
rd.zfcp=0.0.1940,0x5005076009049000,0x4078408800000000
rd.zfcp=0.0.1980,0x5005076009045000,0x4078408800000000
rd.zfcp=0.0.19c0,0x5005076009005000,0x4078408800000000
rd.lvm.lv=rhel_hostname/root rd.lvm.lv=rhel_hostname/swap
cio_ignore=all,!condev
rd.znet=qeth,0.0.0600,0.0.0601,0.0.0602,layer2=1
```

   The size of the example is 425 bytes. Again, if there were 16 redundant paths to the dependencies of the root file system discoverable in your SAN, these kernel parameters would be 425+16*55=1305 bytes, and over the limit.

4. Remove the workaround.
   a) Configure the FCP devices persistently. On DPM, this is done automatically for you.

      Add all discoverable paths for the dependencies of the root file system to the persistent device configuration in `/etc/zfcp.conf`.

      This ensures that all FCP devices for the root file system are set online during boot, not just those defined by the reduced rd.zfcp statements in the boot configuration.

      To activate the paths added to `/etc/zfcp.conf` run this command:

```
# zfcp_cio_free && zfcpconf.sh
```

   b) Enable zfcp automatic LUN scanning for the current system configuration without having to reboot.

      Issue this command:

```
# echo 1 > /sys/module/zfcp/parameters/allow_lun_scan
```

      For more information about module parameters in general, see "Displaying information about module parameters" on page 31, and about zfcp parameters, see "Setting up the zfcp device driver" on page 182.

   c) Perform an automatic LUN scan once to add all paths defined in the SAN setup to the currently running system without a reboot.

      Issue the following command:

```
# for d in /sys/bus/ccw/drivers/zfcp/*/host*/scsi_host/host*/scan; do echo '- - -' > $d;
done
```

For details about triggering a LUN scan, see the corresponding topic in *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413.

d) Remove `zfcp.allow_lun_scan=0` from the files `/boot/loader/entries/*.conf` with the command:

```
# grubby --update-kernel=ALL --remove-args="zfcp.allow_lun_scan=0"
```

This configuration is persistent across reboots.

For information about using the grubby command to make changes to all kernel command-line parameters for all boot entries, see the chapter about "Configuring kernel command-line parameters" in the *Red Hat Enterprise Linux 8 System Design Guide*.

e) Re-create the initial ram disk.

Issue this command:

```
# dracut -f
```

f) Rewrite the boot record by issuing the command:

```
# zipl
```

Function unavailable or degraded in Linux on z/VM

For some functions, Linux on z/VM issues diagnose instructions to the z/VM hypervisor.

Which diagnose codes are available to your Linux instance depends on the z/VM version and configuration. The z/VM administrator can enable, disable, or customize diagnose instructions. For details, see *z/VM: CP Commands and Utilities Reference*, SC24-6268.

Linux on z/VM can fail, or one or more functions might be unavailable or degraded if required diagnose instructions are modified or unavailable.

Read and compare `/sys/kernel/debug/diag_stat` to obtain a list of calls and a counter for each call (see also Chapter 64, “Linux diagnose code use,” on page 801). The counts include failed calls. Thus, comparing the counts with and without running the affected workload provides an indication of the calls that the workload uses or tries to use. Assure that these calls are not restricted by the z/VM configuration.

Chapter 60. Displaying system information

You can display information about the resources, and capabilities of your Linux instance and about the hardware and hypervisor on which your Linux instance runs.

Displaying hardware and hypervisor information

You can display information about the physical and virtual hardware on which your Linux instance runs.

About this task

This procedure describes how to use `/proc/sysinfo`. Alternatively, you can use the "zname - Obtain information about the IBM Z hardware" on page 766 and "zhypinfo - obtain information about virtualization layers on IBM Z" on page 762 commands.

Procedure

Issue the following command:

```
# cat /proc/sysinfo
```

The output of the command is divided into several blocks.

- The first two blocks provide information about the mainframe hardware.
- The third block provides information about the LPAR on which the Linux instance runs, either in LPAR mode or as a guest of a hypervisor.
- Further blocks are present only if the Linux instance runs as a guest of a hypervisor. The field names in these sections have a prefix, VM<*nn*>, where <*nn*> is the hypervisor level.

  If the hypervisor runs in LPAR mode, there is only one such block, with prefix `VM00`. If the hypervisor runs as a guest of another hypervisor, there are multiple such blocks with prefixes `VM00`, `VM01`, and so on. `VM00` always describes the hypervisor that is closest to the Linux instance on which `/proc/sysinfo` was read.

You can use the information from `/proc/sysinfo`, for example, to verify that a guest relocation has taken place.

The following example shows the command output for an instance of Linux as a KVM guest.

Example:

```
# cat /proc/sysinfo
Manufacturer:         IBM
...
LPAR Number:          9
...
LPAR Name:            LP4KVM09
...
LPAR Extended Name:   Partition 9 KVM Host
LPAR UUID:            93724168-fda3-429b-8b28-a5d245dcb3ff
...
VM00 Name:            Linux in
VM00 Control Program: KVM/Linux
VM00 Adjustment:      1000
VM00 CPUs Total:      4
VM00 CPUs Configured: 4
VM00 CPUs Standby:    0
VM00 CPUs Reserved:   0
VM00 Extended Name:   Linux instance 42
VM00 UUID:            82038f2a-1344-aaf7-1a85-2a7250be2076
```

The fields with prefix LPAR include information that identifies and labels the partition:

Number
shows a number that identifies the partition within the mainframe.

Name
shows a partition name of up to 8 characters, as assigned on the HMC.

Extended Name
depending on your hardware, this field can contain an extended partition name, it can be empty, or it might be omitted.

UUID
shows the universally unique identifier (UUID) of the partition.

This field is present only if a UUID is assigned to the partition. On hardware that does not support UUIDs for partitions, this field is always omitted.

The fields with prefix VM<*nn*> show the following information:

Name
Depends on your hypervisor, z/VM or KVM:

z/VM
shows the name of the z/VM guest virtual machine according to the z/VM directory.

KVM
shows the name of the virtual server according to the domain XML on the KVM host. Long names are truncated to 8 characters. The full name is always shown in the **Extended Name** field (see "Extended Name (KVM only)" on page 566.

Control Program
shows hypervisor information.

Adjustment
does not show useful information for Linux on IBM Z.

CPUs Total
shows the number of virtual CPUs that the hypervisor provides to its guest.

CPUs Configured
shows the number of virtual CPUs that are online to Linux.

CPUs Standby
for Linux on z/VM, shows the number of virtual CPUs that are available to Linux but offline.

CPUs Reserved
for Linux on z/VM, shows the number of extra virtual CPUs that z/VM could make available to Linux. This is the difference between the maximum number of CPUs in the z/VM directory entry for the guest virtual machine and the number of CPUs that are currently available to Linux.

For Linux on KVM, this number is always 0.

Extended Name (KVM only)
shows the name of the virtual server as specified in the domain XML on the KVM host. See also "Name" on page 566.

UUID (KVM only)
shows the universally unique identifier (UUID) according to the domain XML on the KVM host. If you do not specify an identifier, libvirt generates a UUID when creating a virtual server definition.

Retrieving STHYI data

Store Hypervisor Information (STHYI) includes information about the IBM Z hardware, LPAR and, if applicable, the hypervisor host system on which your Linux instance runs.

STHYI includes, but is not limited to, the following information:

- The CPU count, by type (CP or IFL)
- Limitations for shared CPUs

- CEC and LPAR identifiers

The methods that you can use to retrieve this information differ between Linux in LPAR mode and Linux as a guest operating system of z/VM or of KVM.

Table 75. Available methods by environment

Method	Linux in LPAR mode	Linux as a z/VM or KVM guest
STHYI instruction with the GCC inline assembly For an example, see `arch/s390/kernel/sthyi.c` in the Linux source tree.	No	Yes
`qclib` or the **`zhypinfo`** command with the `-j` option. See the readme file of the qclib package and the man page of **`zhypinfo`** for details.	Yes	Yes
s390_sthyi() system call See the man page for details.	Yes	No

The return data for both the STHYI instruction and the s390_sthyi() system call matches the content of the STHYI response buffer as described in *z/VM: CP Programming Services*, SC24-6272. The `qclib` library provides an API for querying the information. See the readme file of the `qclib` package about obtaining the API description.

For more information about STHYI on a KVM guest, see *KVM Virtual Server Management*, SC34-2752.

You can find the `qclib` package and more information about `qclib` at github.com/ibm-s390-linux/qclib.

Check whether the Linux instance can be a hypervisor

An instance of Linux on IBM Z must have the SIE (Start Interpretive Execution) capability to be able to act as a hypervisor, such as a KVM host.

Procedure

1. Issue the following command to find out whether you can operate your Linux instance as a hypervisor.

```
# cat /proc/cpuinfo
vendor_id : IBM/S390
# processors : 1
bogomips per cpu: 14367.00
features : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh
highgprs sie
cache0 : level=1 type=Data scope=Private size=128K
...
```

2. Examine the features line in the command output. If the list of features includes `sie`, the Linux instance can be a hypervisor.

   The Linux instance of the example can be a hypervisor.

Chapter 61. Creating a kernel dump

When reporting a problem to IBM Support, you might be asked to supply a kernel dump. The dump tools you can use depend on your hypervisor environment.

Creating a kernel dump of Linux in LPAR mode or of a z/VM guest

The dump tools for Linux in LPAR mode and Linux as a z/VM guest include stand-alone dump tools, kdump, and **zgetdump**.

With kdump in place, a dump is triggered automatically by a kernel panic. Use the **zgetdump** command for a live-system dump.

For Linux as a z/VM guest you can also use VMDUMP.

For details about the dump tools, see *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751.

Creating a kernel dump of a KVM guest

A dump of a KVM guest can be driven by the host or by the guest.

Guest-driven dumps

You can set up kdump to create a kernel dump for an instance of Linux as a KVM guest on IBM Z. With kdump in place, a dump is triggered automatically by a kernel panic. Alternatively, the KVM virtual server can be configured to create a dump as a response to a kernel panic.

You can trigger an automated dump process from a running KVM guest by deliberately causing a kernel panic, for example through the magic sysrequest feature.

Host-driven dumps

The KVM virtual server administrator can use the **virsh dump** command to initiate a dump for a crashed or for a running KVM guest. By default, **virsh dump** pauses a running guest during the dump process, but the guest continues running when the dump is complete.

More information

For more information about kdump and using **virsh dump**, see *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751.

For information about the dump configuration of a KVM virtual server, see *KVM Virtual Server Management*, SC34-2752.

Part 11. Reference

Use these commands, kernel parameters, kernel options to configure Linux on IBM Z. Be aware of the z/VM DIAG calls required by Red Hat Enterprise Linux 9.1.

Newest version

You can find the newest version of this book at
ibm.com/docs/en/linux-on-systems?topic=linuxone-distributions

Restrictions

For prerequisites and restrictions see the IBM Z architecture specific information in the Red Hat Enterprise Linux 9.1 release notes at
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

Chapter 62. Commands for Linux on IBM Z

You can use z/Architecture specific commands to configure and work with the Red Hat Enterprise Linux 9.1 for IBM Z device drivers and features.

s390utils

Most of the commands described in this section are included in the s390utils package.

Some commands come with an initialization script or a configuration file or both. For init as the initialization process, it is assumed that such scripts are installed in `/etc/init.d/`. You can extract any missing files from the `etc` subdirectory in the s390utils package.

smc-tools

SMC-related commands are in a separate package, smc-tools, which is delivered with Red Hat Enterprise Linux and also available at github.com/ibm-s390-linux/smc-tools/tags.

qclib

Two commands serve as an interface to qclib, see “zname - Obtain information about the IBM Z hardware” on page 766 and “zhypinfo - obtain information about virtualization layers on IBM Z” on page 762.

Commands described elsewhere

- For the **zipl** command, see Chapter 6, “Initial program loader for IBM Z - zipl,” on page 57.
- For commands and tools that are related to creating and analyzing system dumps, see *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751.
- For commands related to terminal access over IUCV connections, see *How to Set up a Terminal Server Environment on z/VM*, SC34-2596.
- The **icainfo** and **icastats** commands are provided with the libica package and described in *libica Programmer's Reference*, SC34-2602.
- The **zkey** and **zkey-cryptsetup** commands are described in *Pervasive Encryption for Data Volumes*, SC34-2782.
- The **genprotimg** command is described in *Introducing IBM Secure Execution for Linux*, SC34-7721.

Generic command options

For simplicity, common command options are omitted from some of the syntax diagrams.

-h or --help
to display help information for the command.

--version
to display version information for the command.

The syntax for these options is:

Common command options



where *command* can be any of the Linux on IBM Z commands.

See Appendix B, “Understanding syntax diagrams,” on page 805 for general information about reading syntax diagrams.

chccwdev - Set CCW device attributes

Use the **chccwdev** command to set attributes for CCW devices and to set CCW devices online or offline.

Use "znetconf - List and configure network devices" on page 767 to work with CCW_GROUP devices. For more information about CCW devices and CCW group devices, see "Device categories" on page 7.

The **chccwdev** command uses cio_settle before it changes anything, to ensure that sysfs reflects the latest device status information, and includes newly available devices.

chccwdev syntax



Where:

-e or --online
sets the device online.

-d or --offline
sets the device offline.

-s or --safeoffline
waits until all outstanding I/O requests complete, and then tries to set the device offline. Valid for DASDs only.

-f or --forceonline
forces a boxed device online, if this action is supported by the device driver.

-a *<name>=<value>* or --attribute *<name>=<value>*
sets the *<name>* attribute to *<value>*.

The available attributes depend on the device type. See the chapter for your device for details about the applicable attributes and values.

Setting the `online` attribute has the same effect as using the **-e** or **-d** options.

<device_bus_id>
identifies a device. Device bus-IDs are of the form 0.*<n>*.*<devno>*, where *<n>* is a subchannel set ID and *<devno>* is a device number. Input is converted to lowercase.

<from_device_bus_id>-<to_device_bus_id>
identifies a range of devices. If not all devices in the range exist, the command is limited to the existing ones. If you specify a range with no existing devices, you get an error message.

-h or --help
displays help information for the command. To view the man page, enter **man chccwdev**.

-v or --version
displays version information for the command.

Examples

- To set a CCW device 0.0.b100 online issue:

```
# chccwdev -e 0.0.b100
```

- Alternatively, use **-a** to set a CCW device 0.0.b100 online. Issue:

```
# chccwdev -a online=1 0.0.b100
```

- To set all CCW devices in the range 0.0.b200 through 0.0.b2ff online, issue:

```
# chccwdev -e 0.0.b200-0.0.b2ff
```

- To set a CCW device 0.0.b100 and all CCW devices in the range 0.0.b200 through 0.0.b2ff offline, issue:

```
# chccwdev -d 0.0.b100,0.0.b200-0.0.b2ff
```

- To set several CCW devices in different ranges and different subchannel sets offline, issue:

```
# chccwdev -a online=0 0.0.1000-0.0.1100,0.1.7000-0.1.7010,0.0.1234,0.1.4321
```

- To set devices with bus ID 0.0.0192, and 0.0.0195 through 0.0.0198 offline after completing all outstanding I/O requests:

```
# chccwdev -s 0.0.0192,0.0.0195-0.0.0198
```

  If an outstanding I/O request is blocked, the command might wait forever. Reasons for blocked I/O requests include reserved devices that can be released or disconnected devices that can be reconnected.

  1. Try to resolve the problem that blocks the I/O request and wait for the command to complete.
  2. If you cannot resolve the problem, issue **chccwdev -d** to cancel the outstanding I/O requests. The data is lost.

- To set an ECKD DASD 0.0.b100 online and to enable extended error reporting and logging issue:

```
# chccwdev -e -a eer_enabled=1 -a erplog=1 0.0.b100
```

chchp - Change channel path status

Use the **chchp** command to set channel paths online or offline.

The actions are equivalent to performing a Configure Channel Path Off or Configure Channel Path On operation on the Hardware Management Console.

The channel path status that results from a configure operation is persistent across IPLs.

Note: Changing the configuration state of an I/O channel path might affect the availability of I/O devices. It can also trigger associated functions (such as channel-path verification or device scanning) which in turn can result in a temporary increase in processor, memory, and I/O load.

chchp syntax



Where:

-c *<value>* or --configure *<value>*
sets the device to configured (1) or standby (0).

Note: Setting the configured state to standby can stop running I/O operations.

-v *<value>* or --vary *<value>*
changes the logical channel-path state to online (1) or offline (0).

Note: Setting the logical state to offline can stop running I/O operations.

-a *<key>* = *<value>* or --attribute *<key>* = *<value>*
changes the channel-path sysfs attribute *<key>* to *<value>*. The *<key>* can be the name of any available channel-path sysfs attribute (that is, `configure` or `status`). *<value>* can take any valid value that can be written to the attribute (for example, `0` or `offline`). Using `-a` is a generic way of writing to the corresponding sysfs attribute. It is intended for cases where sysfs attributes or attribute values are available in the kernel but not in **chchp**.

0.*<id>* and 0.*<id>* - 0.*<id>*
where *<id>* is a hexadecimal, two-digit, lowercase identifier for the channel path. An operation can be performed on more than one channel path by specifying multiple identifiers as a comma-separated list, or a range, or a combination of both.

--version
displays the version number of **chchp** and exits.

-h or --help
displays a short help text, then exits.

Examples

- To set channel path 0.19 into standby state issue:

```
# chchp -a configure=0 0.19
```

- To set the channel path with the channel path ID 0.40 to the standby state, write 0 to the configure file with the **chchp** command:

```
# chchp --configure 0 0.40
Configure standby 0.40... done.
```

- To set a channel-path to the configured state, write 1 to the configure file with the **chchp** command:

```
# chchp --configure 1 0.40
Configure online 0.40... done.
```

- To set channel-paths 0.65 to 0.6f to the configured state issue:

```
# chchp -c 1 0.65-0.6f
```

- To set channel-paths 0.12, 0.7f and 0.17 to 0.20 to the logical offline state issue:

```
# chchp -v 0 0.12,0.7f,0.17-0.20
```

chcpumf - Set limits for the CPU measurement sampling facility buffer

Use the **chcpumf** command to set limits for the CPU measurement sampling facility buffer.

The sampling facility is designed for autonomous buffer management, and you do not usually need to intervene. However, you might want to change the minimum or maximum size, for example, for one of the following reasons:

- There are considerable resource constraints on your system, and the sampling facility stops because it tries to allocate more buffer space than is available.
- As an expert user of perf and the sampling facility, you want to explore results with particular buffer settings.

chcpumf syntax



where:

-m *<min_sdb>* or --min *<min_sdb>*
specifies the minimum sampling facility buffer size in sample-data-blocks. A sample-data-block occupies approximately 4 KB. The sampling facility starts with this buffer size if it exceeds the initial buffer size that is calculated by the sampling facility.

-x *<max_sdb>* or --max *<max_sdb>*
specifies the maximum sampling facility buffer size in sample-data-blocks. A sample-data-block occupies approximately 4 KB. While it is running, the sampling facility dynamically adjusts the buffer size to a suitable value, but cannot exceed this limit.

-V or --verbose
displays the buffer size settings after the changes.

-v or --version
displays the version number of **chcpumf** and exits.

-h or --help
displays out a short help text, then exits. To view the man page, enter **man chcpumf**.

Example

To change the minimum buffer size to 500 times the size of a sample-data-block and the maximum buffer size to 1000 times the size of a sample-data-block, issue:

```
# chcpumf -V -m 500 -x 1000
Sampling buffer sizes:
    Minimum:    500 sample-data-blocks
    Maximum:   1000 sample-data-blocks
```

chreipl - Modify the re-IPL configuration

Use the **chreipl** tool to modify the re-IPL configuration for Linux on IBM Z.

You can configure a particular device as the reboot device. For **zipl** boot menu configurations, you can set the boot menu entry to be used for the next reboot. You can also specify additional kernel parameters for the next reboot.

chreipl syntax



Notes:

[1] You can specify the *<device_bus_id>*, *<wwpn>*, and *<lun>* in any order if you use the corresponding command options.

[2] -L can be used if the device node or directory maps to a DASD or a virtio block device. The -b option can be used if the device node or directory maps to a SCSI disk or NVMe device.

Where:

<device_bus_id>* or -d *<device_bus_id>* or --device *<device_bus_id>
specifies the device bus-ID of a CCW re-IPL device or of the FCP device through with a SCSI re-IPL device is attached.

<wwpn>* or -w *<wwpn>* or --wwpn *<wwpn>
specifies the worldwide port name (WWPN) of a SCSI re-IPL device.

<lun>* or -l *<lun>* or --lun *<lun>
specifies the logical unit number (LUN) of a SCSI re-IPL device.

nvme
declares that the following parameters refer to an NVMe device.

-i *<function_id>* or --fid *<function_id>*
specifies the PCIe function ID of the NVMe device. The function ID is an 8-digit hexadecimal value. Specify the ID with a leading 0x. In the value itself you can omit leading zeroes.

-s *<namespace_id>* or --nsid *<namespace_id>*
specifies the name space ID of the NVMe device. Name space IDs are assigned by NVMe disk controllers to divide a physical NVMe device into multiple logical devices. The name space ID is an

8-digit hexadecimal value. Specify the ID with a leading 0x. In the value itself you can omit leading zeroes. The default is `0x00000001`, which is equivalent to `0x1`.

<node>
specifies a device node of a DASD, SCSI, NVMe, or logical device mapper re-IPL device. For more information about logical boot devices, see "Preparing a logical device as a boot device" on page 64.

<dir>
specifies a directory in the Linux file system on the re-IPL device.

-c or --clear
controls memory clearing during the re-IPL. Possible values are 1 and 0:

0
does not clear memory during the re-IPL. This is the default.

1
clears all memory during the re-IPL. For large memory sizes, memory clearing can considerably slow down the re-IPL process. Use this setting if you must clear memory, even at the expense of a prolonged re-IPL procedure.

You cannot control memory clearing for all environments and re-IPL devices. For unsupported devices and environments, this option causes the command to fail with an error message.

This setting can affect the online state of hotplug memory after the re-IPL, see "Memory state and reboot" on page 364

nss
declares that the following parameters refer to a z/VM named saved system (NSS).

<name>* or -n *<name>* or --name *<name>
specifies the name of an NSS as defined on the z/VM system.

-L *<n>S<m>* or --loadparm *<n>S<m>*

loadparm *<n>*
Applies only to menu configurations. Omit this specification if you are not addressing a menu configuration. *<n>* can be a positive integer, 0, or prompt.

Positive integer
Specifies the configuration number.

0
Specifies the default configuration.

prompt
Forces the menu to be displayed.

When the menu is displayed, you can specify additional kernel parameters (see "Example for a DASD menu configuration on z/VM" on page 111). These additional kernel parameters are appended to the parameters you might have provided in a parameter file. The combined parameter string must not exceed a length that is set when the kernel is compiled.

Depending on the menu configuration, omitting a value for *<n>* might display the menu or select the default configuration.

For more details about menu configurations, see "Menu configurations" on page 76. For examples of how a boot menu is displayed, see "Example for a DASD menu configuration (LPAR)" on page 98 and "Example for a DASD menu configuration on z/VM" on page 111.

loadparm S*<m>*
Applies only to zipl environments with site-specific sections. Omit the capital S and the *<m>* if you are not working with site-specific sections.

S*<m>*
Specifies the section for which kernel parameters from a zipl environment should be used. If the section does not exist, an empty section is created. *<m>* can be a digit in the range 0 to 9.

S
Omitting *<m>*, but specifying a capital S, defaults to the lowest number of any enumerated sections. If no section is found, the common specification is used.

SS
Uses the subchannel set ID (SSID) as the section ID, for example, if the SSID is 0, section 0 is used.

For more information about sections, see "Defining sections in the zipl environment file" on page 88.

loadparm *<n>*S*<m>*
If you work with a combination of menu configurations and site-specific sections in a zipl environment file, specify both *<n>* and *<m>* as described in the preceding sections. Separate *<n>* and *<m>* with an S.

-b *<n>* or --bootprog *<n>*
specifies the entry in the boot menu to be used for the next reboot. This parameter applies only if the re-IPL device is a SCSI disk or an NVMe device with a **`zipl`** boot menu configuration .

Omitting this parameter eliminates an existing selection in the boot configuration and the default boot configuration is used.

-p or --bootparms
specifies boot parameters for the next reboot. The boot parameters, which typically are kernel parameters, are appended to the kernel parameter line in the boot configuration. The number of characters you can specify depends on your environment and re-IPL device as shown in Table 76 on page 582.

Table 76. Maximum characters for additional kernel parameters by re-IPL device

Virtual hardware where Linux runs	DASD	SCSI or NVMe	NSS
z/VM guest virtual machine	64	3452	56
LPAR	none	3452	n/a
KVM	none	n/a	n/a

If you omit this parameter, the existing boot parameters in the next boot configuration are used without any changes.

-f or --force
With this option, you can force the re-IPL from a target device even if the target cannot be verified by the system. This is the case, for example, if the device is on the cio_ignore exclusion list (blacklist).

Note: Use this option with great care. Specifying a non-existing device causes the re-IPL to fail.

-h or --help
displays help information for the command. To view the man page, enter **`man chreipl`**.

-v or --version
displays version information.

For disk-type re-IPL devices, the command accepts but does not require an initial statement:

ccw
declares that the following parameters refer to a DASD or a virtio block re-IPL device.

fcp
declares that the following parameters refer to a SCSI re-IPL device.

node
declares that the following parameters refer to a disk re-IPL device that is identified by a device node or by a directory in the Linux file system on that device. The disk device can be a DASD or a SCSI disk.

Examples

These examples illustrate common uses for **chreipl**.

- The following commands all configure the same DASD as the re-IPL device, assuming that the device bus-ID of the DASD is 0.0.7e78, that the standard device node is /dev/dasdc, that udev creates an alternative device node /dev/disk/by-path/ccw-0.0.7e78, that /mnt/boot is located on the Linux file system in a partition of the DASD.
  - Using the bus ID:

```
# chreipl 0.0.7e78
```

  - Using the bus ID and the optional ccw statement:

```
# chreipl ccw 0.0.7e78
```

  - Using the bus ID, the optional statement and the optional **--device** keyword:

```
# chreipl ccw --device 0.0.7e78
```

  - Using the standard device node:

```
# chreipl /dev/dasdc
```

  - Using the udev-created device node:

```
# chreipl /dev/disk/by-path/ccw-0.0.7e78
```

  - Using a directory within the file system on the DASD:

```
# chreipl /mnt/boot
```

- The following commands all configure the same SCSI disk as the re-IPL device, assuming that the device bus-ID of the FCP device through which the device is attached is 0.0.1700, the WWPN of the storage server is 0x500507630300c562, and the LUN is 0x401040b300000000. Further it is assumed that the standard device node is /dev/sdb, that udev creates an alternative device node /dev/disk/by-id/scsi-36005076303ffc56200000000000010b4, and that /mnt/fcpboot is located on the Linux file system in a partition of the SCSI disk.
  - Using bus ID, WWPN, and LUN:

```
# chreipl 0.0.1700 0x500507630300c562 0x401040b300000000
```

  - Using bus ID, WWPN, and LUN with the optional fcp statement:

```
# chreipl fcp 0.0.1700 0x500507630300c562 0x401040b300000000
```

  - Using bus ID, WWPN, LUN, the optional statement, and keywords for the parameters. When you use the keywords, the parameters can be specified in any order:

```
# chreipl fcp --wwpn 0x500507630300c562 -d 0.0.1700 --lun 0x401040b300000000
```

  - Using the standard device node:

```
# chreipl /dev/sdb
```

  - Using the udev-created device node:

```
# chreipl /dev/disk/by-id/scsi-36005076303ffc56200000000000010b4
```

  - Using a directory within the file system on the SCSI disk:

```
# chreipl /mnt/fcpboot
```

- To configure a DASD with bus ID `0.0.7e78` as the re-IPL device, using the first entry of the **zipl** boot menu:

```
# chreipl 0.0.7e78 -L 1
Re-IPL type:  ccw
Device:       0.0.7e78
Loadparm:     "1"
Bootparms:    ""
clear:        0
```

- The following examples configures a DASD with bus ID `0.0.7e78` as the re-IPL device. To also display all kernel messages on the console, add `ignore_loglevel` to the existing kernel parameters in the boot configuration. To clear memory at the expense of a prolonged re-IPL procedure, set the clear mode to 1.

```
# chreipl 0.0.7e78 -p "ignore_loglevel"
Re-IPL type:  ccw
Device:       0.0.7e78
Loadparm:     ""
Bootparms:    "ignore_loglevel"
clear:        1
```

- The following examples configures an NVMe device with Function ID `0x00000013` and name space ID `0x00000001` as the re-IPL device.

```
# chreipl nvme -i 0x13 -s 1
Re-IPL type: nvme
FID: 0x00000013
NSID: 0x00000001
bootprog: 0
br_lba: 0
Loadparm: ""
Bootparms: ""
clear:       0
```

- The following example configures a DASD 0.0.1000 as the re-IPL device with site-specific kernel parameters, here for site 2.

```
# chreipl ccw 0.0.1000 -L "S2"
Re-IPL type: ccw
Device:      0.0.1000
Loadparm:    "S2"
Bootparms:   ""
clear: 0
```

chshut - Control the system shutdown actions

Use the **chshut** command to change the shutdown actions for specific shutdown triggers.

The shutdown triggers are:

- `halt`
- `poff`
- `reboot`

The shutdown triggers `restart` and `panic` are handled by the dumpconf service script, see *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751 for details.

Linux on IBM Z performs shutdown actions according to sysfs attribute settings within the `/sys/firmware` directory structure. The **chshut** command sets a shutdown action for a shutdown trigger by changing the corresponding sysfs attribute setting. For more information about the sysfs attributes and the shutdown actions, see Chapter 8, “Shutdown actions,” on page 123.

chshut syntax



Where:

halt
sets an action for the `halt` shutdown trigger.

Red Hat Enterprise Linux by default maps `halt` to `poff`. You can undo this mapping by editing the file `/etc/sysconfig/shutdown` and replacing `HALT="auto"` with `HALT="halt"`.

poff
sets an action for the `poff` shutdown trigger.

reboot
sets an action for the `reboot` shutdown trigger.

ipl
sets IPL as the action to be taken.

reipl
sets re-IPL as the action to be taken.

stop
sets "stop" as the action to be taken.

vmcmd "*<cp_command>*"
sets the action to be taken to issuing a z/VM CP command. The command must be specified in uppercase characters and enclosed in quotation marks. To issue multiple commands, repeat the `vmcmd` attribute with each command.

-h or --help
displays help information for the command. To view the man page, enter **man chshut**.

-v or --version
displays version information.

Examples

These examples illustrate common uses for **chshut**.

- To make the system start again after a power off:

```
# chshut poff ipl
```

- To log off the z/VM guest virtual machine if the Linux **poweroff** command was run successfully:

```
# chshut poff vmcmd LOGOFF
```

- To send a message to z/VM user ID OPERATOR and automatically log off the z/VM guest virtual machine if the Linux **poweroff** command is run:

```
# chshut poff vmcmd "MSG OPERATOR Going down" vmcmd "LOGOFF"
```

chzcrypt - Modify the cryptographic configuration

Use the **chzcrypt** command to configure cryptographic adapters that are managed by zcrypt and modify zcrypt's AP bus attributes.

In sysfs, AP queues are listed as children of a cryptographic device with a name of the form: `card<adapter_id>`. For example, if cryptographic devices with the adapter IDs 00 and 02 are selected, and the domain IDs 0002, 0003 and 000e have been configured, then the following cryptographic devices and AP queues are defined to Linux:

```
/sys/devices/ap/card00
/sys/devices/ap/card00/00.0002
/sys/devices/ap/card00/00.0003
/sys/devices/ap/card00/00.000e
/sys/devices/ap/card02
/sys/devices/ap/card02/02.0002
/sys/devices/ap/card02/02.0003
/sys/devices/ap/card02/02.000e
```

Setting a cryptographic device online or offline with **chzcrypt** affects its AP queues.

To display the attributes, use “lszcrypt - Display cryptographic devices” on page 682.

chzcrypt syntax



Where:

--config-on
For Linux in LPAR mode, configures cryptographic devices and all associated AP queues for the LPAR.

--config-off
For Linux in LPAR mode, sets the LPAR configuration status of cryptographic devices and all associated AP queues to "`not configured`". As a result, the devices and all associated AP queues are set offline in Linux.

-e or --enable
sets the given cryptographic devices and AP queues online in Linux.. Cryptographic devices can be set online only if they are configured at the LPAR level (see `--config-on`).

-d or --disable
sets the given cryptographic devices and AP queues offline in Linux..

-a or --all
can be combined with the `-e`, `-d`, `--enable`, `--disable`, `--config-on`, or `--config-off` option to act on all available cryptographic devices.

<device ID>
specifies a cryptographic device. A cryptographic device can be specified either in decimal notation or hexadecimal notation with a '0x' prefix.

<ap_queue>
specifies an AP queue in hexadecimal notation, omitting the '0x' prefix.

-p or --poll-thread-enable
enables zcrypt's poll thread.

-n or --poll-thread-disable
disables zcrypt's poll thread.

-c *<timeout>* or --config-time *<timeout>*
sets configuration timer for rescanning the AP bus to *<timeout>* seconds.

-t *<time>* or --poll-timeout=*<time>*
sets the high-resolution polling timer to *<time>* nanoseconds. To display the value, use **`lszcrypt -b`**.

-q *<domain>* or --default-domain *<domain>*
changes the default domain. Specify the domain as either a hexadecimal or decimal value.

Important: Be sure to enter an existing domain. The Trusted Key Entry (TKE) workstation does not find the cryptographic adapters if a non-existing domain is entered here. All CCA applications use the default domain, and do not work correctly if the specified domain does not exist.

-V or --verbose
displays verbose messages.

-h or --help
displays short information about command usage.

-v or --version
displays version information.

Examples

These examples illustrate common uses for **`chzcrypt`**.

- To configure cryptographic devices with the adapter IDs 0, 1, 4, 5, and 12 and their associated AP queues for the LPAR (in decimal notation):

```
chzcrypt --config-on 0 1 4 5 12
```

- To set the cryptographic devices with the adapter IDs 0, 1, 4, 5, and 12 and their associated AP queues online (in decimal notation):

```
chzcrypt -e 0 1 4 5 12
```

  Or, in hexadecimal notation:

```
chzcrypt -e 0x00 0x01 0x04 0x05 0x0C
```

- To set all available cryptographic adapters including all AP queues, offline:

```
chzcrypt -d -a
```

- To set the AP queue defined by adapter ID 00 and domain 77 (0x4d) offline:

```
chzcrypt -d 00.004d
```

- To set the configuration timer for rescanning the AP bus to 60 seconds and disable zcrypt's poll thread:

```
chzcrypt -c 60
chzcrypt -n
```

- To change the default domain to 77 (0x4d):

```
chzcrypt -q 0x4d
```

or

```
chzcrypt -q 77
```

chzdev - Configure IBM Z devices

Use the **chzdev** command to configure devices and device drivers on IBM Z. Supported devices include storage devices (DASD and zFCP) and networking devices (QETH, CTC, and LCS).

You can apply configuration changes to the active configuration of the currently running system, or to the persistent configuration stored in configuration files:

- Changes to the active configuration are effective immediately. They are lost on reboot, when a device driver is unloaded, or when a device becomes unavailable.
- Changes to the persistent configuration are applied when the system boots, when a device driver is loaded, or when a device becomes available.

By default, **chzdev** applies changes to the active and the persistent configuration.

You can also temporarily remove existing auto-configuration data, see “Managing auto-configuration data” on page 22.

chzdev supports enabling and disabling devices, exporting and importing configuration data to and from a file, and displaying a list of available device types and attributes.

chzdev actions and options



Notes:

[1] This option is intended for utilities that run early in the boot process.

Device selection



Device type selection

Attribute settings



Notes:

[1] Specify at least one of the options

chzdev help functions



where:

<type>
restricts the scope of an action to the specified device type:

- Specify a device type and optionally a device ID to work on devices with matching type and ID only.

- Specify a device type together with the `--type` option to manage the configuration of the device type itself.

Note:

As a precaution, use the most specific device type when you configure a device by ID. Otherwise, the same device ID might accidentally match other devices of a different subtype. To get a list of supported device types, use the `--list-types` option.

<device>
selects a single device or a range of devices by device ID. Separate multiple IDs or ranges with a comma (,). To select a range of devices, specify the ID of the first and the last device in the range separated by a hyphen (-).

-t or --type *<device_type>*
selects a device type as target for a configuration or query action.

<attribute=value>
specifies a device attribute and its value. To specify multiple attributes, separate attribute-value pairs with a blank.

You can use the `--list-attributes` option to display a list of available attributes and the `--help-attribute` to get more detailed information about a specific attribute.

Tip: To specify an attribute that is not known to **chzdev**, use the `--force` option.

-r or --remove *<attrib>*
removes the setting for attribute *<attrib>*.

Active configuration
For attributes that maintain a list of values, clears all values for that list.

Persistent configuration
Removes any setting for the specified attribute. When the device or device driver is configured again, the attribute is set to its default value.

Some attributes cannot be removed.

-R or --remove-all
removes the settings for all attributes of the selected device or device driver.

Active configuration
For attributes that maintain a list of values, clears all values for that list.

Persistent configuration
Removes all attribute settings that can be removed. When the device or device driver is configured again, the attribute is set to its default value.

Some attributes cannot be removed.

--by-attrib *<attrib=value> | <attrib!=value>*
selects devices with a specific attribute, *<attrib>* that has a value of *<value>*. When specified as *<attrib>*!=*<value>*, selects all devices that do not provide an attribute named *<attrib>* with a value of *<value>*.

Tip: You can use the `--list-attributes` option to display a list of available attributes and the `--help-attribute` to get more detailed information about a specific attribute.

--by-interface *<interface>*
selects devices by network interface, for example, encf5a0. *<interface>* must be the name of an existing networking interface.

--by-node *<device_node>*
selects devices by device node, for example, `/dev/sda`. *<device_node>* must be the path to the device node for a block device or character device.

Note: If *<device_node>* is the device node for a logical device (such as a device mapper device), **lszdev** tries to resolve the corresponding physical device nodes. The **lsblk** tool must be available for this resolution to work.

--by-path *<path>*
selects devices by file-system path, for example, `/usr`. The *<path>* parameter can be the mount point of a mounted file system, or a path on that file system.

Note: If the file system that provides *<path>* is stored on multiple physical devices (such as supported by btrfs), **lszdev** tries to resolve the corresponding physical device nodes. The **lsblk** tool must be available and the file system must provide a valid UUID for this resolution to work.

--all
selects all existing and configured devices.

--configured
narrows the selection to those devices for which a persistent configuration exists.

--existing
narrows the selection to all devices that are present in the active configuration.

--configured --existing
specifying both `--configured` and `--existing` narrows the selection to devices that are present in both configurations, persistent and active.

--online
narrows the selection to devices that are enabled in the active configuration.

--offline
narrows the selection to devices that are disabled in the active configuration.

-a or --active
applies changes to the active configuration only. The persistent configuration is not changed unless you also specify `--persistent`.

Note: Changes to the active configuration are effective immediately. They are lost on reboot, when a device driver is unloaded, or when a device becomes unavailable.

-p or --persistent
applies changes to the persistent configuration only. The persistent configuration takes effect when the system boots, when a device driver is loaded, or when a device becomes available.

--auto-conf
applies changes to the auto-configuration. Changes to the auto-configuration take effect when a device becomes available, but do not persist across reboots. This option is primarily intended for use by the boot process. For details about auto-configuration data, see Chapter 3, “Device auto-configuration for Linux in LPAR mode,” on page 21.

--export *<filename>*|-
writes configuration data to a text file called *<filename>*. If a single hyphen (-) is specified instead of a file name, data is written to the standard output stream. The output format of this option can be used with the `--import` option. To reduce the scope of exported configuration data, you can select specific devices, a device type, or define whether to export only data for the active or persistent configuration.

--import *<filename>*|-
reads configuration data from *<filename>* and applies it. If a single hyphen (-) is specified instead of a file name, data is read from the standard input stream. The input format must be the same as the format produced by the `--export` option.

By default, all configuration data that is read is also applied. To reduce the scope of imported configuration data, you can select specific devices, a device type, or define whether to import only data for the active or persistent configuration.

You can use this option to import auto-configuration data, see “Displaying auto-configuration data” on page 22.

-a or --apply
applies the persistent configuration of all selected devices and device types to the active configuration.

-e or --enable
enables the selected devices. Any steps necessary for the devices to function are taken, for example: create a CCW group device, remove a device from the CIO exclusion list, or set a CCW device online.

Active configuration
Performs all setup steps required for a device to become operational, for example, as a block device or as a network interface.

Persistent configuration
Creates configuration files and settings associated with the selected devices.

-d or --disable
disables the selected devices.

Active configuration
Disables the selected devices by reverting the configuration steps necessary to enable them.

Persistent configuration
Removes configuration files and settings associated with the selected devices.

--dry-run
processes the actions and displays command output without changing the configuration of any devices or device types. Combine with `--verbose` to display details about skipped configuration steps.

--base *<path>* | *<key=value>*
changes file system paths that are used to access files. If *<path>* is specified without an equal sign (=), it is used as base path for accessing files in the active and persistent configuration. If the specified parameter is in *<key=value>* format, only those paths that begin with *<key>* are modified. For these paths, the initial *<key>* portion is replaced with *<value>*.

Example: `lszdev --active --base /etc=/mnt/etc`

-f or --force
overrides safety checks and confirmation questions, including:

- More than 256 devices selected
- Configuring unknown attributes
- Combining apparently inconsistent settings

--no-root-update
skips any additional steps that are required to change the root device configuration persistently. Typically such steps include rebuilding the initial RAM disk, or modifying the kernel command line.

-y or --yes
answers all confirmation questions with "yes".

-q or --quiet
prints only minimal run-time information.

-l or --list-attributes
lists all supported device or device type attributes, including a short description. Use the `--help-attribute` option to get more detailed information about an attribute.

-L or --list-types
lists the name and a short description for all device types supported by chzdev.

--no-settle
continue without waiting for udev processing to complete. This option is intended for utilities that run during the early initial RAM disc stage of the boot process, when udev is not fully functional.

-V or --verbose
prints additional run-time information.

-v or --version
displays the version number of **chzdev**, then exits.

-h or --help
displays help information for the command.

-H or --help-attribute
displays help information for the command.

Examples

- To enable an FCP device with device number 0.0.198d, WWPN 0x50050763070bc5e3, and LUN 0x4006404600000000 in the currently active configuration, issue:

```
# chzdev --enable --active zfcp-lun 0.0.198d:0x50050763070bc5e3:0x4006404600000000
```

- To export configuration data for all FCP devices to a file called `config.txt`, issue:

```
# chzdev zfcp-lun --all --export config.txt
```

- To enable a QETH device in the currently active configuration, issue:

```
# chzdev --enable --active qeth 0.0.a000:0.0.a001:0.0.a002
```

- To get help for the QETH-device attribute layer2, issue:

```
# chzdev qeth --help-attribute layer2
```

- To enable a device that provides networking interface encf500, issue:

```
# chzdev --by-interface encf500 --active
```

- To enable DASD 0.0.8000 in the currently active configuration, issue:

```
# chzdev -e -a dasd 8000
```

- To enable DASDs 0.0.1000 and 0.0.2000 through 0.0.2010 in the currently active configuration, issue:

```
# chzdev dasd 1000,200-2010 -e -a
```

- To change the dasd device type parameter eer_pages to 14 in the currently active configuration, issue:

```
# chzdev dasd --type eer_pages=14 -a
```

See the man page for information about the command exit codes.

Activating a device early during the boot process

Use the `zdev:early` device attribute to activate a device early during the boot process and to override any existing auto-configuration with a persistent device configuration.

zdev:early=1
The device is activated during the initial RAM disc phase according to the persistent configuration.

zdev:early=0
The device is activated as usual during the boot process. This is the default. If auto-configuration data is present, the device is activated during the initial RAM disc phase according to the auto-configuration.

Example: To assure that the qeth device with bus-ID 0.0.f5f0 is enabled early in the boot process in layer 2 mode, issue:

```
# chzdev -e qeth 0.0.f5f0:0.0.f5f1:0.0.f5f2 layer2=1 zdev:early=1
```

See also Chapter 3, “Device auto-configuration for Linux in LPAR mode,” on page 21.

Files used

The **chzdev** command uses these files:

/etc/udev/rules.d/
: **chzdev** creates udev rules to store the persistent configuration of devices.

/etc/modprobe.d/
: **chzdev** creates modprobe configuration files to store the persistent configuration of certain device types.

/run
: holds udev rules that represent auto-configuration data. **chzdev** can remove entries so the auto-configuration settings do not apply as devices appear on the running Linux instance.

cio_ignore - Manage the I/O exclusion list

Use the **cio_ignore** command to specify I/O devices that are to be ignored by Linux.

When an instance of Linux on IBM Z boots, it senses and analyzes all available I/O devices. You can use the cio_ignore kernel parameter (see "cio_ignore - List devices to be ignored" on page 780) to specify devices that are to be ignored. This exclusion list can cover all possible devices, even devices that do not actually exist.

The **cio_ignore** command manages this exclusion list on a running Linux instance. You can change the exclusion list and display it in different formats. Changes made with the **cio_ignore** command do not persist across reboots.

cio_ignore syntax



Where:

-a or --add
adds one or more device specifications to the exclusion list.

When you add specifications for a device that is already sensed and analyzed, there is no immediate effect of adding it to the exclusion list. For example, the device still appears in the output of the **lscss** command and can be set online. However, if the device subsequently becomes unavailable, it is ignored when it reappears. For example, if the device is detached in z/VM, it is ignored when it is attached again.

See the **-p** option about how to make devices that are already sensed and analyzed unavailable to Linux.

-r or --remove
removes one or more device specifications from the exclusion list.

When you remove device specifications from the exclusion list, the corresponding devices are sensed and analyzed if they exist. Where possible, the corresponding device driver is informed, and the devices become available to Linux.

<device_bus_id>
identifies a device to be added or removed.

Device bus-IDs are of the form 0.*<n>*.*<devno>*, where *<n>* is a subchannel set ID and *<devno>* is a device number. If the subchannel set ID is 0, you can abbreviate the specification to the device number, with or without a leading 0x.

Example: The specifications 0.0.0190, 190, 0190, and 0x190 are all equivalent. There is no short form of 0.1.0190.

<from_device_bus_id>-<to_device_bus_id>
identifies a range of devices. *<from_device_bus_id>* and *<to_device_bus_id>* have the same format as *<device_bus_id>*.

-A or --add-all
adds the entire range of possible devices to the exclusion list.

When you add specifications for a device that is already sensed and analyzed, there is no immediate effect of adding it to the exclusion list. For example, the device still appears in the output of the **lscss** command and can be set online. However, if the device subsequently becomes unavailable, it is ignored when it reappears. For example, if the device is detached in z/VM, it is ignored when it is attached again.

See the **-p** option about making devices that are already sensed and analyzed unavailable to Linux.

-R or --remove-all
removes all devices from the exclusion list.

When you remove device specifications from the exclusion list, the corresponding devices are sensed and analyzed if they exist. Where possible, the corresponding device driver is informed, and the devices become available to Linux.

-l or --list
displays the current exclusion list.

-L or --list-not-blacklisted
displays specifications for all devices that are not in the current exclusion list.

-k or --kernel-param
returns the current exclusion list in kernel parameter format.

You can make the current exclusion list persistent across rebooting Linux by using the output of the **cio_ignore** command with the -k option as part of the Linux kernel parameter. See Chapter 4, “Kernel and module parameters,” on page 25.

-u or --unused
discards the current exclusion list and replaces it with a specification for all devices that are not online. This includes specification for possible devices that do not actually exist.

-p or --purge
makes all devices that are in the exclusion list and that are currently offline unavailable to Linux. This option does not make devices unavailable if they are online.

-h or --help
displays help information for the command. To view the man page, enter **man cio_ignore**.

-v or --version
displays version information.

Examples

These examples illustrate common uses for **cio_ignore**.

- The following command shows the current exclusion list:

```
# cio_ignore -l
Ignored devices:
================
0.0.0000-0.0.7e8e
0.0.7e94-0.0.f4ff
0.0.f503-0.0.ffff
0.1.0000-0.1.ffff
0.2.0000-0.2.ffff
0.3.0000-0.3.ffff
```

- The following command shows specifications for the devices that are not on the exclusion list:

```
# cio_ignore -L
Accessible devices:
===================
0.0.7e8f-0.0.7e93
0.0.f500-0.0.f502
```

- The following command adds a device specification, `0.0.7e8f`, to the exclusion list:

```
# cio_ignore -a 0.0.7e8f
```

  The previous example then becomes:

```
# cio_ignore -L
Accessible devices:
===================
0.0.7e90-0.0.7e93
0.0.f500-0.0.f502
```

- The following command shows the current exclusion list in kernel parameter format:

```
# cio_ignore -k
cio_ignore=all,!7e90-7e93,!f500-f502
```

cmsfs-fuse - Mount a z/VM CMS file system

Use the cmsfs-fuse command to mount the enhanced disk format (EDF) file system on a z/VM minidisk.

In Linux, the minidisk is represented as a DASD and the file system is mounted as a cmsfs-fuse file system. The cmsfs-fuse file system translates the record-based file system on the minidisk into Linux semantics.

Through the cmsfs-fuse file system, the files on the minidisk become available to applications on Linux. Applications can read from and write to files on minidisks. Optionally, the cmsfs-fuse file system converts text files between EBCDIC on the minidisk and ASCII within Linux.

Attention: You can inadvertently damage files and lose data when directly writing to files within the cmsfs-fuse file system. To avoid problems when you write, multiple restrictions must be observed, especially regarding linefeeds (see restrictions for write).

Tip: If you are unsure about how to safely write to a file on the cmsfs-fuse file system, copy the file to a location outside the cmsfs-fuse file system, edit the file, and then copy it back to its original location.

Use **fusermount** to unmount file systems that you mounted with **cmsfs-fuse**. See the **fusermount** man page for details.

Before you begin:

- The fuse module must be loaded, for example, with **modprobe fuse**.
- The FUSE library must be installed on your system. Install the fuse-libs RPM delivered with Red Hat Enterprise Linux 9.1.
- The DASD must be online.
- Depending whether you intend to read, write, or both, you must have the appropriate permissions for the device node.

cmsfs-fuse syntax



Where:

-a or --ascii
treats all files on the minidisk as text files and converts them from EBCDIC to ASCII.

-t or --filetype

treats files with extensions as listed in the **cmsfs-fuse** configuration file as text files and converts them from EBCDIC to ASCII.

The **cmsfs-fuse** command uses `~/.cmsfs-fuse/filetypes.conf` as the configuration file. If this file within the user's home directory does not exist, the default configuration file `/etc/cmsfs-fuse/filetypes.conf` is used.

The `filetypes.conf` file lists one file type per line. Lines that start with a number sign (#) followed by a space are treated as comments and are ignored.

--from *<code-page>*
specifies the encoding of the files on the z/VM minidisk. If this option is not specified, code page CP1047 is used. Enter **iconv --list** to display a list of all available code pages.

--to *<code-page>*
specifies the encoding to which the files on the z/VM minidisk are converted in Linux. If this option is not specified, code page ISO-8859-1 is used. Enter **iconv --list** to display a list of all available code pages.

<mount-options>
options as available for the **mount** command. See the **mount** man page for details.

<fuse-options>
options for FUSE. The following options are supported by the **cmsfs-fuse** command. To use an option, it must also be supported by the version of FUSE that you have.

-d or -o debug
enables debug output (implies **-f**).

-f
runs the command as a foreground operation.

-o allow_other
allows access to other users.

-o allow_root
allows access to root.

-o default_permissions
enables permission checking by the kernel.

-o max_read=*<n>*
sets maximum size of read requests.

-o kernel_cache
caches files in the kernel.

-o [no]auto_cache
enables or disables caching based on modification times.

-o umask=*<mask>*
sets file permissions (octal).

-o uid=*<n>*
sets the file owner.

-o gid=*<n>*
sets the file group.

-o max_write=*<n>*
sets the maximum size of write requests.

-o max_readahead=*<n>*
sets the maximum readahead value.

-o async_read
performs reads asynchronously (default).

-o sync_read
performs reads synchronously.

-o big_writes
enables write operations with more than 4 KB.

<node>
the device node for the DASD that represents the minidisk in Linux.

<mount-point>
the mount point in the Linux file system where you want to mount the CMS file system.

-h or --help
displays help information for the command. To view the man page, enter **man cmsfs-fuse**.

-v or --version
displays version information for the command.

You can use the following extended attributes to handle the CMS characteristics of a file:

user.record_format
specifies the format of the file. The format is F for fixed record length files and V for variable record length files. This attribute can be set only for empty files. The default file format for new files is V.

user.record_lrecl
specifies the record length of the file. This attribute can be set only for an empty fixed record length file. A valid record length is an integer in the range 1-65535.

user.file_mode
specifies the CMS file mode of the file. The file mode consists of a mode letter from A-Z and mode number in the range 0 - 6. The default file mode for new files is A1.

You can use the following system calls to work with extended attributes:

listxattr
to list the current values of all extended attributes.

getxattr
to read the current value of a particular extended attribute.

setxattr
to set a particular extended attribute.

You can use these system calls through the **getfattr** and **setfattr** commands, available from the attr RPM. For more information, see the man pages of these commands and of the listxattr, getxattr, and setxattr system calls.

Restrictions

When you work with files in the cmsfs-fuse file system, restrictions apply for the following system calls:

write
Be aware of the following restrictions when you writeto a file on the cmsfs-fuse file system:

Write location
Writing is supported only at the end of a file.

Padding
For fixed-length record files, the last record is padded to make up a full record length. The padding character is zero in binary mode and the space character in ASCII mode.

Sparse files
Sparse files are not supported. To prevent the **cp** tool from writing in sparse mode specify `-sparse=never`.

Records and linefeeds with ASCII conversion (-a and -t)

In the ASCII representation of an EBCDIC file, a linefeed character determines the end of a record. Follow these rules about linefeed characters requirements when you write to EBCDIC files in ASCII mode:

For fixed record length files
Use linefeed characters to separate character strings of the fixed record length.

For variable record length files
Use linefeed characters to separate character strings. The character strings must not exceed the maximum record length.

The CMS file system does not support empty records. cmsfs-fuse adds a space to records that consist of a linefeed character only.

rename and creat
Uppercase file names are enforced.

truncate
Only shrinking of a file is supported. For fixed-length record files, the new file size must be a multiple of the record length.

Examples

- To mount the CMS file system on the minidisk represented by the file node `/dev/dasde` at `/mnt`:

```
# cmsfs-fuse /dev/dasde /mnt
```

- To mount the CMS file system on the minidisk represented by the file node `/dev/dasde` at `/mnt` and enable EBCDIC to ASCII conversion for text files with extensions as specified in `~/.cmsfs-fuse/filetypes.conf` or `/etc/cmsfs-fuse/filetypes.conf` if the former does not exist:

```
# cmsfs-fuse -t /dev/dasde /mnt
```

- To mount the CMS file system on the minidisk represented by the file node `/dev/dasde` at `/mnt` and allow root to access the mounted file system:

```
# cmsfs-fuse -o allow_root /dev/dasde /mnt
```

- To unmount the CMS file system that was mounted at `/mnt`:

```
# fusermount -u /mnt
```

- To show the record format of a file, PROFILE.EXEC, on a z/VM minidisk that is mounted on `/mnt`:

```
# getfattr -n user.record_format /mnt/PROFILE.EXEC
F
```

- To set record length 80 for an empty fixed record format file, PROFILE.EXEC, on a z/VM minidisk that is mounted on `/mnt`:

```
# setfattr -n user.record_lrecl -v 80 /mnt/PROFILE.EXEC
```

cpacfstats - Monitor CPACF cryptographic activity

Use the **cpacfstats** command to display the number of cryptographic operations that are performed by the Central Processor Assist for Cryptographic Function (CPACF). You can display and enable, disable, or reset specific hardware counters for AES, DES, ECC, TDES, SHA, and pseudo random functions.

CPACF performance counters are available on LPARs only.

All counters are initially disabled and must be enabled in the LPAR activation profile on the SE or HMC to measure CPACF activities. There is a slight performance penalty with CPACF counters enabled.

Prerequisites

- On the HMC or SE, authorize the LPAR for each counter set you want to use. Customize the LPAR activation profile and modify the Counter Facility Security Options. You need to activate the "Crypto activity counter set authorization control" checkbox.
- The cpacfstatsd daemon must be running. Check the syslog for the message: `cpacfstatsd: Running`. To start the daemon, issue:

```
# cpacfstatsd
```

  The daemon requires root privileges to open and work with the perf kernel API functions. Issue **man cpacfstatsd** for more information about the daemon.

Note: The counter value is increased once per API call and also for every additional 4096 bytes of data.

Setting up the cpacfstats group

Only root and members of the group cpacfstats are allowed to communicate with the daemon process. You must create the group and add users to it.

1. Create the group cpacfstats:

```
# groupadd cpacfstats
```

2. Add all users who are allowed to run the cpacfstats client application to the group:

```
usermod -a -G cpacfstats <user>
```

All users in the cpacfstats group are also able to modify the CPACF counter states (enable, disable, reset).

cpacfstats syntax



Where:

-e or --enable *<counter>*
enables one or all CPACF performance counters. The optional counter argument can be one of:

aes
counts all AES-related cipher message CPACF instructions.

des
counts all DES- and 3DES-related cipher message CPACF instructions.

ecc
counts all ECC (elliptic curve cryptography) related CPACF instructions.

rng
counts all pseudo-random related CPACF instructions.

sha
counts all message digest (that is, SHA-1 through SHA-512) related CPACF instructions.

all
counts all CPACF instructions.

If you omit the counter, all performance counters are enabled. Enabling a counter does not reset it. New events are added to the current counter value.

-d or --disable *<counter>*
disables one or all CPACF performance counters. If you omit the counter, all performance counters are disabled. Disabling a counter does not reset it. The counter value is preserved when a counter is disabled, and counting resumes with the preserved value when the counter is re-enabled.

-r or --reset *<counter>*
resets one or all CPACF performance counters. If you omit the counter, all performance counters are reset to 0.

-p or --print *<counter>*
displays the value of one or all CPACF performance counters. If you omit the counter, all performance counters are displayed.

-h or --help
displays help information for the command. To view the command man page, enter **`man cpacfstats`**.

-v or --version
displays version information for **`cpacfstats`**.

If no option is specified, the command prints out all the counters (as if **`--print all`** were specified).

Examples

- To print status and values of all CPACF performance counters:

```
# cpacfstats
des counter: disabled
aes counter: disabled
sha counter: disabled
rng counter: disabled
```

- To enable the AES CPACF performance counter:

```
# cpacfstats --enable aes
aes counter: 0
```

- To enable all CPACF performance counters:

```
# cpacfstats -e
des counter: 0
aes counter: 192
sha counter: 0
rng counter: 0
```

For the already enabled aes counter, the value is not reset.

cpuplugd - Control CPUs and memory

Use the **cpuplugd** command and a set of rules in a configuration file to dynamically enable or disable CPUs. For Linux on z/VM, you can also dynamically add or remove memory.

Rules that are tailored to a particular system environment and the associated workload can increase performance. The rules can include various system load variables.

You can start cpuplugd from the command line in two ways:

- With the service utility
- Through a command-line interface

Note: Do not run multiple instances of cpuplugd simultaneously.

cpuplugd service utility syntax

If you run the **cpuplugd** daemon through the service utility, you configure the daemon through specifications in the `/etc/cpuplugd.conf` configuration file.

Where:

start
starts the cpuplugd daemon with the configuration in `/etc/cpuplugd.conf`. Do not run multiple instances of cpuplugd simultaneously. Check the cpuplugd status before starting a new instance.

stop
stops the cpuplugd daemon.

status
shows current status of cpuplugd.

restart
stops and restarts the cpuplugd daemon. Useful to re-read the configuration file when it was changed.

Examples

- To stop a running instance of cpuplugd:

```
# service cpuplugd stop
```

- To display the status:

```
# service cpuplugd status
...
   Active: active (running) ...
```

cpuplugd command-line syntax

You can start cpuplugd through a command interface.

Before you begin: Do not run multiple instances of cpuplugd simultaneously. Check the cpuplugd status through the service utility before you issue the **cpuplugd** command (see “cpuplugd service utility syntax” on page 606).

cpuplugd syntax

Where:

-c *<config file>* or --config *<config file>*
specifies the path to the configuration file with the rules (see "Configuration file structure" on page 607).

After you install cpuplugd for the first time, you can find a sample configuration file at `/etc/cpuplugd.conf`. If you are upgrading from a prior version of cpuplugd, see "Migrating old configuration files" on page 608.

-f or --foreground
runs in foreground.

-V or --verbose
displays verbose messages.

-h or --help
displays help information for the command. To view the command man page, enter **`man cpuplugd`**. To view the man page for the configuration file, enter **`man cpuplugd.conf`**

-v or --version
displays version information.

Examples

- To start cpuplugd in daemon mode with a configuration file `/etc/cpuplugd.conf`:

```
# cpuplugd -c /etc/cpuplugd
```

- To run cpuplugd in the foreground with verbose messages and with a configuration file `/etc/cpuplugd.conf`:

```
# cpuplugd -V -f -c /etc/cpuplugd
```

Configuration file structure

The cpuplugd configuration file can specify rules for controlling the number of active CPUs and for Linux on z/VM, for controlling the amount of memory.

The configuration file contains:

- *<variable>*="*<value>*" pairs

  These pairs must be specified within one line. The maximum valid line length is 2048 characters. The values can be decimal numbers or algebraic or Boolean expressions.

- Comments

  Any part of a line that follows a number sign (#) is treated as a comment. There can be full comment lines with the number sign at the beginning of the line or comments can begin in mid-line.

- Empty lines

Attention: These configuration file samples illustrate the syntax of the configuration file. Do not use the sample rules on production systems. Useful rules differ considerably, depending on the workload, resources, and requirements of the system for which they are designed.

Migrating old configuration files

With Red Hat Enterprise Linux 6.2, an enhanced version of **cpuplugd** was introduced.

This enhanced version includes extensions to the configuration file and a new sample configuration file, `/etc/sysconfig/cpuplugd`.

If a configuration file from a prior version of cpuplugd already exists at `/etc/sysconfig/cpuplugd`, this file is not replaced but complemented with new variables. The new sample configuration file is then copied to `/var/adm/fillup-templates/sysconfig.cpuplugd`.

The new sample file contains comments that describe the enhanced file layout. View the file to see this information. Consider merging the existing configuration file with a copy of the new sample file to obtain a configuration file with the existing rules, the new variables, and the new descriptions.

Basic configuration file for CPU control

A configuration file for dynamically enabling or disabling CPUs has several required specifications.

The configuration file sample of Figure 107 on page 608 has been reduced to the specifications that are required for dynamically enabling or disabling CPUs.

```
UPDATE="10"
CPU_MIN="2"
CPU_MAX="10"

HOTPLUG = "idle < 10.0"
HOTUNPLUG = "idle > 100"
```

Figure 107. Simplified configuration file with CPU hotplug rules

In the configuration file:

UPDATE
specifies the time interval, in seconds, at which cpuplugd evaluates the rules and, if a rule is met, enables or disables CPUs. This variable is also required for controlling memory (see "Basic configuration file for memory control" on page 609).

In the example, the rules are evaluated every 10 seconds.

CPU_MIN
specifies the minimum number of CPUs. Even if the rule for disabling CPUs is met, cpuplugd does not reduce the number of CPUs to less than this number.

In the example, the number of CPUs cannot become less than 2.

CPU_MAX
specifies the maximum number of CPUs. Even if the rule for enabling CPUs is met, cpuplugd does not increase the number of CPUs to more than this number. If 0 is specified, the maximum number of CPUs is the number of CPUs available on the system.

In the example, the number of CPUs cannot become more than 10.

HOTPLUG
specifies the rule for dynamically enabling CPUs. The rule resolves to a boolean true or false. Each time this rule is true, cpuplugd enables one CPU, unless the number of CPUs has already reached the maximum specified with CPU_MAX.

Setting HOTPLUG to `0` disables dynamically adding CPUs.

In the example, a CPU is enabled when the idle times of all active CPUs sum up to less than 10.0%. See "Keywords for CPU hotplug rules" on page 610 for information about available keywords.

HOTUNPLUG
specifies the rule for dynamically disabling CPUs. The rule resolves to a boolean true or false. Each time this rule is true, cpuplugd disables one CPU, unless the number of CPUs has already reached the minimum specified with CPU_MIN.

Setting HOTUNPLUG to `0` disables dynamically removing CPUs.

In the example, a CPU is disabled when the idle times of all active CPUs sum up to more than 100%. See "Keywords for CPU hotplug rules" on page 610 for information about available keywords.

If one of these variables is set more than once, only the last occurrence is used. These variables are not case sensitive.

If both the HOTPLUG and HOTUNPLUG rule are met simultaneously, HOTUNPLUG is ignored.

Basic configuration file for memory control

For Linux on z/VM, you can also use cpuplugd to dynamically add or take away memory. There are several required specifications for memory control.

The configuration file sample of Figure 108 on page 609 was reduced to the specifications that are required for dynamic memory control.

```
UPDATE="10"
CMM_MIN="0"
CMM_MAX="131072"   # 512 MB
CMM_INC="10240"    #  40 MB

MEMPLUG = "swaprate > 250"
MEMUNPLUG = "swaprate < 10"
```

Figure 108. Simplified configuration file with memory hotplug rules

In the configuration file:

UPDATE
specifies the time interval, in seconds, at which cpuplugd evaluates the rules and, if a rule is met, adds or removes memory. This variable is also required for controlling CPUs (see "Basic configuration file for CPU control" on page 608).

In the example, the rules are evaluated every 10 seconds.

CMM_MIN
specifies the minimum amount of memory, in 4 KB pages, that Linux surrenders to the CMM static page pool (see "Cooperative memory management background" on page 410). Even if the MEMPLUG rule for taking memory from the CMM static page pool and adding it to Linux is met, cpuplugd does not decrease this amount.

In the example, the amount of memory that is surrendered to the static page pool can be reduced to 0.

CMM_MAX
specifies the maximum amount of memory, in 4 KB pages, that Linux surrenders to the CMM static page pool (see "Cooperative memory management background" on page 410). Even if the MEMUNPLUG rule for removing memory from Linux and adding it to the CMM static page pool is met, cpuplugd does not increase this amount.

In the example, the amount of memory that is surrendered to the static page pool cannot become more than 131072 pages of 4 KB (512 MB).

CMM_INC
specifies the amount of memory, in 4 KB pages, that is removed from Linux when the MEMUNPLUG rule is met. Removing memory from Linux increases the amount that is surrendered to the CMM static page pool.

In the example, the amount of memory that is removed from Linux is 10240 pages of 4 KB (40 MB) at a time.

CMM_DEC
Optional: specifies the amount of memory, in 4 KB pages, that is added to Linux when the MEMPLUG rule is met. Adding memory to Linux decreases the amount that is surrendered to the CMM static page pool.

If this variable is omitted, the amount of memory that is specified for CMM_INC is used.

In the example, CMM_DEC is omitted and the amount of memory added to Linux is 10240 pages of 4 KB (40 MB) at a time, as specified with CMM_INC.

MEMPLUG
specifies the rule for dynamically adding memory to Linux. The rule resolves to a boolean true or false. Each time this rule is true, cpuplugd adds the number of pages that are specified by CMM_DEC, unless the CMM static page pool already reached the minimum that is specified with CMM_MIN.

Setting MEMPLUG to `0` disables dynamically adding memory to Linux.

In the example, memory is added to Linux if there are more than 250 swap operations per second. See “Keywords for memory hotplug rules” on page 611 for information about available keywords.

MEMUNPLUG
specifies the rule for dynamically removing memory from Linux. The rule resolves to a boolean true or false. Each time this rule is true, cpuplugd removes the number of pages that are specified by CMM_INC, unless the CMM static page pool already reached the maximum that is specified with CMM_MAX.

Setting MEMUNPLUG to `0` disables dynamically removing memory from Linux.

In the example, memory is removed from Linux when there are less than 10 swap operations per second. See “Keywords for memory hotplug rules” on page 611 for information about available keywords.

If any of these variables are set more than once, only the last occurrence is used. These variables are not case-sensitive.

If both the MEMPLUG and MEMUNPLUG rule are met simultaneously, MEMUNPLUG is ignored.

CMM_DEC and CMM_INC can be set to a decimal number or to a mathematical expression that uses the same algebraic operators and variables as the MEMPLUG and MEMUNPLUG hotplug rules (see “Keywords for memory hotplug rules” on page 611 and “Writing more complex rules” on page 612).

Predefined keywords

There is a set of predefined keywords that you can use for CPU hotplug rules and a set of keywords that you can use for memory hotplug rules. All predefined keywords are case sensitive.

Keywords for CPU hotplug rules

Use the predefined keywords in the CPU hotplug rules, HOTPLUG and HOTUNPLUG.

loadavg
is the current load average.

onumcpus
is the current number of online CPUs.

runnable_proc
is the current number of runnable processes.

user
is the current CPU user percentage.

nice
is the current CPU nice percentage.

system
is the current CPU system percentage.

idle
is the current CPU idle percentage.

iowait
is the current CPU iowait percentage.

irq
is the current CPU irq percentage.

softirq
is the current CPU softirq percentage.

steal
is the current CPU steal percentage.

guest
is the current CPU guest percentage for a z/VM or KVM guest.

guest_nice
is the current CPU guest_nice percentage for a z/VM or KVM guest.

cpustat.*<name>*
is data from `/proc/stat` and `/proc/loadavg`. In the keyword, *<name>* can be any of the previously listed keywords, for example, `cpustat.idle`. See the proc man page for more details about the data that is represented by these keywords.

With this notation, the keywords resolve to raw timer ticks since system start, not to current percentages. For example, `idle` resolves to the current idle percentage and `cpustat.idle` resolves to the total timer ticks spent idle. See "Using historical data" on page 612 about how to obtain average and percentage values.

`loadavg`, `onumcpus`, and `runnable_proc` are not percentages and resolve to the same values as `cpustat.loadavg`, `cpustat.onumcpus`, and `cpustat.runnable_proc`.

cpustat.total_ticks
is the total number of timer ticks since system start.

time
is the UNIX epoch time in the format "seconds.microseconds".

Percentage values are accumulated for all online CPUs. Hence, the values for the percentages range from 0 to 100 × (number of online CPUs). To get the average percentage per CPU device, divide the accumulated value by the number of CPUs. For example, `idle / onumcpus` yields the average idle percentage per CPU.

Keywords for memory hotplug rules

Use the predefined keywords in the memory hotplug rules, MEMPLUG and MEMUNPLUG.

apcr
is the number of page cache operations, pgpin + pgpout, from `/proc/vmstat` in 512-byte blocks per second.

freemem
is the amount of free memory in MB.

swaprate
is the number of swap operations, pswpin + pswpout, from `/proc/vmstat` in 4 KB pages per second.

meminfo.*<name>*
is the value for the symbol *<name>* as shown in the output of **`cat /proc/meminfo`**. The values are plain numbers but refer to the same units as those used in `/proc/meminfo`.

vmstat.*<name>*
is the value for the symbol *<name>* as shown in the output of **`cat /proc/vmstat`**.

Using historical data

Historical data is available for the keyword `time` and the sets of keywords `cpustat.<name>`, `meminfo.<name>`, and `vmstat.<name>`.

See "Keywords for CPU hotplug rules" on page 610 and "Keywords for memory hotplug rules" on page 611 for details about these keywords.

Use the suffixes [*<n>*] to retrieve the data of *<n>* intervals in the past, where *<n>* can be in the range 0 - 100.

Examples

cpustat.idle
: yields the current value for the counted idle ticks.

cpustat.idle[1]
: yields the idle ticks as counted one interval ago.

cpustat.idle[5]
: yields the idle ticks as counted five intervals ago.

cpustat.idle - cpustat.idle[5]
: yields the idle ticks during the past five intervals.

time - time[1]
: yields the length of an update interval in seconds.

cpustat.total_ticks - cpustat.total_ticks[5]
: yields the total number of ticks during the past five intervals.

(cpustat.idle - cpustat.idle[5]) / (cpustat.total_ticks - cpustat.total_ticks[5])
: yields the average ratio of idle ticks to total ticks during the past five intervals.

  Multiplying this ratio with 100 yields the percentage of idle ticks during the last five intervals.

  Multiplying this ratio with 100 * `onumcpus` yields the accumulated percentage of idle ticks for all processors during the last five intervals.

Writing more complex rules

In addition to numbers and keywords, you can use mathematical and Boolean operators, and you can use user-defined variables to specify rules.

- The keywords of "Predefined keywords" on page 610
- Decimal numbers
- The mathematical operators

  +
  : addition

  -
  : subtraction

  *****
  : multiplication

  /
  : division

  <
  : less than

  >
  : greater than
- Parentheses (and) to group mathematical expressions
- The Boolean operators

&
: and

|
: or

!
: not

- User-defined variables

  You can specify complex calculations as user-defined variables, which can then be used in expressions. User-defined variables are case-sensitive and must not match a pre-defined variable or keyword. In the configuration file, definitions for user-defined variables must precede their use in expressions.

  Variable names consist of alphanumeric characters and the underscore (_) character. An individual variable name must not exceed 128 characters. All user-defined variable names and values, in total, must not exceed 4096 characters.

Examples

- `HOTPLUG = "loadavg > onumcpus + 0.75"`
- `HOTPLUG = "(loadavg > onumcpus + 0.75) & (idle < 10.0)"`
- ```
  my_idle_rate = "(cpustat.idle - cpustat.idle[5]) / (cpustat.total_ticks -
  cpustat.total_ticks[5])"
  my_idle_percent_total = "my_idle_rate * 100 * onumcpus"
  ...
  HOTPLUG = "(loadavg > onumcpus + 0.75) & (my_idle_percent_total < 10.0)"
  ```

Sample configuration file

A typical configuration file includes multiple user-defined variables and values from procfs, for example, to calculate the page scan rate or the cache size.
  ```

```
# Required static variables

CPU_MIN="1"
CPU_MAX="0"
UPDATE="1"
CMM_MIN="0"
CMM_MAX="131072"    # 512 MB

# User-defined variables

pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] +
vmstat.pgscan_direct_movable[0]"
pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] +
vmstat.pgscan_direct_movable[1]"
# page scan rate in pages / timer tick
pgscanrate="(pgscan_d - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
# cache usage in kilobytes
avail_cache="meminfo.Cached - meminfo.Shmem"

user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"
user_2="(cpustat.user[2] - cpustat.user[3])"
nice_2="(cpustat.nice[2] - cpustat.nice[3])"
system_2="(cpustat.system[2] - cpustat.system[3])"
CP_Active0="(user_0 + nice_0 + system_0) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_Active2="(user_2 + nice_2 + system_2) / (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
idle_2="(cpustat.idle[2] - cpustat.idle[3])"
iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"
CP_idle0="(idle_0 + iowait_0) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle2="(idle_2 + iowait_2) / (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_idleAVG="(CP_idle0 + CP_idle2) / 2"

# More required variables

# cmm_inc: 10% of free memory, in 4K pages
CMM_INC="meminfo.MemFree / 40"
# cmm_dec: 10% of total memory, in 4K pages
CMM_DEC="meminfo.MemTotal / 40"

# Hotplug rules
HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"
MEMPLUG="pgscanrate > 20"
MEMUNPLUG="(meminfo.MemFree + avail_cache) > (meminfo.MemTotal / 10)"
```

Figure 109. Sample configuration file for Linux on z/VM with CPU and memory hotplug

Attention: The sample file of Figure 109 on page 614 illustrates the syntax of the configuration file. Useful rules might differ considerably, depending on the workload, resources, and requirements of the system for which they are designed.

After you install cpuplugd with the s390utils RPM, a commented sample configuration file is available at `/etc/cpuplugd`. This file is used by the cpuplugd service.

dasdfmt - Format a DASD

Use the **dasdfmt** command to low-level format ECKD-type direct access storage devices (DASD).

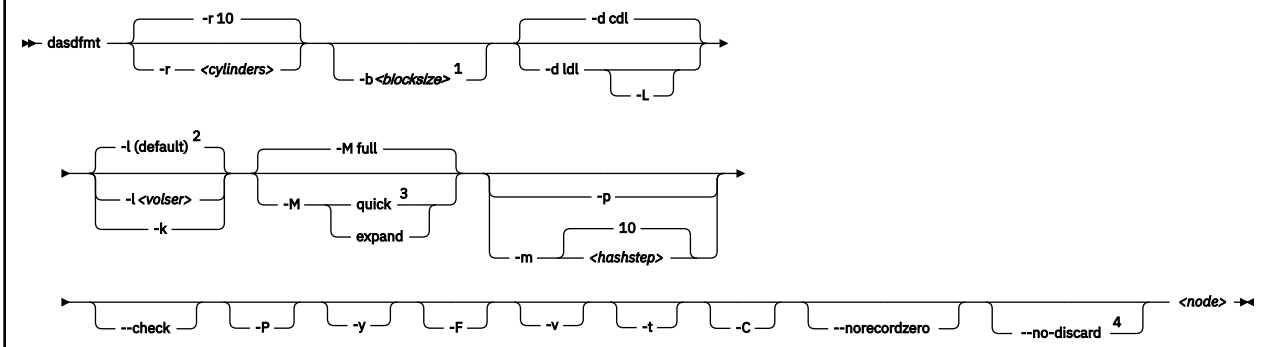
dasdfmt uses an ioctl call to the DASD driver to format tracks. A block size (hard sector size) can be specified. The formatting process can take quite a long time (hours for large DASD).

Tips:

- For DASDs that have previously been formatted with **dasdfmt**, use the **dasdfmt** quick format mode.
- Use the **-p** option to monitor the progress.

CAUTION: As on any platform, formatting irreversibly destroys data on the target disk. Be sure not to format a disk with vital data unintentionally.

dasdfmt syntax



Notes:

[1] You are prompted for the block size if you omit it for an option that requires this specification.
[2] If neither the **-l** option nor the **-k** option are specified, a VOLSER is generated from the device number through which the volume is accessed.
[3] quick is the default for ESE DASDs
[4] For ESE DASDs

Where:

-r *<cylinders>* or --requestsize=*<cylinders>*
specifies the number of cylinders to be processed in one formatting step. The value must be an integer in the range 1 - 255. The default is 10 cylinders. This parameter exploits any available PAV devices. Ideally, the number of cylinders matches the number of associated devices, counting the base device and all alias devices.

-b *<block_size>* or --blocksize=*<block_size>*
specifies one of the following block sizes in bytes: 512, 1024, 2048, or 4096.

For the **quick** and **expand** modes and for the **--check** option, you can omit the block size. Otherwise, you are prompted if you do not specify a value for the block size. You can then press Enter to accept 4096 or specify a different value.

Tip: Set *<block_size>* as large as possible (ideally 4096); the net capacity of an ECKD DASD decreases for smaller block sizes. For example, a DASD formatted with a block size of 512 byte has only half of the net capacity of the same DASD formatted with a block size of 4096 byte.

<node>
specifies the device node of the device to be formatted, for example, `/dev/dasdzzz`. See "DASD naming scheme" on page 142 for more details about device nodes).

-d *<disklayout>* or --disk_layout=*<disklayout>*
formats the device with the compatible disk layout (`cdl`) or the Linux disk layout (`ldl`). If the parameter is not specified, the default (`cdl`) is used.

-L or --no_label
valid for `-d ldl` only, where it suppresses the default LNX1 label.

-l *<volser>* or --label=*<volser>*
specifies the volume serial number (see VOLSER) to be written to the disk. If the VOLSER contains special characters, it must be enclosed in single quotation marks. In addition, any '$' character in the VOLSER must be preceded by a backslash ('\').

-k or --keep_volser
keeps the volume serial number when writing the volume label (see VOLSER). Keeping the volume serial number is useful, for example, if the volume serial number was written with a z/VM tool and should not be overwritten.

-M or --mode=*<mode>*
specifies the mode to be used for formatting the device. Valid modes are:

full
Format the entire disk with the specified block size. This is the default mode.

If you specify `mode=full` for an ESE DASD, the entire DASD is formatted as for a non-ESE DASD. The entire available capacity is occupied, and the `space_allocated` is the same as the `logical_capacity`.

quick
formats the first two tracks and writes label and partition information. Only use this option if you are sure that the target DASD already contains a regular format with the specified block size.

For ESE DASD, this is the default mode, see “Formatting ESE DASD” on page 169.

expand
format all unformatted tracks at the end of the target DASD. This mode assumes that tracks at the beginning of the DASD volume have already been correctly formatted, while a consecutive set of tracks at the end are unformatted. You can use this mode to make added space available for Linux use after dynamically increasing the size of a DASD volume.

For the **quick** and **expand** modes, omit the block size specification (**-b** option) to use the existing block size. If you specify a block size, **dasdfmt** checks that the specification matches the existing block size before formatting.

-p or --progressbar
displays a progress bar. Do not use this option if you are using a line-mode terminal console driver. For example, if you are using a 3215 terminal device driver or a line-mode hardware console device driver.

-P or --percentage
displays one line for each formatted cylinder. The line shows the number of the cylinder and percentage of formatting process. Intended for use by higher level interfaces.

-m *<hashstep>* or --hashmarks=*<hashstep>*
displays a number sign (#) after every *<hashstep>* cylinders are formatted. *<hashstep>* must be in the range 1 - 1000. The default is 10.

The **-m** option is useful where the console device driver is not suitable for the progress bar (**-p** option).

--check
performs a complete format check on a DASD volume.

Omit the block size specification (**-b** option) to check for a consistent format for any valid block size. Specify a block size to confirm that the DASD has been formatted consistently with that particular block size.

-y
starts formatting immediately without prompting for confirmation.

-F or --force
formats the device without checking whether it is mounted.

--no-discard
For ESE DASDs formatted with quick mode, prevents storage from being discarded when formatting.

-v
displays extra information messages (verbose).

-t or --test
runs the command in test mode. Analyzes parameters and displays what would happen, but does not modify the disk.

-C or --check_host_count
checks the host-access open count to ensure that the device is not online to another operating system instance. Use this option to ensure that the operation is safe, and cancel it if other operating system instances are accessing the volume.

-- norecordzero
prevents a format write of record zero. This option is intended for experts: Subsystems in DASD drivers are by default granted permission to modify or add a standard record zero to each track when needed. Before you revoke the permission with this option, you must ensure that the device contains standard record zeros on all tracks.

-V or --version
displays the version number of **dasdfmt** and exits.

-h or --help
displays an overview of the syntax. Any other parameters are ignored. To view the man page, enter **man dasdfmt**.

Examples

- To format a 100 cylinder z/VM minidisk with the standard Linux disk layout and a 4 KB blocksize with device node /dev/dasdc:

```
# dasdfmt -b 4096 -d ldl -p /dev/dasdc
Drive Geometry: 100 Cylinders * 15 Heads =  1500 Tracks
Device Type: Fully Provisioned

I am going to format the device /dev/dasdc in the following way:
   Device number of device : 0x192
   Labelling device        : yes
   Disk label              : LNX1
   Disk identifier         : 0X0192
   Extent start (trk no)   : 0
   Extent end (trk no)     : 1499
   Compatible Disk Layout  : no
   Blocksize               : 4096
   Mode                    : Full

--->> ATTENTION! <<---
All data of that device will be lost.
Type yes to continue, no will leave the disk untouched: yes
Formatting the device. This may take a while (get yourself a coffee).

cyl   100 of   100 |##############################################|100% [1s]

Finished formatting the device.
Rereading the partition table... ok
#
```

- To format the same disk with the compatible disk layout (accepting the default value of the **-d** option).

```
# dasdfmt -b 4096 -p /dev/dasdc
Drive Geometry: 100 Cylinders * 15 Heads =  1500 Tracks
Device Type: Fully Provisioned

I am going to format the device /dev/dasdc in the following way:
   Device number of device : 0x192
   Labelling device        : yes
   Disk label              : VOL1
   Disk identifier         : 0X0192
   Extent start (trk no)   : 0
   Extent end (trk no)     : 1499
   Compatible Disk Layout  : yes
   Blocksize               : 4096

--->> ATTENTION! <<---
All data of that device will be lost.
Type yes to continue, no will leave the disk untouched: yes
Formatting the device. This may take a while (get yourself a coffee).

cyl   100 of   100 |####################
##########################
#########| 100%

Finished formatting the device.
Rereading the partition table... ok
#
```

- To format using the **-P** option:

```
# dasdfmt -P /dev/dasde

cyl     1 of   500 |    0%
cyl     2 of   500 |    0%
cyl     3 of   500 |    0%
cyl     4 of   500 |    0%
cyl     5 of   500 |    1%

...
cyl   496 of   500 |   99%
cyl   497 of   500 |   99%
cyl   498 of   500 |   99%
cyl   499 of   500 |   99%
cyl   500 of   500 |  100%
```

- To make best use of PAV when formatting a DASD that has one base device and four alias devices, specify five cylinders:

```
# dasdfmt /dev/dasdd -y -b 4096 -d cdl -r 5
Finished formatting the device.
Rereading the partition table... ok
```

- To format a previously formatted DASD in quick format mode.

```
# dasdfmt -b 4096 -p --mode=quick /dev/dasdf
```

- To format tracks that have been added at the end of an already formatted DASD.

```
# dasdfmt -b 4096 -p --mode=expand /dev/dasdg
```

- To check whether a DASD has been correctly formatted with a block size of 4096 bytes.

```
# dasdfmt -b 4096 -p --check /dev/dasdg
Checking format of the entire disk...
cyl    1113 of    1113 |##################################|100% [19s]
Done. Disk is fine.
```

- To ensure that the DASD is not online to an operating system instance in a different LPAR when you start formatting the DASD:

```
# dasdfmt -b 4096 -p -C /dev/dasdh
```

dasdfmt always checks the host-access open count. If the count indicates access by another operating system instance, the response depends on the **-C** option. With this option, the command is canceled. Otherwise, a warning is displayed before you are prompted to confirm that you want to proceed.

- To format a thinly provisioned DASD in quick format mode:

```
# dasdfmt /dev/dasdc
Drive Geometry: 262668 Cylinders * 15 Heads =  3940020 Tracks
Device Type: Thinly Provisioned

I am going to format the device /dev/dasdc in the following way:
   Device number of device : 0x95d1
   Labelling device        : yes
   Disk label              : VOL1
   Disk identifier         : 0X95D1
   Extent start (trk no)   : 0
   Extent end (trk no)     : 1
   Compatible Disk Layout  : yes
   Blocksize               : 4096
   Mode                    : Quick
   Full Space Release      : yes

--->> ATTENTION! <<---
All data of that device will be lost.
Type "yes" to continue, no will leave the disk untouched: yes
Releasing space for the entire device...
Skipping format check due to thin-provisioned device.
Formatting the first two tracks of the device.
Finished formatting the device.
Rereading the partition table...
```

The quick format mode is the default for thinly provisioned DASDs. You can override the mode, for example, with `mode=full`, which formats the entire DASD as usual. All storage allocated for the DASD on the storage server is discarded. Use the `--no-discard` option to keep the storage allocated:

```
#  dasdfmt --no-discard /dev/dasdc
Drive Geometry: 262668 Cylinders * 15 Heads =  3940020 Tracks
Device Type: Thinly Provisioned

I am going to format the device /dev/dasdc in the following way:
   Device number of device : 0x95d1
   Labelling device        : yes
   Disk label              : VOL1
   Disk identifier         : 0X95D1
   Extent start (trk no)   : 0
   Extent end (trk no)     : 1
   Compatible Disk Layout  : yes
   Blocksize               : 4096
   Mode                    : Quick
   Full Space Release      : no

--->> ATTENTION! <<---
All data of that device will be lost.
Type "yes" to continue, no will leave the disk untouched: yes
Skipping format check due to thin-provisioned device.
Formatting the first two tracks of the device.
Finished formatting the device.
Rereading the partition table...
```

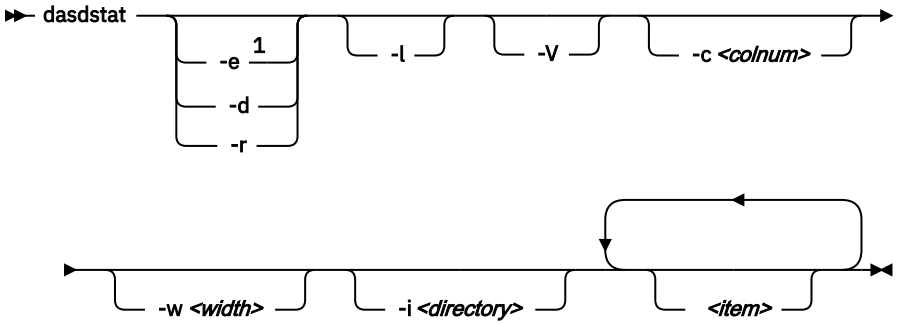
dasdstat - Display DASD performance statistics

Use the **dasdstat** command to display DASD performance statistics, including statistics about Parallel Access Volume (PAV) and High Performance Ficon.

This command includes and extends the performance statistics that is also available through the **tunedasd** command.

Before you begin: debugfs must be mounted, see “debugfs” on page ix.

dasdstat syntax



Notes:

[1] Omit the **-e**, **-d**, and **-r** options to read statistics.

Where:

-e or --enable
starts statistics data collection.

-d or --disable
stops statistics data collection.

-r or --reset
sets the statistics counters to zero.

-l or --long
displays more detailed statistics information, for example, differentiates between read and write requests.

-V or --verbose
displays more verbose command information.

-c *<colnum>* or --columns *<colnum>*
formats the command output in a table with the specified number of columns. The default is 16. Each row gets wrapped after the specified number of lines.

-w *<width>* or --column-width *<width>*
sets the minimum width, in characters, of a column in the output table.

-i *<directory>* or --directory *<directory>*
specifies the directory that contains the statistics. The default is *<mountpoint>*/dasd, where *<mountpoint>* is the mount point of debugfs. You need to specify this parameter if the **dasdstat** command cannot determine this mount point or if the statistics are copied to another location.

<item>
limits the command to the specified items. For *<item>* you can specify:

- `global` for summary statistics for all available DASDs.
- The block device name by which a DASD is known to the DASD device driver.

- The bus ID by which a DASD is known as a CCW device. DASDs that are set up for PAV or HyperPAV have a CCW base device and, at any one time, can have one or more CCW alias devices for the same block device. Alias devices are not permanently associated with the same block device. Statistics that are based on bus ID, therefore, show additional detail for PAV and HyperPAV setups.

If you do not specify any individual item, the command applies to all DASD block devices, CCW devices, and to the summary.

-v or --version
displays the version number of **dasdstat**, then exits.

-h or --help
displays help information for the command.

Examples

- This command starts data collection for dasda, 0.0.b301, and for a summary of all available DASDs.

```
# dasdstat -e dasda 0.0.b301 0.0.b302 global
```

- This command resets the statistics counters for dasda.

```
# dasdstat -r dasda
```

- This command reads the summary statistics:

```
statistics data for statistic: global
start time of data collection: Wed Aug 17 09:52:47 CEST 2011

3508 dasd I/O requests
with 67616 sectors(512B each)
0 requests used a PAV alias device
3458 requests used HPF
 __<4 ___8 __16 __32 __64 _128 _256 _512 __1k __2k __4k __8k _16k _32k _64k 128k
 _256 _512 __1M __2M __4M __8M _16M _32M _64M 128M 256M 512M __1G __2G __4G _>4G
Histogram of sizes (512B secs)
    0    0 2456  603  304  107   18    9    3    8    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
Histogram of I/O times (microseconds)
    0    0    0    0    0    0  100 1738  813  725   30   39   47   15    1    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
Histogram of I/O time till ssch
    0    0  901  558  765   25   28  288  748  161   17   16    1    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
Histogram of I/O time between ssch and irq
    0    0    0    0    0    0  316 2798  283   13   19   22   41   15    1    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
Histogram of I/O time between irq and end
    0 3023  460    8    4    9    4    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
# of req in chanq at enqueuing (0..31)
 ___0 ___1 ___2 ___3 ___4 ___5 ___6 ___7 ___8 ___9 __10 __11 __12 __13 __14 __15
 __16 __17 __18 __19 __20 __21 __22 __23 __24 __25 __26 __27 __28 __29 __30 __31
    0 2295  319  247  647    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

For details about the data items, see “Interpreting the data rows” on page 160.

dasdview - Display DASD structure

Use the **dasdview** command to display DASD information.

dasdview displays:

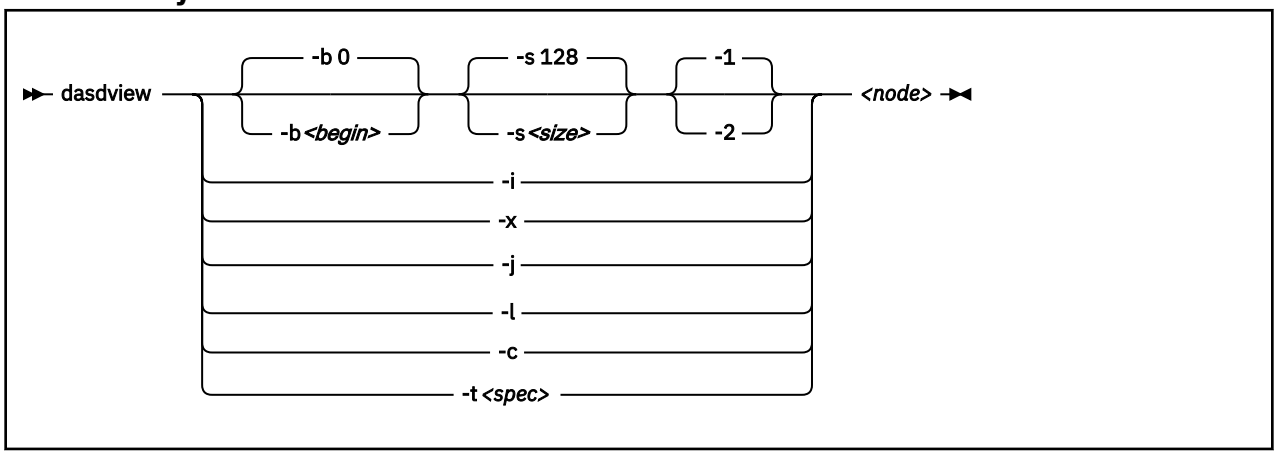
- The volume label.
- VTOC details (general information, and the DSCBs of format 1, format 3, format 4, format 5, format 7, format 8, and format 9).
- The content of the DASD, by specifying:
  - Starting point
  - Size

  You can display these values in hexadecimal, EBCDIC, and ASCII format.
- Whether the data on the DASD is encrypted.
- Whether the disk is a solid-state device.

If you specify a start point and size, you can also display the contents of a disk dump.

For more information about partitioning, see "The IBM label partitioning scheme" on page 138.

dasdview syntax



Where:

-b *<begin>* or --begin=*<begin>*
displays disk content on the console, starting from *<begin>*. The contents of the disk are displayed as hexadecimal numbers, ASCII text, and EBCDIC text. If *<size>* is not specified, **dasdview** takes the default size (128 bytes). You can specify the variable *<begin>* as:

```
<begin>[k|m|b|t|c]
```

If the disk is in raw-track access mode, you can specify only track (t) or cylinder (c) entities.

The default for *<begin>* is 0.

dasdview displays a disk dump on the console with the DASD driver. The DASD driver might suppress parts of the disk, or add information that is not relevant. This discrepancy might occur, for example, when **dasdview** displays the first two tracks of a disk that was formatted with compatible disk layout option (`-d cdl`. In this situation, the DASD driver pads shorter blocks with zeros to maintain a constant blocksize. All Linux applications (including **dasdview**) process according to this rule.

Here are some examples of how this option can be used:

```
-b 32   (start printing at Byte 32)
-b 32k  (start printing at kByte 32)
```

```
  -b 32m  (start printing at MByte 32)
  -b 32b  (start printing at block 32)
  -b 32t  (start printing at track 32)
  -b 32c  (start printing at cylinder 32)
```

-s *<size>* or --size=*<size>*
displays a disk dump on the console, starting at *<begin>*, and continuing for `size=`*<size>*. The contents of the dump are displayed as hexadecimal numbers, ASCII text, and EBCDIC text. If a start value, *<begin>*, is not specified, **dasdview** takes the default. You can specify the variable *<size>* as:

```
size[k|m|b|t|c]
```

If the disk is in raw-track access mode, you can specify only track (t) or cylinder (c) entities.

The default for *<size>* is 128 bytes.

Here are some examples of how this option can be used:

```
  -s 16   (use a 16 Byte size)
  -s 16k  (use a 16 kByte size)
  -s 16m  (use a 16 MByte size)
  -s 16b  (use a 16 block size)
  -s 16t  (use a 16 track size)
  -s 16c  (use a 16 cylinder size)
```

-1
displays the disk dump with format 1 (as 16 Bytes per line in hexadecimal, ASCII and EBCDIC). A line number is not displayed. You can use option **-1** only together with **-b** or **-s**.

Option **-1** is the default.

-2
displays the disk dump with format 2 (as 8 Bytes per line in hexadecimal, ASCII and EBCDIC). A decimal and hexadecimal byte count are also displayed. You can use option **-2** only together with **-b** or **-s**.

-i or --info
displays basic information such as device node, device bus-ID, device type, or geometry data.

-x or --extended
displays the information that is obtained by using **-i** option, but also open count, subchannel identifier, and so on.

-j or --volser
prints volume serial number (volume identifier).

-l or --label
displays the volume label.

-c or --characteristics
displays model-dependent device characteristics, for example disk encryption status or whether the disk is a solid-state device.

-t *<spec>* or --vtoc=*<spec>*
displays the VTOC's table-of-contents, or a single VTOC entry, on the console. The variable *<spec>* can take these values:

info
displays overview information about the VTOC, such as a list of the data set names and their sizes.

f1
displays the contents of all *format 1* data set control blocks (DSCBs).

f3
displays the contents of all (z/OS-specific) *format 3* DSCBs.

f4
displays the contents of all *format 4* DSCBs.

f5
displays the contents of all *format 5* DSCBs.

f7
displays the contents of all *format 7* DSCBs.

f8
displays the contents of all *format 8* DSCBs.

f9
displays the contents of all *format 9* DSCBs.

all
displays the contents of *all* DSCBs.

<node>
specifies the device node of the device for which you want to display information, for example, `/dev/dasdzzz`. See "DASD naming scheme" on page 142 for more details about device nodes).

-h or --help
displays short usage text on console. To view the man page, enter **man dasdview**.

-v or --version
displays version number on console, and exit.

Examples

- To display basic information about a DASD:

```
# dasdview -i /dev/dasdzzz
```

This example displays:

```
--- general DASD information ----------------------------------------------
device node             : /dev/dasdzzz
busid                   : 0.0.0193
type                    : ECKD
device type             : hex 3390       dec 13200

--- DASD geometry ---------------------------------------------------------
number of cylinders     : hex 64         dec 100
tracks per cylinder     : hex f          dec 15
blocks per track        : hex c          dec 12
blocksize               : hex 1000       dec 4096
#
```

- To display device characteristics:

```
# dasdview -c /dev/dasda
```

This example displays:

```
encrypted disk          : no
solid state device      : no
```

- To include extended information:

```
# dasdview -x /dev/dasdzzz
```

This example displays:

```
--- general DASD information ----------------------------------------------
device node            : /dev/dasdzzz
busid                  : 0.0.0193
type                   : ECKD
device type            : hex 3390       dec 13200

--- DASD geometry ---------------------------------------------------------
number of cylinders    : hex 64         dec 100
tracks per cylinder    : hex f          dec 15
blocks per track       : hex c          dec 12
blocksize              : hex 1000       dec 4096

--- extended DASD information ----------------------------------------------
real device number     : hex 452bc08    dec 72530952
subchannel identifier  : hex e          dec 14
CU type  (SenseID)     : hex 3990       dec 14736
CU model (SenseID)     : hex e9         dec 233
device type  (SenseID) : hex 3390       dec 13200
device model (SenseID) : hex a          dec 10
open count             : hex 1          dec 1
req_queue_len          : hex 0          dec 0
chanq_len              : hex 0          dec 0
status                 : hex 5          dec 5
label_block            : hex 2          dec 2
FBA_layout             : hex 0          dec 0
characteristics_size   : hex 40         dec 64
confdata_size          : hex 100        dec 256

characteristics        : 3990e933 900a5f80  dff72024 0064000f
                         e000e5a2 05940222  13090674 00000000
                         00000000 00000000  24241502 dfee0001
                         0677080f 007f4a00  1b350000 00000000

configuration_data     : dc010100 4040f2f1  f0f54040 40c9c2d4
                         f1f3f0f0 f0f0f0f0  f0c6c3f1 f1f30509
                         dc000000 4040f2f1  f0f54040 40c9c2d4
                         f1f3f0f0 f0f0f0f0  f0c6c3f1 f1f30500
                         d4020000 4040f2f1  f0f5c5f2 f0c9c2d4
                         f1f3f0f0 f0f0f0f0  f0c6c3f1 f1f3050a
                         f0000001 4040f2f1  f0f54040 40c9c2d4
                         f1f3f0f0 f0f0f0f0  f0c6c3f1 f1f30500
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         800000a1 00001e00  51400009 0909a188
                         0140c009 7cb7efb7  00000000 00000800
#
```

- To display volume label information:

```
# dasdview -l /dev/dasdzzz
```

This displays:

```
--- volume label ------------------------------------------------------------
volume label key        : ascii  'åÖÖñ'
                        : ebcdic 'VOL1'
                        : hex    e5d6d3f1

volume label identifier : ascii  'åÖÖñ'
                        : ebcdic 'VOL1'
                        : hex    e5d6d3f1

volume identifier       : ascii  'ðçðñùó'
                        : ebcdic '0X0193'
                        : hex    f0e7f0f1f9f3

security byte           : hex    40


VTOC pointer            : hex    0000000101
                                 (cyl 0, trk 1, blk 1)

reserved                : ascii  '@@@@@'
                        : ebcdic '     '
                        : hex    4040404040

CI size for FBA         : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

blocks per CI (FBA)     : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

labels per CI (FBA)     : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

reserved                : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

owner code for VTOC     : ascii  '@@@@@@@@@@@@@@'
                          ebcdic '              '
                          hex    40404040 40404040  40404040 4040

reserved                : ascii  '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'
                          ebcdic '                                '
                          hex    40404040 40404040  40404040 40404040
                                 40404040 40404040  40404040 40
#
```

- To display partition information:

```
# dasdview -t info /dev/dasdzzz
```

This example displays:

```
--- VTOC info ------------------------------------------------------------
The VTOC contains:
  3 format 1 label(s)
  1 format 4 label(s)
  1 format 5 label(s)
  0 format 7 label(s)
  0 format 8 label(s)
  0 format 9 label(s)
Other S/390 and zSeries operating systems would see the following data sets:
 +-------------------------------------------+-------------+-------------+
 | data set                                  | start       | end         |
 +-------------------------------------------+-------------+-------------+
 | LINUX.V0X0193.PART0001.NATIVE             |         trk |         trk |
 | data set serial number : '0X0193'         |           2 |         500 |
 | system code            : 'IBM LINUX    '  |     cyl/trk |     cyl/trk |
 | creation date          :  year 2001, day 317 |       0/  2 |      33/  5 |
 +-------------------------------------------+-------------+-------------+
 | LINUX.V0X0193.PART0002.NATIVE             |         trk |         trk |
 | data set serial number : '0X0193'         |         501 |         900 |
 | system code            : 'IBM LINUX    '  |     cyl/trk |     cyl/trk |
 | creation date          :  year 2001, day 317 |      33/  6 |      60/  0 |
 +-------------------------------------------+-------------+-------------+
 | LINUX.V0X0193.PART0003.NATIVE             |         trk |         trk |
 | data set serial number : '0X0193'         |         901 |        1499 |
 | system code            : 'IBM LINUX    '  |     cyl/trk |     cyl/trk |
 | creation date          :  year 2001, day 317 |      60/  1 |      99/ 14 |
 +-------------------------------------------+-------------+-------------+
#
```

- To display VTOC information:

```
# dasdview -t f4 /dev/dasdzzz
```

  This example displays:

```
--- VTOC format 4 label --------------------------------------------------
DS4KEYCD    : 04040404040404040404040404040404040404040404040404040404040404...
DS4IDFMT    : dec 244, hex f4
DS4HPCHR    : 0000000105 (cyl 0, trk 1, blk 5)
DS4DSREC    : dec 7, hex 0007
DS4HCCHH    : 00000000 (cyl 0, trk 0)
DS4NOATK    : dec 0, hex 0000
DS4VTOCI    : dec 0, hex 00
DS4NOEXT    : dec 1, hex 01
DS4SMSFG    : dec 0, hex 00
DS4DEVAC    : dec 0, hex 00
DS4DSCYL    : dec 100, hex 0064
DS4DSTRK    : dec 15, hex 000f
DS4DEVTK    : dec 58786, hex e5a2
DS4DEVI     : dec 0, hex 00
DS4DEVL     : dec 0, hex 00
DS4DEVK     : dec 0, hex 00
DS4DEVFG    : dec 48, hex 30
DS4DEVTL    : dec 0, hex 0000
DS4DEVDT    : dec 12, hex 0c
DS4DEVDB    : dec 0, hex 00
DS4AMTIM    : hex 0000000000000000
DS4AMCAT    : hex 000000
DS4R2TIM    : hex 0000000000000000
res1        : hex 0000000000
DS4F6PTR    : hex 0000000000
DS4VTOCE    : hex 0100000000100000001
              typeind    : dec 1, hex 01
              seqno      : dec 0, hex 00
              llimit     : hex 00000001 (cyl 0, trk 1)
              ulimit     : hex 00000001 (cyl 0, trk 1)
res2        : hex 00000000000000000000
DS4EFLVL    : dec 0, hex 00
DS4EFPTR    : hex 0000000000 (cyl 0, trk 0, blk 0)
res3        : hex 000000000000000000
#
```

- To print the contents of a disk to the console starting at block 2 (volume label):

```
# dasdview -b 2b -s 128 /dev/dasdzzz
```

This example displays:

```
+-----------------------------------+-----------------+-----------------+
| HEXADECIMAL                       | EBCDIC          | ASCII           |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+-----------------------------------+-----------------+-----------------+
|  E5D6D3F1 E5D6D3F1  F0E7F0F1 F9F34000  | VOL1VOL10X0193?. | ?????????????@. |
|  00000101 40404040  40404040 40404040  | ................ | ................ |
|  40404040 40404040  40404040 40404040  | ???????????????? | @@@@@@@@@@@@@@@@ |
|  40404040 40404040  40404040 40404040  | ???????????????? | @@@@@@@@@@@@@@@@ |
|  40404040 40404040  40404040 40404040  | ???????????????? | @@@@@@@@@@@@@@@@ |
|  40404040 88001000  10000000 00808000  | ????h........... | @@@@?........... |
|  00000000 00000000  00010000 00000200  | ................ | ................ |
|  21000500 00000000  00000000 00000000  | ?............... | !............... |
+-----------------------------------+-----------------+-----------------+
#
```

- To display the contents of a disk on the console starting at block 14 (first FMT1 DSCB) with format 2:

```
# dasdview -b 14b -s 128 -2 /dev/dasdzzz
```

This example displays:

```
 +---------------+---------------+---------------------+----------+----------+
 |     BYTE      |     BYTE      |     HEXADECIMAL     |  EBCDIC  |  ASCII   |
 |    DECIMAL    |  HEXADECIMAL  |  1 2 3 4   5 6 7 8  | 12345678 | 12345678 |
 +---------------+---------------+---------------------+----------+----------+
 |         57344 |          E000 |  D3C9D5E4  E74BE5F0  | LINUX.V0 | ?????K?? |
 |         57352 |          E008 |  E7F0F1F9  F34BD7C1  | X0193.PA | ?????K?? |
 |         57360 |          E010 |  D9E3F0F0  F0F14BD5  | RT0001.N | ??????K? |
 |         57368 |          E018 |  C1E3C9E5  C5404040  | ATIVE??? | ?????@@@ |
 |         57376 |          E020 |  40404040  40404040  | ???????? | @@@@@@@@ |
 |         57384 |          E028 |  40404040  F1F0E7F0  | ????10X0 | @@@@???? |
 |         57392 |          E030 |  F1F9F300  0165013D  | 193.???? | ???.?e?= |
 |         57400 |          E038 |  63016D01  0000C9C2  | ??_?..IB | c?m?..?? |
 |         57408 |          E040 |  D440D3C9  D5E4E740  | M?LINUX? | ?@?????@ |
 |         57416 |          E048 |  40404065  013D0000  | ??????.. | @@@e?=.. |
 |         57424 |          E050 |  00000000  88001000  | ....h.?. | ....?.?. |
 |         57432 |          E058 |  10000000  00808000  | ?....??. | ?....??. |
 |         57440 |          E060 |  00000000  00000000  | ........ | ........ |
 |         57448 |          E068 |  00010000  00000200  | .?....?. | .?....?. |
 |         57456 |          E070 |  21000500  00000000  | ?.?..... | !.?..... |
 |         57464 |          E078 |  00000000  00000000  | ........ | ........ |
 +---------------+---------------+---------------------+----------+----------+
#
```

- To see what is at block 1234 (in this example there is nothing there):

```
# dasdview -b 1234b -s 128 /dev/dasdzzz
```

This example displays:

```
+-----------------------------------+-----------------+-----------------+
| HEXADECIMAL                       | EBCDIC          | ASCII           |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+-----------------------------------+-----------------+-----------------+
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
+-----------------------------------+-----------------+-----------------+
#
```

- To try byte 0 instead:

```
# dasdview -b 0 -s 64 /dev/dasdzzz
```

This example displays:

```
+----------------------------------------+-----------------+-----------------+
| HEXADECIMAL                            | EBCDIC          | ASCII           |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+----------------------------------------+-----------------+-----------------+
|  C9D7D3F1 000A0000  0000000F 03000000  | IPL1............ | ????............ |
|  00000001 00000000  00000000 40404040  | ................ | ................ |
|  40404040 40404040  40404040 40404040  | ???????????????? | @@@@@@@@@@@@@@@@ |
|  40404040 40404040  40404040 40404040  | ???????????????? | @@@@@@@@@@@@@@@@ |
+----------------------------------------+-----------------+-----------------+
#
```

- To display the contents of a disk on the console starting at cylinder 2 and printing one track of data:

```
# dasdview -b 2c -s 1t /dev/dasdk
```

This example displays:

```
+----------------------------------------+-----------------+-----------------+
| HEXADECIMAL                            | EBCDIC          | ASCII           |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+----------------------------------------+-----------------+-----------------+
|  52B7DBEE D6B9530B  0179F420 CB6EA95E  | ????O?????4??>z; | R?????S??y???n?^ |
|  EF49C03C 513542E7  D8F17D9D 06DC44F7  | ??{????XQ1'????7 | ?I?<Q5B???}???D? |

...
|  92963D5B 0200B0FA  53745C12 C3B45125  | ko?$?........... | ??=[?........... |
|  0D6040C2 F933381E  7A4C4797 F40FEDAB  | ?-?B9???:<?p4??? | ??@??38?zLG????? |
...
```

- To display the full record information of the same disk when it in raw-track access mode:

```
# dasdview -b 2c -s 1t /dev/dasdk
```

This example displays:

```
cylinder 2, head 0, record 0
+----------------------------------------------------------------------------+
| count area:                                                                |
|          hex: 0002000000000008                                             |
|     cylinder:                2                                             |
|         head:                0                                             |
|       record:                0                                             |
|   key length:                0                                             |
|  data length:                8                                             |
+----------------------------------------------------------------------------+
| key area:                                                                  |
| HEXADECIMAL                          | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+--------------------------------------+------------------+------------------+
+--------------------------------------+------------------+------------------+
| data area:                                                                 |
| HEXADECIMAL                          | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+--------------------------------------+------------------+------------------+
|  00000000 00000000                   | ................ | ................ |
+--------------------------------------+------------------+------------------+

cylinder 2, head 0, record 1
+----------------------------------------------------------------------------+
| count area:                                                                |
|          hex: 0002000001000200                                             |
|     cylinder:                2                                             |
|         head:                0                                             |
|       record:                1                                             |
|   key length:                0                                             |
|  data length:              512                                             |
+----------------------------------------------------------------------------+
| key area:                                                                  |
| HEXADECIMAL                          | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+--------------------------------------+------------------+------------------+
+--------------------------------------+------------------+------------------+
| data area:                                                                 |
| HEXADECIMAL                          | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+--------------------------------------+------------------+------------------+
|  52B7DBEE D6B9530B  0179F420 CB6EA95E  | ????O?????4??>z; | R?????S??y???n?^ |
|  EF49C03C 513542E7  D8F17D9D 06DC44F7  | ??{????XQ1'????7 | ?I?<Q5B???}???D? |
...
+--------------------------------------+------------------+------------------+

cylinder 2, head 0, record 2
+----------------------------------------------------------------------------+
| count area:                                                                |
|          hex: 0002000002000200                                             |
|     cylinder:                2                                             |
|         head:                0                                             |
|       record:                2                                             |
|   key length:                0                                             |
|  data length:              512                                             |
+----------------------------------------------------------------------------+
| key area:                                                                  |
| HEXADECIMAL                          | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+--------------------------------------+------------------+------------------+
+--------------------------------------+------------------+------------------+
| data area:                                                                 |
| HEXADECIMAL                          | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+--------------------------------------+------------------+------------------+
|  92963D5B 0200B0FA  53745C12 C3B45125  | ko?$?.^???*?C??? | ??=[?.??St\???Q% |
|  0D6040C2 F933381E  7A4C4797 F40FEDAB  | ?-?B9???:<?p4??? | ??@??38?zLG????? |
...
```

- To display the contents of a disk, which is in raw-access mode, printing one track of data from the start of the disk:

```
# dasdview -s 1t /dev/dasdk
```

This example displays:

```
cylinder 0, head 0, record 0
+-----------------------------------------------------------------------------+
| count area:                                                                 |
|          hex: 0000000000000008                                              |
|     cylinder:                0                                              |
|         head:                0                                              |
|       record:                0                                              |
|   key length:                0                                              |
|  data length:                8                                              |
+-----------------------------------------------------------------------------+
| key area:                                                                   |
| HEXADECIMAL                          | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+--------------------------------------+------------------+------------------+
+--------------------------------------+------------------+------------------+
| data area:                                                                  |
| HEXADECIMAL                          | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+--------------------------------------+------------------+------------------+
|  00000000 00000000                   | ................ | ................ |
+--------------------------------------+------------------+------------------+

cylinder 0, head 0, record 1
+-----------------------------------------------------------------------------+
| count area:                                                                 |
|          hex: 0000000001040018                                              |
|     cylinder:                0                                              |
|         head:                0                                              |
|       record:                1                                              |
|   key length:                4                                              |
|  data length:               24                                              |
+-----------------------------------------------------------------------------+
| key area:                                                                   |
| HEXADECIMAL                          | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+--------------------------------------+------------------+------------------+
|  C9D7D3F1                            | IPL1............ | ????............ |
+--------------------------------------+------------------+------------------+
| data area:                                                                  |
| HEXADECIMAL                          | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+--------------------------------------+------------------+------------------+
|  000A0000 0000000F  03000000 00000001  | .?.....??......? | .?.....??......? |
|  00000000 00000000                     | ................ | ................ |
+--------------------------------------+------------------+------------------+
...
```

fdasd – Partition a DASD

Use the **fdasd** command to manage partitions on ECKD-type DASD that were formatted with the compatible disk layout.

See “dasdfmt - Format a DASD” on page 615 for information about formatting a DASD. With **fdasd** you can create, change, and delete partitions, and also change the volume serial number.

fdasd checks that the volume has a valid volume label and VTOC. If either is missing or incorrect, **fdasd** re-creates it. See “IBM Z compatible disk layout” on page 139 for details about the volume label and VTOC.

Calling **fdasd** with a node, but without options, enters interactive mode. In interactive mode, you are given a menu through which you can display DASD information, add or remove partitions, or change the volume identifier. Your changes are not written to disk until you type the **write** option on the menu. You can quit without altering the disk at any time before this.

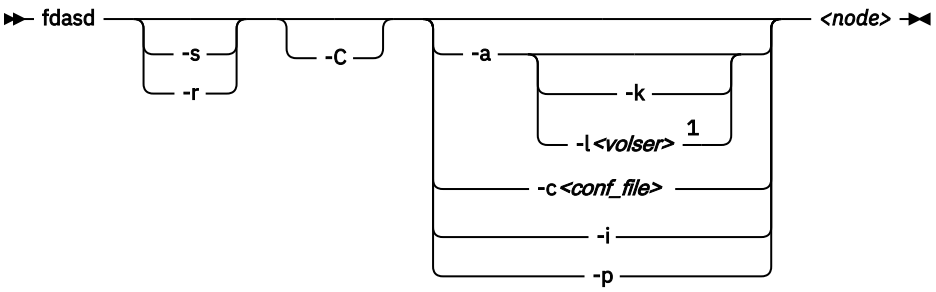
For more information about partitions, see “The IBM label partitioning scheme” on page 138.

Before you begin:

- To partition a SCSI disk, use **fdisk** rather than **fdasd**.
- The disk must be formatted with **dasdfmt**, using the compatible disk layout.

Attention: Careless use of **fdasd** can result in loss of data.

fdasd syntax



Notes:

[1] If neither the **-l** option nor the **-k** option is specified, a VOLSER is generated from the device number through which the volume is accessed.

Where:

-h or --help
displays help on command line arguments.

-v or --version
displays the version of **fdasd**.

-s or --silent
suppresses messages.

-r or --verbose
displays additional messages that are normally suppressed.

-a or --auto
auto-creates one partition using the whole disk in non-interactive mode.

-k or --keep_volser
keeps the volume serial number when writing the volume label (see VOLSER). Keeping the serial number is useful, for example, if the volume serial number was written with a z/VM tool and should not be overwritten.

-l *<volser>* or --label *<volser>*
specifies the volume serial number (see VOLSER).

A volume serial consists of one through six alphanumeric characters or the following special characters:

$ # @ %

All other characters are ignored. Avoid using special characters in the volume serial. Special characters can cause problems when accessing a disk by VOLSER. If you must use special characters, enclose the VOLSER in single quotation marks. In addition, any '$' character in the VOLSER must be preceded by a backslash ('\').

For example, specify:

```
-l 'a@b\$c#'
```

to get:

```
A@B$C#
```

VOLSER is interpreted as an ASCII string and is automatically converted to uppercase, padded with blanks and finally converted to EBCDIC before it is written to disk.

Do not use the following reserved volume serials:

- SCRTCH
- PRIVAT
- MIGRAT
- Lnnnnn (L followed by a five-digit number)

These volume serials are used as keywords by other operating systems (z/OS).

Omitting this parameter causes **fdasd** to prompt for it, if it is needed.

-c *<conf_file>* or --config *<conf_file>*
creates several partitions in non-interactive mode, according to specifications in the plain-text configuration file *<conf_file>*.

For each partition you want to create, add one line of the following format to *<conf_file>*:

```
[<first_track>,<last_track>,<type>]
```

<first_track> and *<last_track>* are required and specify the first and last track of the partition. You can use the keyword `first` for the first possible track on disk and, correspondingly, the keyword `last` for the last possible track on disk.

<type> describes the partition type and is one of:

native
for partitions to be used for Linux file systems.

gpfs
for partitions to be used as part of an Elastic Storage file system setup.

swap
for partitions to be used as swap devices.

raid
for partitions to be used as part of a RAID setup.

lvm
for partitions to be used as part of a logical volume group.

The type specification is optional. If the type is omitted, `native` is used.

The type describes the intended use of a partition to tools or other operating systems. For example, swap partitions could be skipped by backup programs. How Linux actually uses the partition depends on how the partition is formatted and set up. For example, a partition of type `native` can still be used in an LVM logical volume or in a RAID configuration.

Example: With the following sample configuration file you can create three partitions:

```
[first,1000,raid]
[1001,2000,swap]
[2001,last]
```

-i or --volser
displays the volume serial number and exits.

-p or --table
displays the partition table and exits.

<node>
specifies the device node of the DASD you want to partition, for example, `/dev/dasdzzz`. See “DASD naming scheme” on page 142 for more details about device nodes.

-C or --check_host_count
checks the host-access open count to ensure that the device is not online to another operating system instance. The operation is canceled if another operating system instance is accessing the device.

fdasd menu

If you call **fdasd** in the interactive mode (that is, with just a node), a menu is displayed.

```
Command action
   m print this menu
   p print the partition table
   n add a new partition
   d delete a partition
   v change volume serial
   t change partition type
   r re-create VTOC and delete all partitions
   u re-create VTOC re-using existing partition sizes
   s show mapping (partition number - data set name)
   q quit without saving changes
   w write table to disk and exit

Command (m for help):
```

Menu commands

Use the **fdasd** menu commands to modify or view information about DASDs

m
re-displays the **fdasd** command menu.

p
displays information about the DASD and any partitions on the DASD.

DASD information:

- Number of cylinders
- Number of tracks per cylinder
- Number of blocks per track
- Block size
- Volume label

- Volume identifier
- Number of partitions defined

Partition information:

- Linux node
- Start track
- End track
- Number of tracks
- Partition ID
- Partition type

There is also information about the free disk space that is not used for a partition.

n
adds a partition to the DASD. You are asked to give the start track and the length or end track of the new partition.

d
deletes a partition from the DASD. You are asked which partition to delete.

v
changes the volume identifier. You are asked to enter a new volume identifier. See VOLSER for the format.

t
changes the partition type. You are asked to identify the partition to be changed. You are then asked for the new partition type (Linux native or swap). This type is a guideline; the actual use Linux makes of the partition depends on how it is defined with the **mkswap** or mk*xx*fs tools. The main function of the partition type is to describe the partition to other operating systems. Then, for example, swap partitions can be skipped by backup programs.

r
re-creates the VTOC and deletes all partitions.

u
re-creates all VTOC labels without removing all partitions. Existing partition sizes are reused. This option is useful to repair damaged labels or migrate partitions that are created with older versions of **fdasd**.

s
displays the mapping of partition numbers to data set names. For example:

```
Command (m for help): s

device .........: /dev/dasdzzz
volume label ...: VOL1
volume serial ..: 0X0193

WARNING: This mapping may be NOT up-to-date,
         if you have NOT saved your last changes!

/dev/dasdzzz1  -  LINUX.V0X0193.PART0001.NATIVE
/dev/dasdzzz2  -  LINUX.V0X0193.PART0002.NATIVE
/dev/dasdzzz3  -  LINUX.V0X0193.PART0003.NATIVE
```

q
quits **fdasd** without updating the disk. Any changes that you made (in this session) are discarded.

w
writes your changes to disk and exits. After the data is written, Linux rereads the partition table.

Example using the menu

This example shows how to use **fdasd** to create two partitions on a z/VM minidisk, change the type of one of the partitions, save the changes, and check the results.

This example shows you how to format a z/VM minidisk with the compatible disk layout. The minidisk has device number 193.

1. Call **fdasd**, specifying the minidisk:

```
# fdasd /dev/dasdzzz
```

   fdasd reads the existing data and displays the menu:

```
reading volume label: VOL1
reading vtoc : ok

Command action
   m print this menu
   p print the partition table
   n add a new partition
   d delete a partition
   v change volume serial
   t change partition type
   r re-create VTOC and delete all partitions
   u re-create VTOC re-using existing partition sizes
   s show mapping (partition number - data set name)
   q quit without saving changes
   w write table to disk and exit
Command (m for help):
```

2. Use the p option to verify that no partitions are created yet on this DASD:

```
Command (m for help): p

Disk /dev/dasdzzz:
  cylinders .............: 100
  tracks per cylinder ..: 15
  blocks per track .....: 12
  bytes per block ......: 4096
  volume label .........: VOL1
  volume serial ........: 0X0193
  max partitions .......: 3

 ------------------------------ tracks -------------------------------
               Device      start      end   length   Id  System
                                2     1499     1498       unused
```

3. Define two partitions, one by specifying an end track and the other by specifying a length. (In both cases the default start tracks are used):

```
Command (m for help): n
First track (1 track = 48 KByte) ([2]-1499):
Using default value 2
Last track or +size[c|k|M] (2-[1499]): 700
You have selected track 700
```

```
Command (m for help): n
First track (1 track = 48 KByte) ([701]-1499):
Using default value 701
Last track or +size[c|k|M] (701-[1499]): +400
You have selected track 1100
```

4. Check the results by using the p option:

```
Command (m for help): p

Disk /dev/dasdzzz:
  cylinders .............: 100
  tracks per cylinder ..: 15
  blocks per track .....: 12
  bytes per block ......: 4096
  volume label .........: VOL1
  volume serial ........: 0X0193
  max partitions .......: 3

 ------------------------------ tracks -------------------------------
              Device      start      end   length   Id  System
       /dev/dasdzzz1          2      700      699    1  Linux native
       /dev/dasdzzz2        701     1100      400    2  Linux native
                           1101     1499      399       unused
```

5. Change the type of a partition:

```
Command (m for help): t

Disk /dev/dasdzzz:
  cylinders .............: 100
  tracks per cylinder ..: 15
  blocks per track .....: 12
  bytes per block ......: 4096
  volume label .........: VOL1
  volume serial ........: 0X0193
  max partitions .......: 3

 ------------------------------ tracks -------------------------------
              Device      start      end   length   Id  System
       /dev/dasdzzz1          2      700      699    1  Linux native
       /dev/dasdzzz2        701     1100      400    2  Linux native
                           1101     1499      399       unused

change partition type
partition id (use 0 to exit):
```

Enter the ID of the partition you want to change; in this example partition 2:

```
partition id (use 0 to exit): 2
```

6. Enter the new partition type; in this example type 2 for swap:

```
current partition type is: Linux native

    1 Linux native
    2 Linux swap

new partition type: 2
```

7. Check the result:

```
Command (m for help): p


Disk /dev/dasdzzz:
  cylinders .............: 100
  tracks per cylinder ..: 15
  blocks per track .....: 12
  bytes per block ......: 4096
  volume label .........: VOL1
  volume serial ........: 0X0193
  max partitions .......: 3

 ------------------------------ tracks -------------------------------
              Device      start      end   length   Id  System
       /dev/dasdzzz1          2      700      699    1  Linux native
       /dev/dasdzzz2        701     1100      400    2  Linux swap
                           1101     1499      399       unused
```

8. Write the results to disk with the `w` option:

```
Command (m for help): w
writing VTOC...
rereading partition table...
#
```

Example using options

You can partition a DASD by using the **-a** or **-c** option without entering the menu mode.

This method is useful for partitioning by using scripts, for example, if you need to partition several hundred DASDs.

With the **-a** option you can create one large partition on a DASD:

```
# fdasd -a /dev/dasdzzz
auto-creating one partition for the whole disk...
writing volume label...
writing VTOC...
rereading partition table...
#
```

This example creates a partition as follows:

```
        Device      start      end   length   Id  System
  /dev/dasdzzz1          2     1499     1498    1  Linux native
```

Using a configuration file that you can create several partitions. For example, the following configuration file, `config`, creates three partitions:

```
[first,500]
[501,1100]
[1101,last]
```

Submitting the command with the **-c** option creates the partitions:

```
# fdasd -c config /dev/dasdzzz
parsing config file 'config'...
writing volume label...
writing VTOC...
rereading partition table...
#
```

This example creates partitions as follows:

```
        Device      start      end   length   Id  System
  /dev/dasdzzz1          2      500      499    1  Linux native
  /dev/dasdzzz2        501     1100      600    2  Linux native
  /dev/dasdzzz3       1101     1499      399    3  Linux native
```

hmcdrvfs - Mount a FUSE file system for remote access to media in the HMC media drive

Use the **hmcdrvfs** command for read-only access to contents in a DVD, CD, or USB-attached storage in the media drive of an HMC.

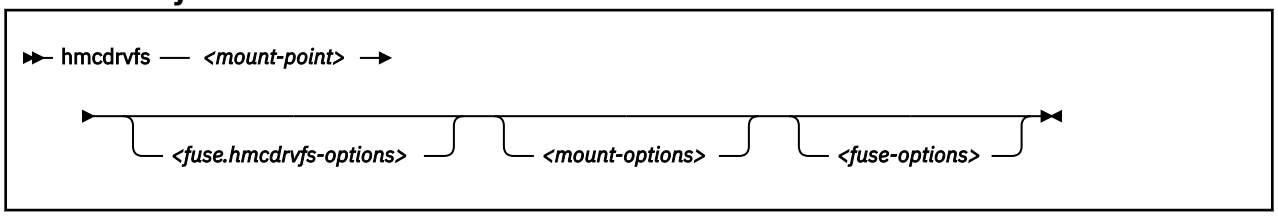
Before you begin:

- The fuse.hmcdrvfs file system needs access to device node `/dev/hmcdrv`. This node is created automatically when the hmcdrv kernel module is loaded, see Chapter 30, “HMC media device driver,” on page 383.
- On the HMC, the media must be assigned to the associated system image (use menu Access Removable Media).
- In a z/VM environment, the z/VM guest virtual machine must have at least privilege class B. The media must be assigned to the LPAR where the z/VM hypervisor runs.
- For Linux in LPAR mode, the LPAR activation profile must allow issuing SCLP requests.

With the media assigned to your Linux instance, this command creates a `fuse.hmcdrvfs` file system with the media content at the specified mount point.

To unmount file systems that you mounted with **hmcdrvfs**, you can use **fusermount**, whether root or non-root user. See the **fusermount** man page for details.

hmcdrvfs syntax



Where:

-o or --opt
FUSE or mount command options; for the FUSE options see the following lists, for mount options see the **mount** man page.

<fuse.hmcdrvfs-options>
options specific to the `fuse.hmcdrvfs` file system:

-o hmclang=*<language>*
specifies the language setting on the HMC; for valid values, see the **locale** man page.

-o hmctz=*<time zone>*
specifies the time zone setting on the HMC; for valid values, see the **tzset** man page.

<mount-options>
options as available for the **mount** command. See the **mount** man page for details.

<fuse-options>
options for FUSE. The following options are supported by the **cmsfs-fuse** command. To use an option, it must also be supported by the version of FUSE that you have.

-d or -o debug
enables debug output (implies **-f**).

-f
runs the command as a foreground operation.

-s
disables multi-threaded operation.

-o allow_other
allows access to the file system by other users.

-o allow_root
allows access to the file system by root.

-o default_permissions
enables permission checking by the kernel.

-o fsname=*<name>*
sets the file system name.

-o subtype=*<type>*
sets the file system type.

-o max_read=*<n>*
sets maximum size of read requests.

-o direct_io
uses direct I/O.

-o kernel_cache
caches files in the kernel.

-o [no]auto_cache
enables or disables caching based on modification times.

-o umask=*<mask>*
sets file permissions (octal).

-o uid=*<n>*
sets the file owner.

-o gid=*<n>*
sets the file group.

-o entry_timeout=*<secs>*
sets the cache timeout for names. The default is 1.0 second.

-o attr_timeout=*<secs>*
sets the cache timeout for attributes. The default is 1.0 second.

-o ac_attr_timeout=*<secs>*
sets the auto cache timeout for attributes. The default is the `attr_timeout` value.

-o max_readahead=*<n>*
sets the maximum read ahead value.

-o async_read
performs reads asynchronously (default).

-o sync_read
performs reads synchronously.

-o no_remote_lock
disables remote file locking.

-o intr
allows requests to be interrupted

-o intr_signal=*<num>*
specifies the signal to send on interrupt.

-v or --version
displays version information for the command.

-h or --help
displays a short help text, then exits. To view the man page, enter **`man hmcdrvfs`**.

The following options for mount policy can be set in the file `/etc/ fuse.conf` file:

mount_max=*<number>*
sets the maximum number of FUSE mounts allowed for non-root users. The default is 1000.

user_allow_other
allows non-root users to specify the `allow_other` or `allow_root` mount options.

Examples

- To mount the contents of the HMC media drive at `/mnt/hmc` without any special options, use:

```
# hmcdrvfs /mnt/hmc
```

- If the hmcdrv kernel module is not loaded, load it before you issue the **hmcdrvfs** command:

```
# modprobe hmcdrv
# hmcdrvfs /mnt/hmc
```

- To translate the UID and GID of files on the HMC media drive to your system users and groups along with overriding the permissions, issue, for example:

```
# hmcdrvfs /mnt/hmc -o uid=500 -o gid=1000 -o umask=0337
```

- To speed up transfer rates to frequently accessed directories, use the cache timeout option:

```
# hmcdrvfs /mnt/hmc -o entry_timeout=60
```

- If the HMC is in a different timezone and is configured for a different language use, for example:

```
# hmcdrvfs /mnt/hmc -o hmclang=de_DE -o hmctz=Europe/Berlin
```

- To also disregard any Daylight Saving Time, specifying hours west of the Prime Meridian (Coordinated Universal Time):

```
# hmcdrvfs /mnt/hmc -o hmclang=de_DE -o hmctz="GMT-1"
```

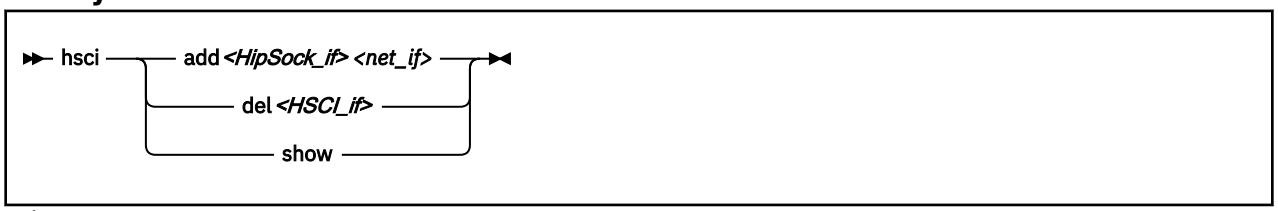
- To unmount the HMC media drive contents mounted on `/mnt/hmc`, issue:

```
# fusermount -u /mnt/hmc
```

hsci - Manage HSCI interfaces

Use the **hsci** command to add, delete, and list HiperSockets Converged Interface (HSCI) interfaces.

hsci syntax



Where:

add *<HipSock_if> <net_if>*
creates an HSCI interface by connecting a HiperSockets device with a network adapter.

del *<HSCI_if>*
deletes an HSCI interface by dissolving the connection between the HiperSockets interface and the network interface.

show
lists all HSCI interfaces.

Examples

- To create an HSCI interface by connecting a HiperSockets interface `encb112` and an OSA Express or RoCE device with a network interface enca100. Both are assigned the same PNET ID, and none have an IP address.

```
# hsci add encb112 enca100
Verifying net dev enca100 and HiperSockets dev encb112
Adding hscib112 with a HiperSockets dev encb112 and an external dev enca100
Set encb112 MAC fe:c2:f4:35:00:12 on enca100 and hscib112
Successfully added HSCI interface hscib112
```

  You can now assign an IP address to the new `hscib112` interface.

- To list available HSCI interfaces, issue:

```
# hsci show
HSCI      PNET_ID   HiperSockets   External
------------------------------------------------------------
hscib112  NET1      encb112        enca100
```

- To delete an HSCI interface, issue:

```
# hsci del hscib112
Deleting HSCI interface hscib112 with the HiperSockets encb112 and the
external enca100
Deleting MAC fe:c2:f4:35:00:12 on enca100
Successfully deleted device hscib112
```

hyptop - Display hypervisor performance data

Use the **hyptop** command to obtain a dynamic real-time view of a hypervisor environment on IBM Z.

It works with both the z/VM and the LPAR PR/SM hypervisor. Depending on the available data, it shows, for example, CPU and memory information about LPARs or z/VM guest virtual machines.

System names provided by hyptop are either LPAR names as shown on the SE or HMC, or z/VM guest IDs that identify z/VM guest virtual machines.

The **hyptop** command provides two main windows:

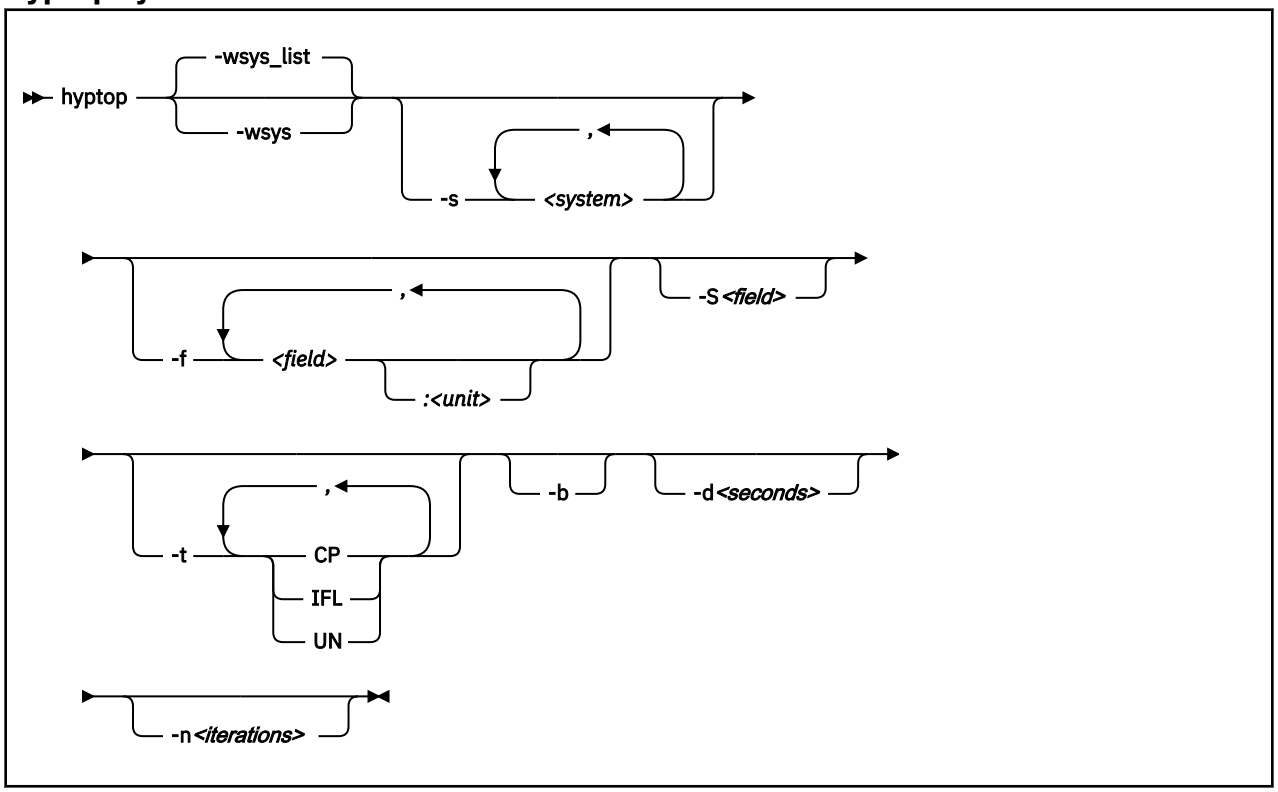
- A list of systems that the hypervisor is currently running (sys_list).
- One system in more detail (sys).

You can run **hyptop** in interactive mode (default) or in batch mode with the **-b** option.

Before you begin:

- The debugfs file system must be mounted, see “debugfs” on page ix.
- The Linux kernel must have the required support to provide the performance data. Check that *<debugfs mount point>*/s390_hypfs is available after you mount debugfs.
- The hyptop user must have read permission for the required debugfs files:
  - z/VM: *<debugfs mount point>*/s390_hypfs/diag_2fc
  - z/VM: *<debugfs mount point>*/s390_hypfs/diag_0c
    (Required only for management time data, identifiers `m` and `M`. See “z/VM fields” on page 647)
  - LPAR: *<debugfs mount point>*/s390_hypfs/diag_204
- You can always monitor the guest operating system where **hyptop** is running. To monitor any other operating system instances running on the same hypervisor as **hyptop**, you will need additional permissions:
  - For z/VM: The guest virtual machine must be assigned privilege class B.
  - For LPAR: On the HMC or SE security menu of the LPAR activation profile, select the **Global performance data control** check box.

hyptop syntax



Where:

-w *<window name>* or --window=*<window name>*
selects the window to display, either sys or sys_list. Use the options **--sys**, **--fields**, and **--sort** to modify the current window. The last window that is specified with the **--window** option is used as the start window. The default window is sys_list.

-s *<system>* or --sys=*<system>*
selects systems for the current window. If you specify this option, only the selected systems are shown in the window. For the sys window, you can specify only one system. *<system>* can be an LPAR name as shown on the SE or HMC, or it can be a z/VM guest ID that identifies a z/VM guest virtual machine. Enter **hyptop** without any options to display the names of all available systems.

-f *<field>*[:*<unit>*] or --fields=*<field>*[:*<unit>*]
selects fields and units in the current window. The *<field>* variable is a one letter unique identifier for a field (for example "c" for CPU time). The *<unit>* variable specifies the unit that is used for the field (for example "us" for microseconds). See"Available fields and units" on page 646 for definitions. If the **--fields** option is specified, only the selected fields are shown.

Note: If your field specification includes the number sign (#), enclose the specification in double quotation marks. Otherwise, the command shell might interpret the number sign and all characters that follow as a comment.

-S *<field>* or --sort=*<field>*
selects the field that is used to sort the data in the current window. To reverse the sort order, specify the option twice. See "Available fields and units" on page 646 for definitions.

-t *<type>* or --cpu_types=*<type>*
selects CPU types that are used for dispatch time calculations. See "CPU types" on page 648 for definitions.

-b or --batch_mode
uses batch mode. Batch mode can be useful for sending output from hyptop to another program, a file, or a line mode terminal. In this mode, no user input is accepted.

-d *<seconds>* or --delay=*<seconds>*
specifies the delay between screen updates.

-n *<iterations>* or --iterations=*<iterations>*
specifies the maximum number of screen updates before the program ends.

-h or --help
prints usage information, then exits. To view the man page, enter **man hyptop**.

-v or --version
displays the version of **hyptop**, then exits.

Navigating between windows

Use letter or arrow keys to navigate between the windows.

When you start the **hyptop** command, the sys_list window opens in normal mode. Data is updated at regular intervals, and sorted by dispatch time. You can navigate between the windows as shown in Figure 110 on page 645.

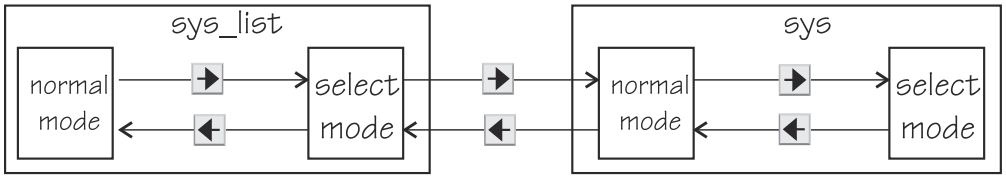


Figure 110. hyptop window navigation overview

To navigate between the windows, use the ← and → arrow keys. The windows have two modes, normal mode and select mode.

You can get online help for every window by pressing the ? key. Press Q in the sys_list window to exit hyptop.

Instead of using the arrow keys, you can use letter keys (equivalent to the vi editor navigation) in all windows as listed in Table 77 on page 645.

Table 77. Using letter keys instead of arrow keys

Arrow key	Letter key equivalent
←	H
↓	J
↑	K
→	L

Selecting data

You can scroll windows and select data rows.

To enter select mode press the → key. The display is frozen so that you can select rows. Select rows by pressing the ↑ and ↓ keys and mark the rows with the Spacebar. Marked rows are displayed in bold font. Leave the select mode by pressing the ← key.

To see the details of one system, enter select mode in the sys_list window, then navigate to the row for the system you want to look at, and press the → key. The sys window for the system opens. The ← key always returns you to the previous window.

To scroll any window, press the ↑ and ↓ keys or the Page Up and Page Down keys. Jump to the end of a window by pressing the Shift + G keys and to the beginning by pressing the G key.

Sorting data

You can sort data according to column.

The sys window or sys_list window table is sorted according to the values in the selected column. Select a column by pressing the hot key of the column. This key is underlined in the heading. If you press the hot key again, the sort order is reversed. Alternatively, you can select columns with the < and > keys.

Filtering data

You can filter the displayed data by CPU types and by data fields.

From the sys or sys_list window you can access the fields selection window and the CPU-type selection window as shown in Figure 111 on page 646.

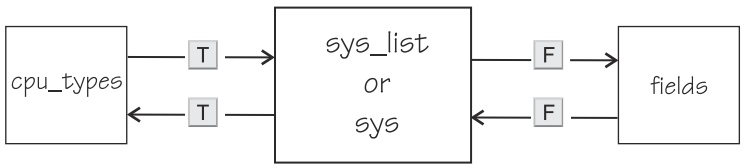


Figure 111. Accessing the fields and CPU-type selection windows

Use the T key to toggle between the CPU-type selection window and the main window. Use the F key to toggle between the fields selection window and the main window. You can also use the ← key to return to the main window from the CPU types and fields windows.

In the fields and CPU-type selection windows, press the field or CPU type identifier key (see "LPAR fields" on page 646, "z/VM fields" on page 647, and "CPU types" on page 648) to select or de-select. Selected rows are bold and de-selected rows are grey. When you return to the main window, the data is filtered according to your field and CPU type selections.

Available fields and units

Different fields are supported depending whether your hypervisor is LPAR PR/SM or z/VM.

The fields might also be different depending on machine type, z/VM version, and kernel version. Each field has a unique one letter identifier that can be used in interactive mode to enable the field in the field selection window. Also, use it to select the sort field in the sys or sys_list window. You can also select fields and sort data using the **`--fields`** and **`--sort`** command line options.

LPAR fields

Some fields for Linux in LPAR mode are available in both the sys_list and sys windows others are available only in the sys_list window or only in the sys window.

The following fields are available under LPAR in both the sys_list and sys windows:

Identifier	Column label	Explanation
c	core	Core dispatch time per second
e	the	Thread time per second
m	mgm	Management time per second
C	Core+	Total core dispatch time
E	thE+	Total thread time

Identifier	Column label	Explanation
M	Mgm+	Total management time
o	online	Online time

If multithreading is not available or not enabled, the values for core and for thread are identical.

In the sys_list window only:

Identifier	Column label	Explanation
y	system	Name of the LPAR (always shown)
#	#cpu	Number of CPUs
T	#The	Number of threads (sum of initial and reserved)

In the sys window only:

Identifier	Column label	Explanation
i	coreid	Core identifier (always shown)
p	type	CPU type. See “CPU types” on page 648
v	visual	Visualization of core dispatch time per second

z/VM fields

Some fields for Linux on z/VM are available in both the sys_list and sys windows. Others are available only in the sys_list window or only in the sys window.

In the sys_list and sys windows:

Identifier	Column label	Explanation
c	cpu	CPU time per second
m	mgm	Management time per second
C	Cpu+	Total CPU time
M	Mgm+	Total management time
o	online	Online time

In the sys_list window only:

Identifier	Column label	Explanation
y	system	Name of the z/VM guest virtual machine (always shown)
#	#cpu	Number of CPUs
O	#cpuop	Number of operating CPUs
u	memuse	Used memory
a	memmax	Maximum memory
r	wcur	Current weight
x	wmax	Maximum weight

In the sys window only:

Identifier	Column label	Explanation
i	cpuid	CPU identifier (always shown)
v	visual	Visualization of CPU time per second

Units

Depending on the field type, the values can be displayed in different units.

In the sys_list and sys windows, the units are displayed under the column headings in parenthesis. Each unit can be specified through the **`--fields`** command line option. Units can also be selected interactively. To change a unit, enter select mode in the fields window. Then, select the field where you want to change the unit, and press the "+" or "-" keys to go through the available units. The following units are supported:

Units of time:

Unit	Explanation
us	Microseconds (10^{-6} seconds)
ms	Milliseconds (10^{-3} seconds)
%	Hundreds of a second (10^{-2} seconds) or percent
s	Seconds
m	Minutes
hm	Hours and minutes
dhm	Days, hours, and minutes

Units of memory:

Unit	Explanation
KiB	Kibibytes (1 024 bytes)
MiB	Mebibytes (1 048 576 bytes)
GiB	Gibibytes (1 073 741 824 bytes)

Other units:

Unit	Explanation
str	String
#	Count or number
vis	Visualization

CPU types

Enable or disable CPU types in interactive mode in the cpu_types window.

The CPU types can also be specified with the `--cpu_types` command line option.

The calculation of the CPU data uses CPUs of the specified types only. For example, if you want to see how much CPU time is consumed by your Linux systems, enable CPU type IFL.

On z/VM the processor type is always UN and you cannot select the type.

In an LPAR the following CPU types can be selected either interactively or with the `--cpu_types` command line option:

Identifier	Column label	Explanation
i	IFL	Integrated Facility for Linux. On older machines IFLs might be shown as CPs.
p	CP	CP processor type.
u	UN	Unspecified processor type (other than CP or IFL).

Examples

These examples show typical uses of **hyptop**.

- To start **hyptop** with the sys_list window in interactive mode, enter:

```
# hyptop
```

  - If your Linux instance is running in an LPAR that has permission to see the other LPARs, the output looks like the following example:

```
12:30:48 | cpu-t: IFL(18)  CP(3)  UN(3)                                ?=help
system   #core    core    mgm    Core+  Mgm+   online
(str)      (#)     (%)    (%)     (hm)  (hm)    (dhm)
S05LP30     10  461.14  10.18  1547:41  8:15 11:05:59
S05LP33      4  133.73   7.57   220:53  6:12 11:05:54
S05LP50      4   99.26   0.01   146:24  0:12 10:04:24
S05LP02      1   99.09   0.00   269:57  0:00 11:05:58
TRX2CFA      1    2.14   0.03     3:24  0:04 11:06:01
S05LP13      6    1.36   0.34     4:23  0:54 11:05:56
TRX1        19    1.22   0.14    13:57  0:22 11:06:01
TRX2        20    1.16   0.11    26:05  0:25 11:06:00
S05LP55      2    0.00   0.00     0:22  0:00 11:05:52
S05LP56      3    0.00   0.00     0:00  0:00 11:05:52
           413  823.39  23.86  3159:57 38:08 11:06:01
```

  - If your Linux instance runs in a z/VM guest virtual machine that has permission to see the other z/VM guest virtual machines, the output looks like the following example:

```
12:32:21 | CPU-T: UN(16)                                               ?=help
system    #cpu    cpu    Cpu+    online  memuse  memmax  wcur
(str)      (#)    (%)    (hm)     (dhm)   (GiB)   (GiB)   (#)
T6360004    6  100.31  959:47 53:05:20    1.56    2.00   100
DTCVSW1     1    0.00    0:00 53:16:42    0.01    0.03   100
T6360002    6    0.00  166:26 40:19:18    1.87    2.00   100
OPERATOR    1    0.00    0:00 53:16:42    0.00    0.03   100
T6360008    2    0.00    0:37 30:22:55    0.32    0.75   100
T6360003    6    0.00 3700:57 53:03:09    4.00    4.00   100
NSLCF1      1    0.00    0:02 53:16:41    0.03    0.25   500
PERFSVM     1    0.00    0:53 2:21:12     0.04    0.06     0
TCPIP       1    0.00    0:01 53:16:42    0.01    0.12  3000
DIRMAINT    1    0.00    0:04 53:16:42    0.01    0.03   100
DTCVSW2     1    0.00    0:00 53:16:42    0.01    0.03   100
RACFVM      1    0.00    0:00 53:16:42    0.01    0.02   100
           75  101.57 5239:47 53:16:42   15.46   22.50  3000
```

  At the top, the sys and sys_list windows show a list of the CPU types that are used for the current CPU and core dispatch time calculation.

- To start **hyptop** with the sys window showing performance data for LPAR MYLPAR, enter:

```
# hyptop -w sys -s mylpar
```

  The result looks like the following example:

```
11:18:50 MYLPAR cpu-t: IFL(0) CP(24) UN(2)                              ?=help
coreid type    core mgm visual
(#)   (str)    (%)  (%) (vis)
0        CP  50.78 0.28 |########################                      |
1        CP  62.76 0.17 |##############################                |
2        CP  71.11 0.48 |##################################            |
3        CP  32.38 0.24 |###############                               |
4        CP  64.35 0.32 |##############################                |
5        CP  67.61 0.40 |################################              |
6        CP  70.95 0.35 |##################################            |
7        CP  62.16 0.41 |##############################                |
8        CP  70.48 0.25 |##################################            |
9        CP  56.43 0.20 |###########################                   |
10       CP   0.00 0.00 |                                              |
11       CP   0.00 0.00 |                                              |
12       CP   0.00 0.00 |                                              |
13       CP   0.00 0.00 |                                              |
=:V:N       609.02 3.10
```

- To start **hyptop** with the sys_list window in batch mode, enter:

```
# hyptop -b
```

- To start **hyptop** with the sys_list window in interactive mode, with the fields dispatch time (in milliseconds), and online time (unit default), and sort the output according to online time, enter:

```
# hyptop -f c:ms,o -S o
```

- To start **hyptop** with the sys_list window in batch mode with update delay 5 seconds and 10 iterations, enter:

```
# hyptop -b -d 5 -n 10
```

- To start **hyptop** with the sys_list window and use only CPU types IFL and CP for dispatch time calculation, enter:

```
# hyptop -t ifl,cp
```

- To start **hyptop** on Linux in LPAR mode with the sys_list window and display all LPAR fields, including the thread information, enter:

```
# hyptop -f "#,T,c,e,m,C,E,M,o"
```

The result looks like the following example:

```
13:47:42 cpu-t: IFL(0) CP(38) UN(0)                                    ?=help
system    #core  #The    core     the  mgm   Core+    thE+  Mgm+    online
(str)       (#)   (#)     (%)     (%)  (%)    (hm)    (hm)  (hm)     (dhm)
S35LP41      12    24  101.28  170.28 0.28 1056:10 1756:11  8:45 158:04:04
S35LP42      16    32   35.07   40.07 0.44 5194:52 6193:52 12:45 158:04:04
S35LP64       3     3    1.20    1.20 0.00    0:31    0:31  0:00  12:03:54
...
```

In the example, the Linux instances in LPARs S35LP41 and S35LP43 run with 2 threads per core. The thread time, as the sum of the two threads, exceeds the core dispatch time.

The Linux instance in LPAR S35LP64 does not use simultaneous multithreading.

- To start **hyptop** on Linux on z/VM with the sys_list window and display a selection of z/VM fields, including the management time, enter:

```
# hyptop -f "#,c,m,C,M,o"
```

The result looks like the following example:

```
17:52:56 cpu-t: IFL(0) UN(2)                                          ?=help
system     #cpu   cpu   mgm   Cpu+  Mgm+  online
(str)       (#)   (%)   (%)   (hm)  (hm)   (dhm)
G3545010      3  0.55  0.05   0:05  0:02 0:03:14
G3545021      3  0.04     -   0:00     - 0:02:43
G3545025      2  0.01     -   0:00     - 0:04:08
...

G3545099      1  0.00     -   0:00     - 0:09:06
             52  0.61  0.05   0:27  0:02 0:09:06
```

In the example, `hyptop` runs on a Linux instance in z/VM guest virtual machine `G3545010`. In the sys_list window, this is the only guest virtual machine for which management data is displayed.

Scenario

Perform the steps described in this scenario to start **hyptop** with the sys window with system MYLPAR with the fields dispatch time (unit milliseconds) and total dispatch time (unit default), sort the output according to the total dispatch time, and then reverse the sort order.

Procedure

1. Start hyptop.

```
# hyptop
```

2. Go to select mode by pressing the → key. The display will freeze.
3. Navigate to the row for the system you want to look (in the example MYLPAR) at using the ↑ and ↓ keys.

```
12:15:00 | CPU-T: IFL(18)  CP(3)  UN(3)                               ?=help
system   #core  core    mgm    Core+  Mgm+   online
(str)     (#)   (%)     (%)    (hm)   (hm)   (dhm)
 MYLPAR    4   199.69   0.04   547:41 8:15 11:05:59
S05LP33    4   133.73   7.57  220:53  6:12 11:05:54
S05LP50    4    99.26   0.01  146:24  0:12 10:04:24
S05LP02    1    99.09   0.00  269:57  0:00 11:05:58
...
S05LP56    3     0.00   0.00    0:00  0:00 11:05:52
         413   823.39  23.86 3159:57 38:08 11:06:01
```

4. Open the sys window for MYLPAR by pressing the → key.

```
12:15:51 MYLPAR CPU-T: IFL(18) CP(3) UN(2)                            ?=help
coreid type    core mgm visual
(#)   (str)    (%)  (%) (vis)
0       IFL  99.84 0.02 |##################################################
1       IFL  99.85 0.02 |##################################################
2       IFL   0.00 0.00 |
3       IFL   0.00 0.00 |
=:V:N       199.69 0.04
```

5. Press the F key to go to the fields selection window:

```
Select Fields and Units                                               ?=help
K S ID     UNIT AGG  DESCRIPTION
p * type   str  none CPU type
c * core   %    sum  Core dispatch time per second
e   the    %    sum  Thread time per second
m * mgm    %    sum  Management time per second
C   core+  hm   sum  Total core dispatch time
E   thE+   %    sum  Total thread time
M   mgm+   hm   sum  Total management time
o   online dhm  max  Online time
v * visual vis  none Visualization of CPU time per second
```

Ensure that dispatch time per second and total dispatch time are selected and for dispatch time microseconds are used as unit:

a. Press the P key, the M key, and the V key to disable `CPU type`, `Management time per second`, and `Visualization`.

b. Press the C key to enable `Total core dispatch time`.

c. Then select the `Core dispatch time per second` row by pressing the → and ↓ keys.

d. Press the minus key (-) to switch from the percentage (%) unit to the microseconds (ms) unit.

```
Select Fields and Units ?=help
K S ID     UNIT AGG  DESCRIPTION
p   type   str  none CPU type
c * core   ms   sum  Core dispatch time per second
e   the    %    sum  Thread time per second
m   mgm    %    sum  Management time per second
C * core+  hm   sum  Total core dispatch time
E   thE+   %    sum  Total thread time
M   mgm+   hm   sum  Total management time
o   online dhm  max  Online time
v   visual vis  none Visualization of CPU time per second
```

Press the ← key twice to return to the sys window.

6. To sort by `Total core dispatch time` and list the values from low to high, press the Shift + C keys twice:

```
13:44:41 MYLPAR cpu-t: IFL(18) CP(3) UN(2)                ?=help
coreid   core   Core+
(#)      (ms)    (hm)
2        0.00    0:00
3        0.00    0:00
1       37.48  492:55
0       23.84  548:52
=:^:N   61.33 1041:47
```

Results

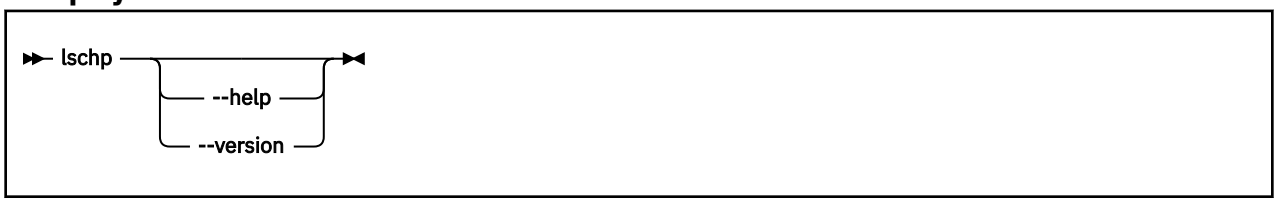
You can do all of these steps in one by entering the command:

```
# hyptop -w sys -s mylpar -f c:ms,C -S C -S C
```

lschp - List channel paths

Use the **lschp** command to display information about channel paths.

lschp syntax



Where:

-v or --version
displays the version number of **lschp** and exits.

-h or --help
displays a short help text, then exits.

Output column description:

CHPID
Channel-path identifier.

Vary
Logical channel-path state:

- 0 = channel-path is not used for I/O.
- 1 = channel-path is used for I/O.

Cfg.
Channel-path configure state:

- 0 = stand-by
- 1 = configured
- 2 = reserved
- 3 = not recognized

Type
Channel-path type identifier.

Cmg
Channel measurement group identifier.

Shared
Indicates whether a channel-path is shared between LPARs:

- 0 = channel-path is not shared
- 1 = channel-path is shared

PCHID
Physical channel path identifier, or, if enclosed in brackets, internal channel identifier. The mapping might not be available to Linux when running as a z/VM guest. If so, use the CP command:

```
QUERY CHPID <num> PCHID
```

A column value of '-' indicates that a facility associated with the corresponding channel-path attribute is not available.

Examples

- To query the configuration status of channel path ID 0.40 issue:

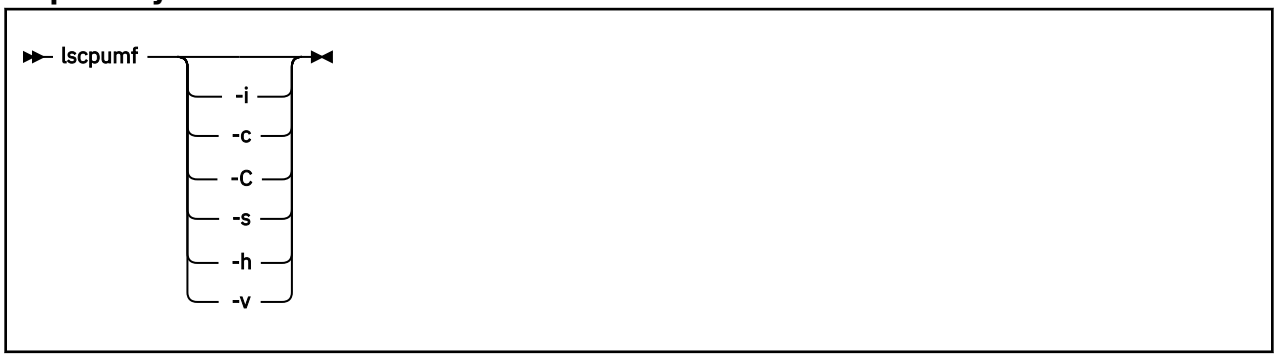
```
# lschp

CHPID Vary Cfg. Type Cmg Shared  PCHID
=====================================
...
...
0.40  1    1    1b   2   1       0580
...
...
```

The value under **Cfg.** shows that the channel path is configured (1).

lscpumf - Display information about the CPU-measurement facilities

Use the **lscpumf** command to display information about the CPU-measurement facilities.

lscpumf syntax



where:

-i or --info
displays detailed information about available and supported CPU measurement facilities.

-c or --list-counters
lists counters that are provided by the CPU-measurement facility, omitting counters for which the LPAR is not authorized. For counter measurements with the perf program, the raw event identifier and symbolic counter name are displayed.

-C or --list-all-counters
lists all counters that are provided by the CPU-measurement counter facility, regardless of LPAR authorization. To list only those counters for which the LPAR is authorized, use the **-c** option. For counter measurements with the perf program, the raw event identifier and symbolic counter name are displayed.

-s or --list-sampling-events
lists perf raw events that activate the sampling facility.

-v or --version
displays the version number of **lscpumf** and exits.

-h or --help
displays out a short help text, then exits. To view the man page, enter **man lscpumf**.

Examples

- To display the supported facilities, issue:

```
# lscpumf
CPU-measurement Counter Facility
CPU-measurement Sampling Facility
```

- To display details about the facilities, issue:

```
# lscpumf -i
CPU-measurement Counter Facility
------------------------------------------------------------------------
Version: 3.7

Authorized counter sets:
    Basic counter Set
    Crypto-Activity counter Set
    Extended counter Set
    MT-diagnostic counter Set
    Problem-State counter Set

Linux perf event support: Yes (PMU: cpum_cf)

CPU-measurement Sampling Facility
------------------------------------------------------------------------
Sampling Interval:
     Minimum:      20800 cycles (approx.    250000 Hz)
     Maximum:  170372800 cycles (approx.        30 Hz)

Authorized sampling modes:
     basic:      (sample size:  32 bytes)
     diagnostic: (sample size: 173 bytes)

Linux perf event support: Yes (PMU: cpum_sf)

Current sampling buffer settings for cpum_sf:
    Basic-sampling mode
        Minimum:     15 sample-data-blocks (  64KB)
        Maximum:   8176 sample-data-blocks (  32MB)

    Diagnostic-sampling mode (including basic-sampling)
        Minimum:     90 sample-data-blocks ( 364KB)
        Maximum:  49056 sample-data-blocks ( 192MB)
        Size factor:  6
```

- To display perf event information for authorized sampling functions, issue:

```
# lscpumf -s
Perf events for activating the sampling facility
================================================================================

Raw
event   Name    Description
------------------------------------------------------------------------------
rb0000  SF_CYCLES_BASIC

                Sample CPU cycles using basic-sampling mode.
                This event is not associated with a counter set.
```

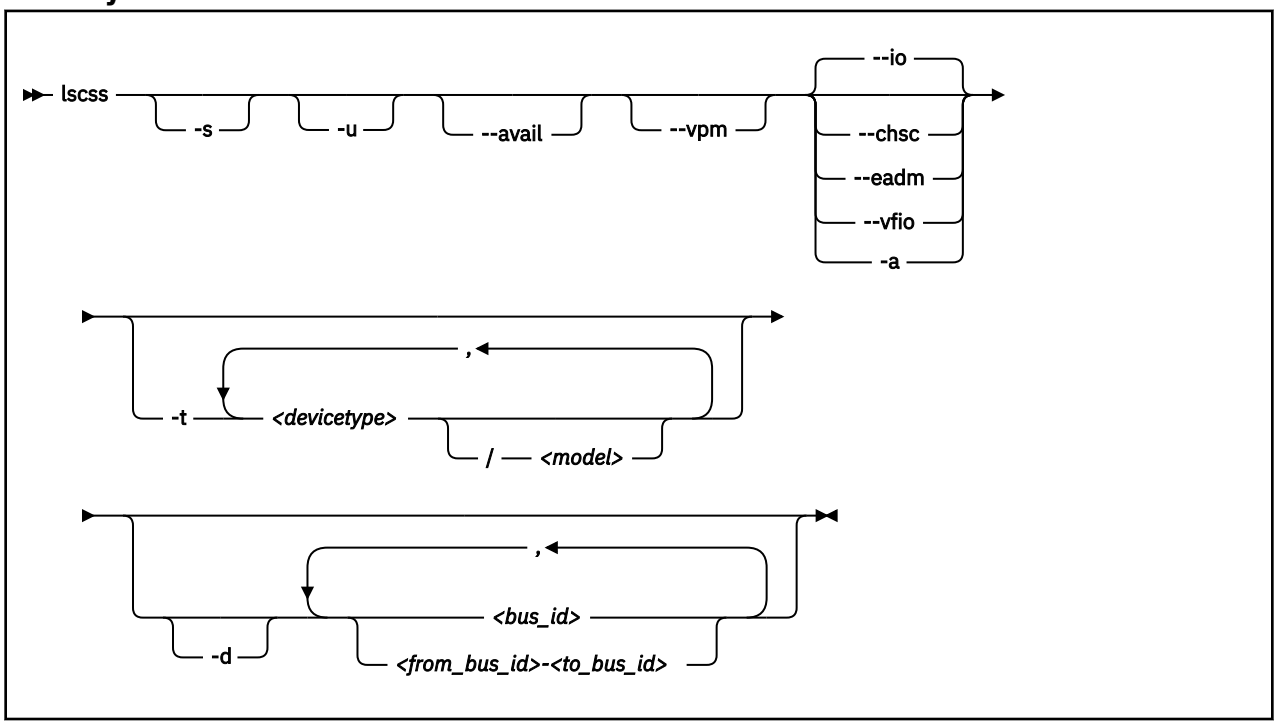
- To list all counters that are provided by your IBM Z hardware, issue:

```
# lscpumf -C
Perf event counter list for IBM z13
================================================================================

Raw
event   Name    Description
--------------------------------------------------------------------------------
r0      CPU_CYCLES

                Cycle Count.
                Counter 0 / Basic Counter Set.

r1      INSTRUCTIONS

                Instruction Count.
                Counter 1 / Basic Counter Set.

r2      L1I_DIR_WRITES

                Level-1 I-Cache Directory Write Count.
                Counter 2 / Basic Counter Set.

r3      L1I_PENALTY_CYCLES

                Level-1 I-Cache Penalty Cycle Count.
                Counter 3 / Basic Counter Set.

r4      L1D_DIR_WRITES

                Level-1 D-Cache Directory Write Count.
                Counter 4 / Basic Counter Set.

r5      L1D_PENALTY_CYCLES

                Level-1 D-Cache Penalty Cycle Count.
                Counter 5 / Basic Counter Set.

r20     PROBLEM_STATE_CPU_CYCLES

                Problem-State Cycle Count.
                Counter 32 / Problem-State Counter Set.

r21     PROBLEM_STATE_INSTRUCTIONS

                Problem-State Instruction Count.
                Counter 33 / Problem-State Counter Set.
 ...
```

lscss - List subchannels

Use the **lscss** command to gather subchannel information from sysfs and display it in a summary format.

lscss syntax



Where:

-s or --short
strips the bus ID in the command output down to the four-digit device number.

-u or --uppercase
displays the output with uppercase letters. The default is lowercase.

Changed default: Earlier versions of **lscss** printed the command output in uppercase. Specify this option to obtain the former output style.

--avail
includes the availability attribute of I/O devices.

--vpm
shows verified paths in a mask. Channel paths listed in this mask are available to Linux device drivers for I/O. Reasons for a channel path being unavailable include:

- The corresponding bit is not set in at least one of the PIM, PAM, or POM masks.
- The channel path is varied offline.
- Linux received no interrupt to I/O using this channel path.

--io
limits the output to I/O subchannels and corresponding devices. This option is the default.

--chsc
limits the output to CHSC subchannels.

--eadm
limits the output to EADM subchannels.

--vfio
For KVM hosts: shows information for subchannels that are used for VFIO CCW mediated devices, see "Setting up VFIO pass-through DASDs" on page 480.

-a or --all
does not limit the output.

-t or --devtype
limits the output to information about the specified device types and, if provided, the specified model.

<devicetype>
specifies a device type.

<model>
is a specific model of the specified device type.

-d or --devrange
interprets bus IDs as specifications of devices. By default, bus IDs are interpreted as specifications of subchannels.

<bus_id>
specifies an individual subchannel; if used with `-d` specifies an individual device. If you omit the leading 0.*<subchannel set ID>.*, 0.0. is assumed.

If you specify subchannels or devices, the command output is limited to these subchannels or devices.

<from_bus_id>-<to_bus_id>
specifies a range of subchannels; if used with `-d` specifies a range of devices. If you omit the leading 0.*<subchannel set ID>.*, 0.0. is assumed.

If you specify subchannels or devices, the command output is limited to these subchannels or devices.

-v or --version
displays the version number of **`lscss`** and exits.

-h or --help
displays a short help text, then exits. To view the man page enter **`man lscss`**.

Examples

- This command lists all subchannels, including subchannels that do not correspond to I/O devices:

```
# lscss -a
IO Subchannels and Devices:
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.f500 0.0.05cf  1732/01 1731/01 yes  80  80  ff   76000000 00000000
0.0.f501 0.0.05d0  1732/01 1731/01 yes  80  80  ff   76000000 00000000
0.0.f502 0.0.05d1  1732/01 1731/01 yes  80  80  ff   76000000 00000000
0.0.6194 0.0.36e0  3390/0c 3990/e9 yes  fc  fc  ff   32333435 40410000
0.0.6195 0.0.36e1  3390/0c 3990/e9 yes  fc  fc  ff   32333435 40410000
0.0.6196 0.0.36e2  3390/0c 3990/e9 yes  fc  fc  ff   32333435 40410000

CHSC Subchannels:
Device   Subchan.
----------------
n/a      0.0.ff40

EADM Subchannels:
Device   Subchan.
----------------
n/a      0.0.ff00
n/a      0.0.ff01
n/a      0.0.ff02
n/a      0.0.ff03
n/a      0.0.ff04
n/a      0.0.ff05
n/a      0.0.ff06
n/a      0.0.ff07
```

- This command lists subchannels with an attached 3480 model 04 or 3590 tape device and strips the bus ID and the subchannel ID in the command output down to the four-digit IDs:

```
# lscss -s -t 3480/04,3590
Device Subchan. DevType CU Type Use PIM PAM POM CHPIDs
------------------------------------------------------------------------
0480   000e     3480/04 3480/01     80  80  ff  10000000 00000000
0a38   000f     3590/11 3590/50     80  80  ff  10000000 00000000
```

- This command limits the output to subchannels with attached DASD model 3390 type 0a:

```
# lscss -t 3390/0a
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
------------------------------------------------------------------------
0.0.2f08 0.0.0a78  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000
0.0.2fe5 0.0.0b55  3390/0a 3990/e9      c0  c0  bf   34400000 00000000
0.0.2fe6 0.0.0b56  3390/0a 3990/e9      c0  c0  bf   34400000 00000000
0.0.2fe7 0.0.0b57  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000
```

- This command limits the output to the subchannel range 0.0.0b00-0.0.0bff:

```
# lscss 0.0.0b00-0.0.0bff
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
------------------------------------------------------------------------
0.0.2fe5 0.0.0b55  3390/0a 3990/e9      c0  c0  bf   34400000 00000000
0.0.2fe6 0.0.0b56  3390/0a 3990/e9      c0  c0  bf   34400000 00000000
0.0.2fe7 0.0.0b57  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000
```

- This command limits the output to subchannels 0.0.0a78 and 0.0.0b57 and shows the availability:

```
# lscss --avail 0a78,0b57
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs            Avail.
------------------------------------------------------------------------------
0.0.2f08 0.0.0a78  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000 good
0.0.2fe7 0.0.0b57  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000 good
```

- This command limits the output to subchannel 0.0.0a78 and displays uppercase output:

```
# lscss -u 0a78
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
------------------------------------------------------------------------
0.0.2F08 0.0.0A78  3390/0A 3990/E9 YES  C0  C0  FF   34400000 00000000
```

- This command limits the output to subchannels that correspond to I/O device 0.0.7e10 and the device range 0.0.2f00-0.0.2fff:

```
# lscss -d 2f00-2fff,0.0.7e10
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
------------------------------------------------------------------------
0.0.2f08 0.0.0a78  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000
0.0.2fe5 0.0.0b55  3390/0a 3990/e9      c0  c0  bf   34400000 00000000
0.0.2fe6 0.0.0b56  3390/0a 3990/e9      c0  c0  bf   34400000 00000000
0.0.2fe7 0.0.0b57  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000
0.0.7e10 0.0.1828  3390/0c 3990/e9 yes  f0  f0  ef   34403541 00000000
```

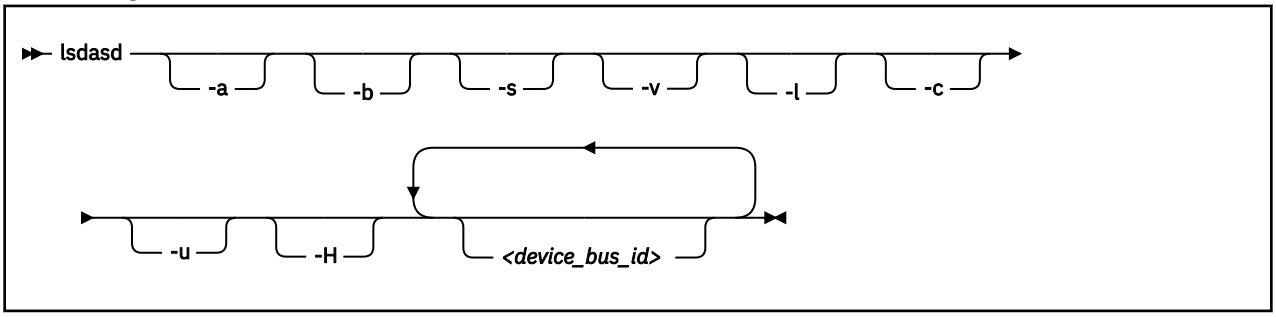
- This example shows a CHPID with PIM, PAM, and POM masks that are OK, but the entry in the VPM column indicates that one of the paths, 0x41, is not usable for I/O.

```
# lscss --vpm
Device    Subchan. DevType  CU Type Use  PIM PAM POM VPM CHPIDs
------------------------------------------------------------------------
0.0.f500  0.0.05cf  1732/01 1731/01 yes  80  80  ff  80  76000000 00000000
0.0.f501  0.0.05d0  1732/01 1731/01 yes  80  80  ff  80  76000000 00000000
0.0.f502  0.0.05d1  1732/01 1731/01 yes  80  80  ff  80  76000000 00000000
0.0.6194  0.0.3700  3390/0c 3990/e9 yes  fc  fc  ff  f8  32333435 40410000
0.0.6195  0.0.3701  3390/0c 3990/e9 yes  fc  fc  ff  f8  32333435 40410000
0.0.6196  0.0.3702  3390/0c 3990/e9 yes  fc  fc  ff  f8  32333435 40410000
0.0.6197  0.0.3703  3390/0c 3990/e9      fc  fc  ff  00  32333435 40410000
0.2.5600  0.2.0040  1732/03 1731/03      80  80  ff  00  5d000000 00000000
```

lsdasd - List DASD devices

Use the **lsdasd** command to gather information about DASD devices from sysfs and display it in a summary format.

lsdasd syntax



Where:

-a or --offline
includes devices that are currently offline.

-b or --base
omits PAV alias devices. Lists only base devices.

-s or --short
strips the bus ID in the command output down to the four-digit device number.

-v or --verbose
Obsolete. This option has no effect on the output.

-l or --long
extends the output to include attributes, the UID and path information.

-c or --compat
creates output of this command as with versions earlier than 1.7.0.

-u or --uid
includes and sorts output by UID.

-H or --host_access_list
shows information about all operating system instances that use this device.

--version
displays the version of the command.

<device_bus_id>
limits the output to information about the specified devices only.

-h or --help
displays a short help text, then exits.

Examples

- The following command lists all DASD (including offline DASDS):

```
# lsdasd -a
Bus-ID      Status     Name     Device     Type    BlkSz     Size       Blocks
================================================================================
0.0.0190    offline
0.0.0191    offline
0.0.019d    offline
0.0.019e    offline
0.0.0592    offline
0.0.4711    offline
0.0.4712    offline
0.0.4f2c    offline
0.0.4d80    active    dasda     94:0       ECKD    4096     4695MB       1202040
0.0.4f19    active    dasdb     94:4       ECKD    4096    23034MB       5896800
0.0.4d81    active    dasdc     94:8       ECKD    4096     4695MB       1202040
0.0.4d82    active    dasdd     94:12      ECKD    4096     4695MB       1202040
0.0.4d83    active    dasde     94:16      ECKD    4096     4695MB       1202040
```

- The following command shows information only for the DASD with device number 0x4d80 and strips the bus ID in the command output down to the device number:

```
# lsdasd -s 0.0.4d80
Bus-ID      Status    Name       Device    Type     BlkSz    Size       Blocks
================================================================================
4d80        active    dasda      94:0      ECKD     4096     4695MB     1202040
```

- The following command shows only online DASDs in the format of **lsdasd** versions earlier than 1.7.0:

```
# lsdasd -c
0.0.4d80(ECKD) at ( 94: 0) is dasda : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4f19(ECKD) at ( 94: 4) is dasdb : active at blocksize 4096, 5896800 blocks, 23034 MB
0.0.4d81(ECKD) at ( 94: 8) is dasdc : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4d82(ECKD) at ( 94: 12) is dasdd : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4d83(ECKD) at ( 94: 16) is dasde : active at blocksize 4096, 1202040 blocks, 4695 MB
```

- The following command shows the device geometry, UID, path information, and some of the settings for the DASD with device bus-ID 0.0.4d82:

```
lsdasd -l 0.0.4d82
0.0.4d82/dasdd/94:12
  status:                           active
  type:                             ECKD
  blksz:                            4096
  size:                             4695MB
  blocks:                           1202040
  extent_size:                      1113
  logical_capacity:                 262668
  space_allocated:                  262668
  use_diag:                         0
  readonly:                         0
  eer_enabled:                      0
  erplog:                           0
  hpf:                              1
  uid:                              IBM.75000000010671.4d82.16
  fc_security:                      Encryption
  paths_installed:                  30 31 32 33 3c 3d
  paths_in_use:                     31 32 33
  paths_non_preferred:
  paths_invalid_cabling:            3c
  paths_cuir_quiesced:              30
  paths_invalid_hpf_characteristics:  3d
  paths_error_threshold_exceeded:
```

  In the example, three of the installed paths are unused for different reasons:

  - The path with CHPID 3c is not used because of a cabling error to the storage system. This channel path does not connect to the same physical disk space as the other channel path for this device.
  - The path with CHPID 30 is not used because of a control-unit initiated reconfiguration (CUIR).
  - The path with CHPID 3d is not used because its High Performance FICON characteristics do not match with the paths currently in use.

- The following command shows whether other operating system instances access device 0.0.bf45:

```
# lsdasd -H bf45
Host information for 0.0.bf45
Path-Group-ID          LPAR  CPU   FL  Status  Sysplex      Max_Cyls        Time
================================================================================
88000d29e72964ce8570b8 0d    29e7  50  ON      TRX1LNX1    268434453           0
88000e29e72964ce8570c3 0e    29e7  50  ON                  268434453           0
88000f29e72964ce8570d1 0f    29e7  50  ON                  268434453           0
88011d29e72964ce8570d4 1d    29e7  50  ON                  268434453           0
88011e29e72964ce8570d9 1e    29e7  50  ON                  268434453           0
88011f29e72964ce8570e3 1f    29e7  50  ON                  268434453           0
88022d29e72964ce8570e6 2d    29e7  50  ON                  268434453           0
88022e29e72964ce8570ea 2e    29e7  50  ON                  268434453           0
88022f29e72964ce8570f1 2f    29e7  50  ON                  268434453           0
88033d29e72964ce8570f7 3d    29e7  50  ON                  268434453           0
88033e29e72964ce8570fe 3e    29e7  50  ON                  268434453           0
88033f29e72964ce85710e 3f    29e7  50  ON                  268434453           0
80004229e72964ce7dce74 42    29e7  00  OFF                     65520           0
80004a29e72964ce7db60d 4a    29e7  00  OFF                     65520           0
80003c29e72964ce8481a6 3c    29e7  00  OFF                     65520           0
80004629e72964ce7f1c13 46    29e7  70  ON-RSV                  65520  1424174863
```

Status values are:

ON
The device is online.

OFF
The device is offline.

ON-RSV
The device is online and reserved.

OFF-RSV
The device is offline and reserved by an operating system instance in another LPAR.

The meaning of the columns is as follows:

Path-group-ID
A 22-digit hexadecimal number assigned by the operating system when setting the DASD online. This ID uniquely identifies the operating system to the storage server.

LPAR
A 2 digit LPAR ID.

CPU
A 4 digit CPU ID, as it is defined in the HMC or can be read from `/proc/cpuinfo`.

FL
A 2 digit hexadecimal flag. 0x20 means reserved, 0x50 means online.

Sysplex
The 8-character EBCDIC name of the SYSPLEX.

MAX_CYLS
The maximum number of cylinders per volume that are supported by the host.

TIME
Time the device has been reserved in seconds since July 1, 1970.

- In this example, first all DASDs are listed, then the details for a thinly provisioned DASD, marked as ESE, is listed:

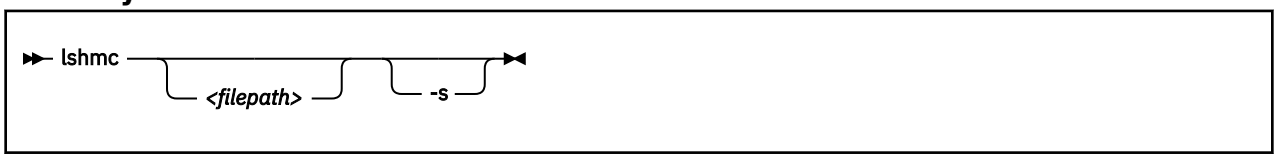
```
# lsdasd
Bus-ID    Status    Name      Device  Type         BlkSz  Size      Blocks
==============================================================================
0.0.95e0  alias                     ECKD
0.0.95e1  alias                     ECKD
0.0.3300  active    dasda     94:0    ECKD         4096   21129MB   5409180
0.0.95d0  active    dasdb     94:4    ECKD (ESE)   4096   42259MB   10818360
0.0.95d1  n/f       dasdc     94:8    ECKD (ESE)

# lsdasd -l 95d0
0.0.95d0/dasdb/94:4
  status:                             active
  type:                               ECKD (ESE)
  blksz:                              4096
  size:                               42259MB
  blocks:                             10818360
  extent_size:                        1113
  logical_capacity:                   60102
  space_allocated:                    27825
  use_diag:                           0
  readonly:                           0
  eer_enabled:                        0
  erplog:                             0
  hpf:                                1
  uid:                                IBM.750000000ABT31.9500.d0
  fc_security:                        Encryption
  paths_installed:                    38 39 3a 3b
  paths_in_use:                       38 39 3a 3b
  paths_non_preferred:
  paths_invalid_cabling:
  paths_cuir_quiesced:
  paths_invalid_hpf_characteristics:
  paths_error_threshold_exceeded:
```

lshmc - List media contents in the HMC media drive

Use the **lshmc** command to display the contents of the media in the HMC media drive.

Before you begin: To be able to use this command, you need the **hmcdrv** module (see Chapter 30, "HMC media device driver," on page 383).

lshmc syntax



Where:

<filepath>
specifies a directory or path to a file to be listed. Path specifications are relative to the root of the file system on the media. You can use the asterisk (*) and question mark (?) as wildcards. If this specification is omitted, the contents of the root directory are listed.

-s or --short
limits the output to regular files in a short listing format. Omits directories, symbolic links, and device nodes and other special files.

-v or --version
displays version information for the command.

-h or --help
displays a short help text, then exits. To view the man page, enter **man lshmc**.

Examples

- To list the files in the root directory of the media in the HMC's media drive, issue:

```
# lshmc
```

- If the hmcdrv kernel module is not loaded, load it before you issue the **lshmc** command:

```
# modprobe hmcdrv
# lshmc
```

- To list all HTML files in subdirectory www, issue:

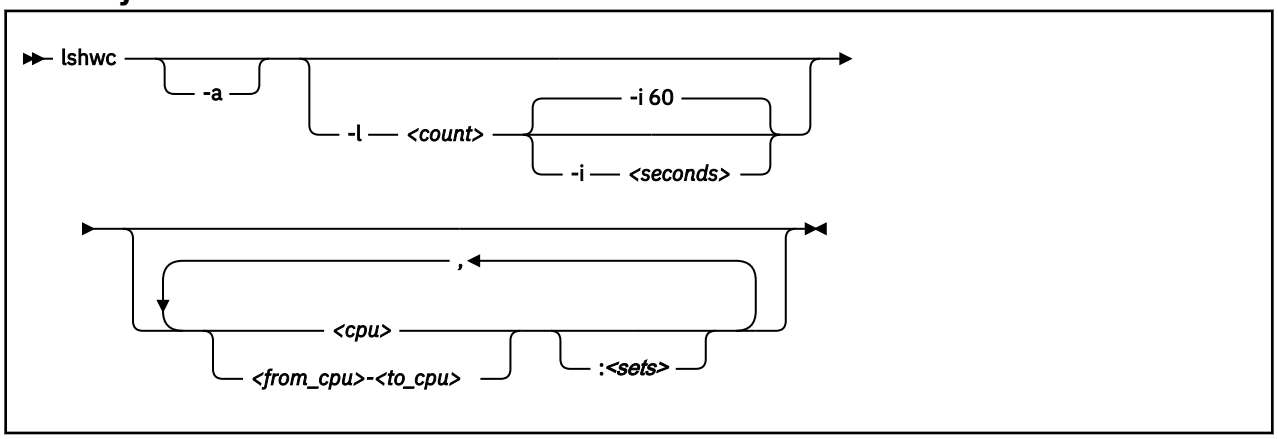
```
# lshmc /www/*.html
```

lshwc - Extract CPU Measurement Facilities counter sets

Use the **lshwc** command to extract complete counter sets from the CPU Measurement Facilities for Linux on IBM Z.

The command output is in comma-separated values (CSV) format. Each output line starts with a timestamp and the CPU number, followed by the extracted counter values.

lshwc syntax



Where:

-a or --allcpu
Displays counter values separately for each CPU. The default is a total summary line of all counters from all CPUs.

-l or --loop *<count>*
Performs the specified number of read operations.

-i or --interval *<seconds>*
Specifies a time interval, in seconds, that the command waits between read operations. The default is 60 seconds.

<cpu>
The number of a logical CPU.

<from_cpu>-<to_cpu>
A range of CPUs, where *<from_cpu>* is the number of the first logical CPU in the range and *<to_cpu>* is the number of the last logical CPU in the range. Ranges are useful if you want to extract the same counter sets for multiple, consecutive CPU numbers.

<sets>
A specification of counter sets to be extracted for a specified CPU or CPU range. By default, all available counters are extracted. Use a single alphabetic character to specify a counter set. You can specify multiple counter sets by specifying a multiple letters without a blank. In the specification, uppercase letters are equivalent to lowercase letters.

b|B
Include the basic counter set.

c|C
Include the crypto counter set.

e|E
Include the extended counter set.

m|M
Include the MT_Diagnostic counter set.

p|P|u|U
Include the problem counter set.

a|A
Include all known counter sets. This is the default.

Examples

- This example selects the basic and problem counter sets on CPU 0 and CPU 1. Two read operations are performed and a summary line is printed for each read operation.

```
# lshwc -l2 0-1:BP
Date,Time,CPU,CPU_CYCLES(0),INSTRUCTIONS(1),L1I_DIR_WRITES(2),L1I_PENALTY_CYCLES(3), ...
2021-04-01,11:50:32,Total,125422,39421,304,13953,454,97489,0,0
2021-04-01,11:51:32,Total,68074231,16386850,194028,21382384,317227,104503489,777383,14198
```

- This example shows the counter values of the problem state counter set per CPU. CPU 0 and CPU 1 are selected.

```
# lshwc -l 3 -a 0-1:p
Date,Time,CPU,PROBLEM_STATE_CPU_CYCLES(32),PROBLEM_STATE_INSTRUCTIONS(33)
2021-04-01,11:54:47,CPU0,0,0
2021-04-01,11:54:47,CPU1,0,0
2021-04-01,11:54:47,Total,0,0
2021-04-01,11:55:47,CPU0,818775,14198
2021-04-01,11:55:47,CPU1,125689,1306
2021-04-01,11:55:47,Total,944464,15504
2021-04-01,11:56:47,CPU0,3207071426,1489122591
2021-04-01,11:56:47,CPU1,3225092021,1489278312
2021-04-01,11:56:47,Total,6432163447,2978400903
```

lsluns - Discover LUNs, or show encryption state of attached LUNs

Use the **lsluns** command to list logical unit numbers (LUNs) discovered in the Fibre Channel storage area networks (SAN), or to show the encryption state of zfcp-attached LUNs.

lsluns is designed for environments where SCSI devices are attached through the zfcp device driver.

lsluns lists all LUNs discovered in the Fibre Channel SAN. See "Discover LUNs in the Fibre Channel storage area network (SAN)" on page 668.

lsluns -a shows the encryption state of the attached LUNs. See "Show the encryption state of zfcp-attached LUNs" on page 669.

For all other uses, such as listing attached LUNs or properties other than encryption, use other tools such as:

- **lszfcp -D** See "lszfcp - List zfcp devices" on page 693
- **lszdev zfcp-lun -ii** See "lszdev - Display IBM Z device configurations" on page 688
- **lsscsi -tvxx** See the man page for more details.

Discover LUNs in the Fibre Channel storage area network (SAN)

Discovering LUNs only makes sense for NPIV-enabled FCP devices without zfcp automatic LUN scan. zfcp automatic LUN scan is available as of kernel version 2.6.37, if not disabled with `zfcp.allow_lun_scan=0`. See "Setting up the zfcp device driver" on page 182.

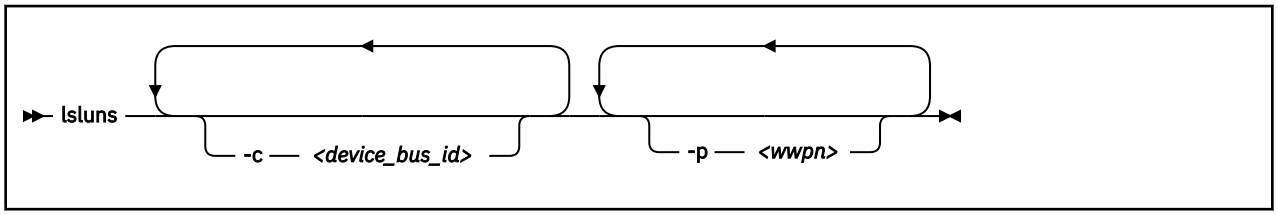
Note: Discovering LUNs causes extra SAN traffic for each target port WWPN.

Temporary LUN Attachment
If not attached already, **lsluns** temporarily attaches LUN 0 (or if this fails, the WLUN 0xc101000000000000) during runtime. Do not terminate **lsluns** with a signal. Signals interfere with the removal of temporarily attached LUNs.

Storage Products
Some storage products return a peripheral device type of `31==0x1f` with peripheral qualifier 0 in a SCSI standard INQUIRY command for an unmapped FCP LUN 0. Examples are: IBM Storwize products, including IBM V7000, IBM V840, IBM V9000, and IBM SAN Volume Controller. For **lsluns** to work with such storage products, you must have a host mapping on the storage side, which maps some volume to exported FCP LUN 0x0000000000000000 (Storwize host map property "SCSI ID" 0) for each used FCP-device initiator WWPN. The volume can be a minimum-sized thin-provisioned shared stand-in volume.

lsluns syntax

Filter the listing by specifying one or more FCP device bus-IDs, target port WWPNs, or both.



Where:

-c *<device_bus_id>* or --ccw *<device_bus_id>*
filters LUNs by one or more adapters with the specified FCP device-bus IDs. When used in conjunction with –p, only those LUNs are listed that also satisfy at least one of the –p constraints.

-p *<wwpn>* or --port *<wwpn>*
filters LUNs by one or more target ports with the specified WWPNs. When used in conjunction with –c, only those LUNs are listed that also satisfy at least one of the –c constraints.

-v or --version
displays version information and exits.

-h or --help
displays an overview of the syntax. To view the man page, enter **man lsluns**.

Examples

- This example lists all LUNs discovered in the FC SAN on adapter 0.0.3922:

```
# lsluns -c 0.0.3922
```

- This example shows all LUNs discovered in the FC SAN on target port 0x500507630300c562:

```
# lsluns -p 0x500507630300c562
Scanning for LUNs on adapter 0.0.5922
        at port 0x500507630300c562:
                0x4010400000000000
                0x4010400100000000
                0x4010400200000000
                0x4010400300000000
                0x4010400400000000
                0x4010400500000000
```

- This example shows all LUNs discovered in the FC SAN on:
  - Adapter 0.0.3922 and port 0x5005123456789000
  - Adapter 0.0.3922 and port 0x5005abcdefabc000
  - Adapter 0.0.fc00 and port 0x5005123456789000
  - Adapter 0.0.fc00 and port 0x5005abcdefabc000

```
# lsluns -c 0.0.3922 –c 0.0.fc00 –p 0x5005123456789000 –p 0x5005abcdefabc000
```

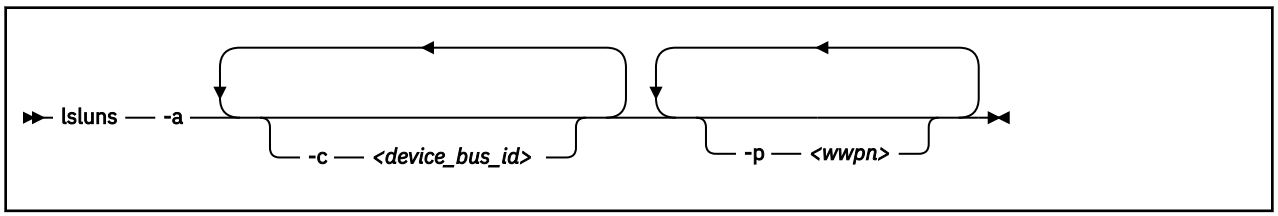
Show the encryption state of zfcp-attached LUNs

lsluns -a shows the encryption state of the attached LUNs.

Note: Running **lsluns -a** causes extra SAN traffic for each attached LUN.

lsluns syntax

Filter the listing by specifying one or more FCP device bus-IDs, target port WWPNs, or both.



Where:

-a or --active
shows the encryption state of the attached LUNs. Encrypted devices are indicated with a bracketed X immediately after the LUN number.

-c *<device_bus_id>* or --ccw=*<device_bus_id>*
filters LUNs by one or more adapters with the specified FCP device-bus IDs. When used in conjunction with –p, only those LUNs are listed that also satisfy at least one of the –p constraints.

-p *<wwpn>* or --port=*<wwpn>*
filters LUNs by one or more target ports with the specified WWPNs. When used in conjunction with –c, only those LUNs are listed that also satisfy at least one of the –c constraints.

-v or --version
displays version information and exits.

-h or --help
displays an overview of the syntax. To view the man page, enter **man lsluns**.

Examples

- This example shows the encryption status of attached LUNs:

```
# lsluns -a
adapter = 0.0.3c02
     port = 0x500507630300c562
          lun = 0x401040a200000000(X) /dev/sg0    Disk    IBM:2107900
          lun = 0x401040a300000000    /dev/sg1    Disk    IBM:2107900
          ...
     port = 0x500507630303c562
          ...
adapter = 0.0.593a
          ...
```

The (X) after the LUN number indicates that the device is encrypted.

lsqeth - List qeth-based network devices

Use the **lsqeth** command to display a summary of information about qeth-based network devices.

Before you begin: To be able to use this command, you must also install **qethconf** (see "qethconf - Configure qeth devices" on page 708). You install **qethconf** and **lsqeth** with the s390utils RPM.

lsqeth syntax

Where:

-p or --proc
displays the interface information in the former `/proc/qeth` format. This option can generate input to tools that expect this particular format.

<interface>
limits the output to information about the specified interface only.

-h or --help
displays a short help text, then exits.

Examples

- The following command lists information about interface encf5a0 in the default format:

```
# lsqeth encf5a0
Device name                     : encf5a0
-------------------------------------------
        card_type               : OSD_1000
        cdev0                   : 0.0.f5a0
        cdev1                   : 0.0.f5a3
        cdev2                   : 0.0.f5a4
        chpid                   : B5
        online                  : 1
        portname                : OSAPORT
        portno                  : 0
        route4                  : no
        route6                  : no
        checksumming            : hw checksumming
        state                   : UP (LAN ONLINE)
        priority_queueing       : always queue 2
        fake_broadcast          : 0
        buffer_count            : 64
        layer2                  : 0
        large_send              : no
        isolation               : none
        sniffer                 : 0
```

- The following command lists information about all qeth-based interfaces in the former `/proc/qeth` format:

```
# lsqeth -p
devices                   CHPID interface        cardtype       port chksum prio-q'ing rtr4 rtr6 lay'2 cnt
------------------------- ----- ---------------  -------------- ---- ------ ---------- ---- ---- ----- ----
0.0.833c/0.0.8340/0.0.8341 xFE   enc833c    HiperSockets   0    sw     always_q_2 no   no   0     128
0.0.f5a0/0.0.f5a3/0.0.f5a4 xB5   encf5a0    OSD_1000       0    hw     always_q_2 no   no   0      64
0.0.fba2/0.0.fba3/0.0.fba4 xB0   encfba2    OSD_1000       0    sw     always_q_2 no   no   1      64
```

lsreipl - List IPL and re-IPL settings

Use the **lsreipl** command to find out which boot device and which options are used if you issue the reboot command.

You can also display information about the current boot device.

lsreipl syntax

Where:

-i or --ipl
displays the IPL setting.

-v or --version
displays the version number of **lsreipl** and exits.

-h or --help
displays an overview of the syntax. Any other parameters are ignored.

By default the re-IPL device is set to the current IPL device. Use the chreipl command to change the re-IPL settings.

Examples

- This example shows the current re-IPL settings:

```
# lsreipl
Re-IPL type:     fcp
WWPN:            0x500507630300c562
LUN:             0x401040b300000000
Device:          0.0.1700
bootprog:        0
br_lba:          0
Loadparm:        ""
Bootparms:       ""
clear:           0
```

lsscm - List storage-class memory increments

Use the **lsscm** command to list status and other information about available storage-class memory increments.

lsscm syntax

Where:

-h or --help
displays help information for the command. To view the man page, enter **man lsscm**.

-v or --version
displays version information for the command.

In the output table, the columns have the following meaning:

SCM Increment
Starting address of the storage-class memory increment.

Size
Size of the block device that represents the storage-class memory increment.

Name
Name of the block device that represents the storage-class memory increment.

Rank
A quality ranking in the form of a number in the range 1 - 15 where a lower number means better ranking.

D_state
Data state of the storage-class memory increment. A number that indicates whether there is data on the increment. The data state can be:

1
The increment contains zeros only.

2
Data was written to the increment.

3
No data was written to the increment since the increment was attached.

O_state
Operation state of the storage-class memory increment.

Pers
Persistence attribute.

ResID
Resource identifier.

Examples

- This command lists all increments:

```
    # lsscm
 SCM Increment    Size    Name  Rank D_state O_state Pers ResID
-------------------------------------------------------------
0000000000000000 16384MB scma     1       2       1    2     1
0000000400000000 16384MB scmb     1       2       1    2     1
```

lsshut - List the current system shutdown actions

Use the **lsshut** command to see how the Linux instance is configured for the `halt`, `poff`, `reboot`, `restart`, and `panic` system shutdown triggers.

For more information about the shutdown triggers and possible shutdown actions, see Chapter 8, "Shutdown actions," on page 123.

If the action is kdump, a second action might be listed. This second action is the backup action that is taken if kdump fails. See *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751 for details about using kdump.

lsshut syntax

Where:

-h or --help
displays a short help text, then exits.

-v or --version
displays the version number of **lsshut** and exits.

Examples

- To query the configuration issue:

```
# lsshut
Trigger    Action
=======================
Halt       stop
Power off  vmcmd (LOGOFF)
Reboot     reipl
Restart    kdump,dump_reipl
Panic      kdump,dump_reipl
```

lstape - List tape devices

Use the **lstape** command to gather information about tape devices and display it in a summary format.

It gathers information about the following types of tape devices:

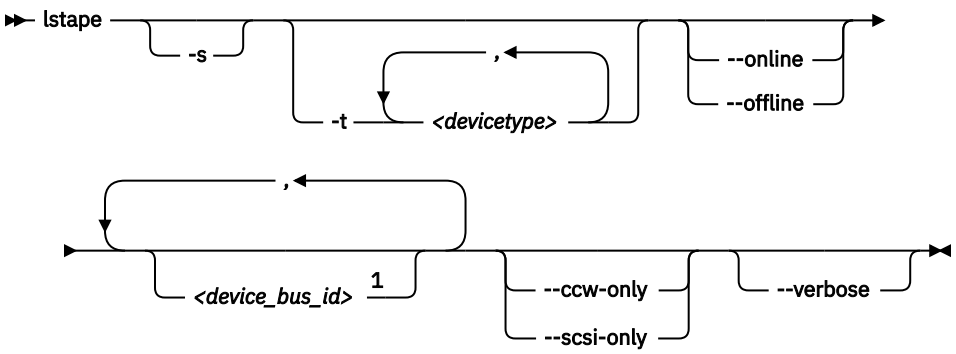
- CCW-attached tape devices
- Tape drive and medium charger devices that are available through the sysfs SCSI bus (see “Displaying tape information” on page 234)

For information about SCSI tape devices, the command uses the following sources for the information displayed:

- The IBMtape or the open source lin_tape driver.
- The sg_inq command from the scsi/sg3_utils package.

  Note: Issuing **lstape** without option `--ccw-only` causes extra SAN traffic for each SCSI tape or changer device.
- The st SCSI tape device driver in the Linux kernel.
- The ch SCSI medium changer device driver in the Linux kernel.

lstape syntax



Notes:

[1] specify the first device bus-ID with a leading blank.

Where:

-s or --shortid
strips the "0.*<n>*." from the device bus-IDs in the command output. For CCW-attached devices only.

-t or --type
limits the output to information about the specified type or types of CCW-attached devices only.

--ccw-only
limits the output to information about CCW-attached devices only.

--scsi-only
limits the output to information about tape devices that are attached to the SCSI bus.

--online | --offline
limits the output to information about online or offline CCW-attached tape devices only.

<device_bus_id>
limits the output to information about the specified CCW-attached tape device or devices.

--verbose
For tape devices attached to the SCSI bus only. Prints the serial of the tape and information about the FCP or virtio-scsi-ccw connection as an additional text line after each SCSI tape in the list.

-h or --help
displays a short help text.

--version
displays the version of the command.

Examples

- This command displays information about all tapes that are found, here one CCW-attached tape and one tape and changer device that is configured for zFCP:

```
#> lstape
FICON/ESCON tapes (found 1):
TapeNo  BusID      CuType/Model  DevType/Model  BlkSize  State   Op   MedState
0       0.0.0480   3480/01       3480/04        auto     UNUSED  ---  UNLOADED

SCSI tape devices (found 2):
Generic  Device       Target        Vendor     Model         Type     State
sg4      IBMchanger0  0:0:0:0       IBM        03590H11      changer  running
sg5      IBMtape0     0:0:0:1       IBM        03590H11      tapedrv  running
```

If only the st tape device driver and the ch changer device driver are loaded, the output lists those names in the device section:

```
#> lstape
FICON/ESCON tapes (found 1):
TapeNo  BusID      CuType/Model  DevType/Model  BlkSize  State   Op    MedState
0       0.0.0480   3480/01       3480/04        auto     UNUSED  ---   UNLOADED

SCSI tape devices (found 2):
Generic Device       Target         Vendor     Model         Type     State
sg0     sch0         0:0:0:0        IBM        03590H11      changer  running
sg1     st0          0:0:0:1        IBM        03590H11      tapedrv  running
```

- This command displays information about all available CCW-attached tapes.

```
# lstape –-ccw-only
TapeNo  BusID      CuType/Model DevType/DevMod  BlkSize State   Op      MedState
0       0.0.0132   3590/50      3590/11         auto    IN_USE  ---     LOADED
1       0.0.0110   3490/10      3490/40         auto    UNUSED  ---     UNLOADED
2       0.0.0133   3590/50      3590/11         auto    IN_USE  ---     LOADED
3       0.0.012a   3480/01      3480/04         auto    UNUSED  ---     UNLOADED
N/A     0.0.01f8   3480/01      3480/04         N/A     OFFLINE ---     N/A
```

- This command limits the output to tapes of type 3480 and 3490.

```
# lstape -t 3480,3490
TapeNo  BusID      CuType/Model DevType/DevMod  BlkSize State   Op      MedState
1       0.0.0110   3490/10      3490/40         auto    UNUSED  ---     UNLOADED
3       0.0.012a   3480/01      3480/04         auto    UNUSED  ---     UNLOADED
N/A     0.0.01f8   3480/01      3480/04         N/A     OFFLINE ---     N/A
```

- This command limits the output to those tapes of type 3480 and 3490 that are currently online.

```
# lstape -t 3480,3490 --online
TapeNo  BusID      CuType/Model DevType/DevMod  BlkSize State   Op      MedState
1       0.0.0110   3490/10      3490/40         auto    UNUSED  ---     UNLOADED
3       0.0.012a   3480/01      3480/04         auto    UNUSED  ---     UNLOADED
```

- This command limits the output to the tape with device bus-ID 0.0.012a and strips the "0.*<n>*." from the device bus-ID in the output.

```
# lstape -s 0.0.012a
TapeNo  BusID      CuType/Model DevType/DevMod  BlkSize State   Op      MedState
3       012a       3480/01      3480/04         auto    UNUSED  ---     UNLOADED
```

- This command limits the output to SCSI devices but gives more details. The serial numbers are only displayed if the **sg_inq** command is found on the system.

```
#> lstape --scsi-only --verbose
Generic Device       Target        Vendor     Model        Type       State
        HBA          WWPN                     Serial
sg0     st0          0:0:0:1       IBM        03590H11     tapedrv    running
        0.0.1708     0x500507630040727b       NO/INQ
sg1     sch0         0:0:0:2       IBM        03590H11     changer    running
        0.0.1708     0x500507630040727b       NO/INQ
```

- Example details about a zfcp-attached SCSI tape library with multiple paths operated by the IBM lin_tape device driver instead of Linux st or ch.

```
# lstape --scsi-only --verbose
SCSI tape devices (found 8):
Generic Device        Target       Vendor   Model            Type     State
        HBA           WWPN                  Serial
sg0     IBMtape0      0:0:0:0      IBM      ULT3580-TD6      tapedrv  running
        0.0.5080      0x2002000e1115c62f    10WT037733
sg1     IBMchanger0   0:0:0:1      IBM      3573-TL          changer  running
        0.0.5080      0x2002000e1115c62f    00L4U78W6497_LL0
sg4     IBMtape2      0:0:1:0      IBM      ULT3580-TD6      tapedrv  running
        0.0.5080      0x2008000e1115c62f    10WT037701
sg5     IBMchanger2   0:0:1:1      IBM      3573-TL          changer  running
        0.0.5080      0x2008000e1115c62f    00L4U78W6497_LL0
sg6     IBMtape3      1:0:0:0      IBM      ULT3580-TD6      tapedrv  running
        0.0.50c0      0x2002000e1115c62f    10WT037733
sg7     IBMchanger3   1:0:0:1      IBM      3573-TL          changer  running
        0.0.50c0      0x2002000e1115c62f    00L4U78W6497_LL0
sg2     IBMtape1      1:0:1:0      IBM      ULT3580-TD6      tapedrv  running
        0.0.50c0      0x2008000e1115c62f    10WT037701
sg3     IBMchanger1   1:0:1:1      IBM      3573-TL          changer  running
        0.0.50c0      0x2008000e1115c62f    00L4U78W6497_LL0
```

Data fields for SCSI tape devices

There are specific data fields for SCSI tape devices.

Table 78. lstape data fields for SCSI tape devices

Attribute	Description
Generic	SCSI generic device file for the tape drive (for example `/dev/sg0`). This attribute is "N/A" if the SCSI generic device driver, sg, is not available.
Device	Main device file for accessing the tape drive, for example: • `/dev/st0` for a tape drive that is attached through the Linux st device driver • `/dev/sch0` for a medium changer device that is attached through the Linux ch device driver • `/dev/IBMchanger0` for a medium changer that is attached through the IBMtape or lin_tape device driver • `/dev/IBMtape0` for a tape drive that is attached through the IBMtape or lin_tape device driver
Target	The ID in Linux used to identify the SCSI device.
Vendor	The vendor field from the tape drive.
Model	The model field from the tape drive.
Type	"tapedrv" for a tape drive or "changer" for a medium changer.

Table 78. lstape data fields for SCSI tape devices (continued)	
Attribute	**Description**
State	The state of the SCSI device in Linux. This state is an internal state of the Linux kernel, any state other than "running" can indicate problems.
HBA	The bus-ID of the FCP device or of the virtio-scsi-ccw virtual HBA to which the tape drive is attached. "N/A" if the device does not have a sysfs ancestor with subsystem ccw.
WWPN	The WWPN (worldwide port name) of the tape drive in the SAN. "N/A" if the device is not attached through zfcp.
Serial	The serial number field from the tape drive. "NO/SG" if the SCSI generic device driver, sg, is not available. "NO/INQ" if sg is available, but the **sg_inq** command from the scsi/sg3_utils package is unavailable.

lsstp - Show STP configuration information

Use the **lsstp** command to display information about the current Server Time Protocol (STP) configuration like Coordinated Timing Network (CTN) ID, timing state, and leap seconds.

lsstp syntax

Where:

-h or --help
displays a short help text. To view the man page, enter **man lsstp**.

-v or --version
displays the version of the **lsstp** command.

Output description:

STP online
Indication of the online state.

CTN ID
The ID of the CTN. If it can be decoded as EBCDIC, it is shown as an EBCDIC string, otherwise a hexadecimal representation is shown.

CTN Type
The type of timing network.

No CTN
STP is not configured for attachment to a CTN.

STP-only
STP is configured and attached to a CTN with only STP nodes.

Mixed
STP is configured and attached to a CTN with both STP and external time reference (ETR) nodes.

Stratum
The number of servers in the timing path between the local STP clock and the selected primary time server.

Timing mode
The timing mode of the Time-of-day (TOD) clock.

Local
The TOD clock is stepped by the local hardware oscillator and is not steered by the STP facility.

ETR
The TOD clock is synchronized with an attached 9037 Sysplex Timer.

STP
The TOD clock is steered by the STP facility to maintain synchronization with a Coordinated Server Time (CST).

Uninitialized
The TOD clock is not initialized. The STP facility is allowed to perform a step adjustment to the TOD clock for synchronization.

Timing state
The synchronization state of the STP facility. Can be unsynchronized, synchronized, or stopped.

DST offset
The Daylight Savings Time (DST) offset relative to UTC in minutes.

Timezone offset
The offset of the local time relative to UTC in minutes.

Time offset
The total time offset at the server. This field is valid only in mixed CTN configurations.

Active leap seconds
The number of leap seconds that are currently in effect at the STP facility.

Scheduled leap second
If a leap second insertion or deletion is scheduled in the STP facility, this field shows the day and time of the scheduled change.

Example

```
# lsstp
STP online:            yes
CTN ID:                STPM46
CTN type:              STP-only
Stratum:               1
Timing mode:           STP
Timing state:          Synchronized
DST offset:            60
Timezone offset:       60
Time offset:           120
Active leap seconds:   27
Scheduled leap second: -
```

lszcrypt - Display cryptographic devices

Use the **lszcrypt** command to display information about cryptographic adapters that are managed by the cryptographic device driver and its AP bus attributes.

To set the attributes, use "chzcrypt - Modify the cryptographic configuration" on page 587. The following information can be displayed for each cryptographic adapter:

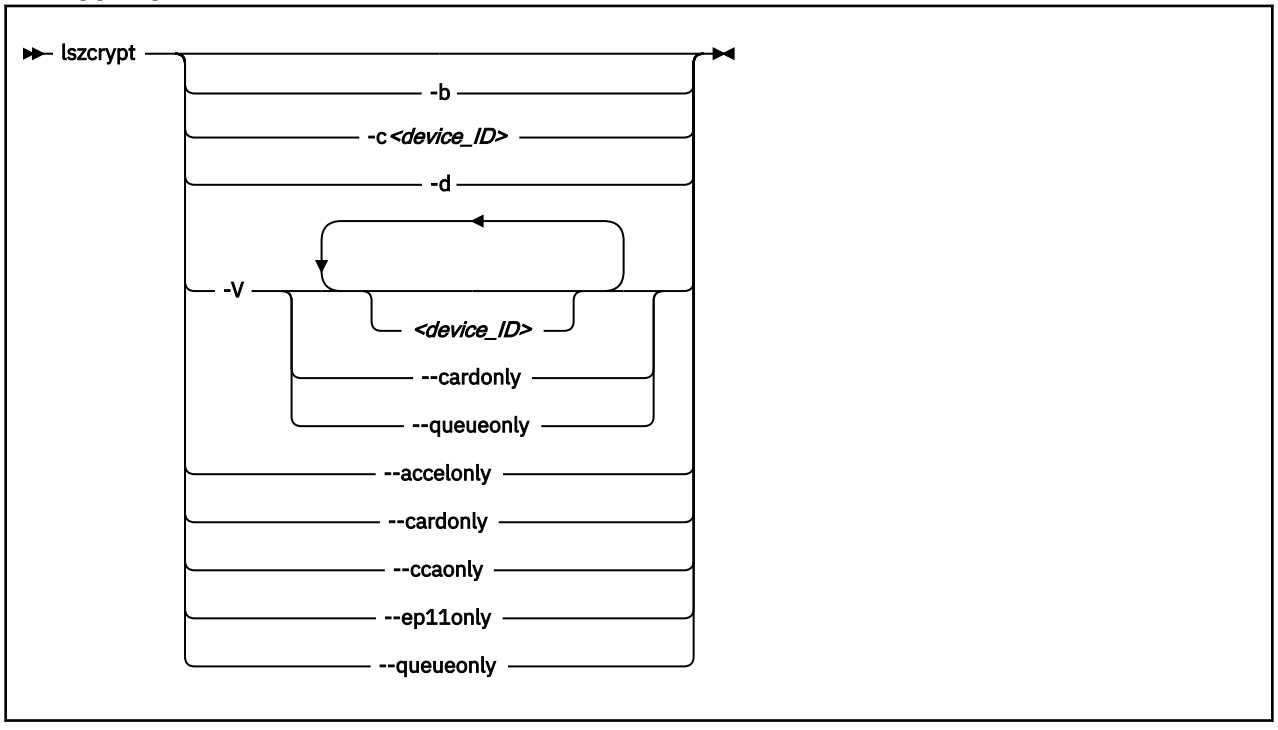
- The card type
- The status
  - online: The card is online to Linux.
  - offline: The card is configured at the LPAR level, but set offline within Linux.
  - deconfigured: The card is available to the LPAR, but not configured at the LPAR level. The card is also offline within Linux.
- The hardware card type
- The card capability
- The hardware queue depth
- The request count
- The zcrypt submodule or alternative device driver that handles the device

  For information about alternative device drivers, see "Freeing AP queues for KVM guests" on page 510.

The following AP bus attributes can be displayed:

- The default AP domain
- The configuration timer
- The poll thread status
- The poll timeout
- The AP interrupt status

lszcrypt syntax



Where:

<device ID>
specifies a cryptographic adapter to display. A cryptographic device can be either an adapter ID or an AP queue device. If no devices are specified, information about all available devices is displayed. Both the adapter ID representation and the AP queue device representation are hexadecimal.

-b or --bus
displays the AP bus attributes.

-c *<device ID>* or --capability *<device ID>*
shows the capabilities of a cryptographic adapter as of Crypto Express2 (CEX2). The capabilities of a cryptographic adapter depend on the card type and the installed function facilities. A cryptographic adapter can provide one or more of the following capabilities:

- RSA 2K Clear Key
- RSA 4K Clear Key
- CCA Secure Key (full function set)
- CCA Secure Key (restricted function set)
- EP11 Secure Key
- Long RNG

The restricted function set for CCA Secure Key applies to shared adapters for z/VM guests (see “Cryptographic devices on z/VM” on page 493).

-d or --domains
shows the usage and control domains of the cryptographic device. The displayed domains of the cryptographic device depends on the initial cryptographic configuration.

- "C" indicates a control domain
- "U" indicates a usage domain
- "B" indicates both (control and usage domain)

-V or --verbose
enables the verbose level for cryptographic device information. It displays card type, online status, hardware card type, hardware queue depth, request count, pending request queue count, outstanding request queue count, and installed function facilities.

The installed functions are shown, as a sequence of letters, in the FUNCTION column of the verbose output mode, with the following meaning:

S
APSC facility available

M and C
RSA 4096 bit support

D
CCA Coprocessor function available

A
Accelerator function available

X
EP11 Coprocessor function available

N
APXA facility available

F
Full function set available

R
Restricted function set.

Depending on the hypervisor configuration, the hypervisor might filter cryptographic requests to allow only a subset of functions within the virtual runtime environment. For example, a shared CCA Coprocessor can be restricted by the hypervisor to allow only clear-key operations within the guests.

--accelonly
limits the output to cryptographic adapters in accelerator mode.

--cardonly
limits the output to adapters only.

--ccaonly
limits the output to cryptographic adapters in CCA-Coprocessor mode.

--ep11only
limits the output to cryptographic adapters in EP11-Coprocessor mode.

--queueonly
limits the output to AP queues only.

-s or --serial
displays the serial numbers of CCA and EP11 cryptographic adapters.

-h or --help
displays short information about command usage.

-v or --version
displays version information.

Examples

These examples illustrate common uses for **lszcrypt**.

- To display information about all available cryptographic devices and AP queues:

```
# lszcrypt
```

This command lists all devices grouped by cryptographic device, similar to the following example. The domain IDs are hexadecimal values.

```
CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS
----------------------------------------------
0a          CEX7P EP11-Coproc online      2506
0a.0011     CEX7P EP11-Coproc online      1615
0a.0036     CEX7P EP11-Coproc online       891
0c          CEX7A Accelerator online      3506
0c.0011     CEX7A Accelerator online      1753
0c.0036     CEX7A Accelerator online      1753
0e          CEX7C CCA-Coproc  online      1507
0e.0011     CEX7C CCA-Coproc  online       753
0e.0036     CEX7C CCA-Coproc  online       754
```

- To display AP bus information:

```
# lszcrypt -b
```

This command displays output similar to the following example:

```
ap_domain=0x11
ap_max_domain_id=0x54
ap_interrupts are enabled
config_time=30 (seconds)
poll_thread is disabled
poll_timeout=250000 (nanoseconds)
```

- To display the capabilities for the cryptographic device with adapter ID 0x0e:

```
# lszcrypt -c 0x0e
```

This command displays output similar to the following example:

```
card0e provides capability for:
RSA 4K Clear Key
CCA Secure Key (full function set)
Long RNG
```

- To list the usage and control domains of the cryptographic devices:

```
# lszcrypt -d
```

This command displays a table that lists all domains (in hex notation) similar to the following example:

```
DOMAIN 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
-------------------------------------------------------
    00  .  .  .  .  .  .  B  .  .  .  .  .  .  .  .  .
    10  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    20  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    30  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    40  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    50  .  B  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    60  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    70  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    80  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    90  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    a0  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    b0  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    c0  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    d0  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    e0  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
    f0  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
-------------------------------------------------------
C: Control domain
U: Usage domain
B: Both (Control + Usage domain)
```

- To display detailed information of all available cryptographic devices:

```
# lszcrypt -V
```

This example shows a CEX6S cryptographic device in accelerator mode (ID 0x03). It also shows three CEX7S devices, two of them in CCA coprocessor mode (IDs 0x08 and 0x0e) and one in EP11 coprocessor mode (ID 0x0a). The configured domains are 17 (0x0011) and 54 (0x0036). Adapter IDs and domain IDs are hexadecimal values.

```
# lszcrypt -V
CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH FUNCTIONS  DRIVER
-------------------------------------------------------------------------------------
03          CEX6A Accelerator online      2095        0     12     08 -MC-A-NF-  cex4card
03.0011     CEX6A Accelerator online      1047        0     12     08 -MC-A-NF-  cex4queue
03.0036     CEX6A Accelerator online      1048        0     12     08 -MC-A-NF-  cex4queue
08          CEX7C CCA-Coproc  online         0        0     13     08 S--D--NF-  cex4card
08.0011     CEX7C CCA-Coproc  -              0        0     13     08 S--D--NF-  -no-driver-
08.0036     CEX7C CCA-Coproc  -              0        0     13     08 S--D--NF-  -no-driver-
0a          CEX7P EP11-Coproc online      2506        0     13     08 -----XNF-  cex4card
0a.0011     CEX7P EP11-Coproc online      1615        0     13     08 -----XNF-  cex4queue
0a.0036     CEX7P EP11-Coproc online       891        0     13     08 -----XNF-  cex4queue
0e          CEX7C CCA-Coproc  online      1507        0     13     08 S--D--NF-  cex4card
0e.0011     CEX7C CCA-Coproc  online       753        0     13     08 S--D--NF-  cex4queue
0e.0036     CEX7C CCA-Coproc  online       754        0     13     08 S--D--NF-  cex4queue
```

"-no-driver-" in the DRIVER column means that the AP queue has been freed for use by alternative device drivers (see “Freeing AP queues for KVM guests” on page 510), but no such device driver is available. In the example, the vfio_ap device driver is not loaded. Otherwise, "vfio_ap" would be displayed instead of "-no-driver-".

In the example, all domains for adapter `0x08` have been freed from control by `zcrypt`. AP queues that are not handled by the `zcrypt` device driver are omitted from the non-verbose listing.

```
# lszcrypt
CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS
-----------------------------------------------
03          CEX6A Accelerator online      2095
03.0011     CEX6A Accelerator online      1047
03.0036     CEX6A Accelerator online      1048
08          CEX7C CCA-Coproc  online         0
0a          CEX7P EP11-Coproc online      2506
0a.0011     CEX7P EP11-Coproc online      1615
0a.0036     CEX7P EP11-Coproc online       891
0e          CEX7C CCA-Coproc  online      1507
0e.0011     CEX7C CCA-Coproc  online       753
0e.0036     CEX7C CCA-Coproc  online       754
```

- To limit the scope of the **`lszcrypt -V`** command, specify one or more device IDs as arguments to the command.

```
# lszcrypt -V 0x0a
CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH FUNCTIONS  DRIVER
-------------------------------------------------------------------------------------
0a          CEX7P EP11-Coproc online      2506        0     13     08 -----XNF-  cex4card
0a.0011     CEX7P EP11-Coproc online      1615        0     13     08 -----XNF-  cex4queue
0a.0036     CEX7P EP11-Coproc online       891        0     13     08 -----XNF-  cex4queue
```

Tip: In the device specification you can also use one-digit hexadecimal or decimal notation. The following specifications are all equivalent:

– `0x0 0x2 0xb`

– `0x00 0x02 0x0b`

– `0 2 11`

- To filter the output by adapter mode, for example, to list only adapters in CCA-Coprocessor mode, issue **`lszcrypt --ccaonly`**:

```
# lszcrypt --ccaonly
CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS
-----------------------------------------------
04          CEX7A CCA-Coproc  online      2095
04.0016     CEX7A CCA-Coproc  online      1047
05          CEX7A CCA-Coproc  online      1048
```

- To filter the output by adapter and AP queues, for example from **`lszcrypt -V`**:

```
# lszcrypt -V 0x0a
CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH FUNCTIONS  DRIVER
-------------------------------------------------------------------------------------
0a          CEX7P EP11-Coproc online      2506        0     13     08 -----XNF-  cex4card
0a.0011     CEX7P EP11-Coproc online      1615        0     13     08 -----XNF-  cex4queue
0a.0036     CEX7P EP11-Coproc online       891        0     13     08 -----XNF-  cex4queue
```

To list only the adapters, issue **`lszcrypt -V --cardonly`**:

```
# lszcrypt -V 0x0a
CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH FUNCTIONS  DRIVER
-------------------------------------------------------------------------------------
0a          CEX7P EP11-Coproc online      2506        0     13     08 -----XNF-  cex4card
```

To list the AP queues, issue **`lszcrypt -V --queueonly`**:

```
# lszcrypt -V 0x0a
CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH FUNCTIONS  DRIVER
-------------------------------------------------------------------------------------
0a.0011     CEX7P EP11-Coproc online      1615        0     13     08 -----XNF-  cex4queue
0a.0036     CEX7P EP11-Coproc online       891        0     13     08 -----XNF-  cex4queue
```

- To display the serial number of adapters;

```
# lszcrypt --serial
CARD.DOM TYPE MODE         STATUS     SERIALNR
-----------------------------------------------
04       CEX8C CCA-Coproc  online     93AADHR3
05       CEX8C CCA-Coproc  online     93AADHZV
06       CEX8P EP11-Coproc online     93AADFK7
0c       CEX7C CCA-Coproc  deconfig   -
0d       CEX7C CCA-Coproc  online     93AADEY1
0f       CEX7C CCA-Coproc  online     93AADEVV
17       CEX8P EP11-Coproc online     93AADH0C
1a       CEX7P EP11-Coproc online     93AADFAD
```

lszdev - Display IBM Z device configurations

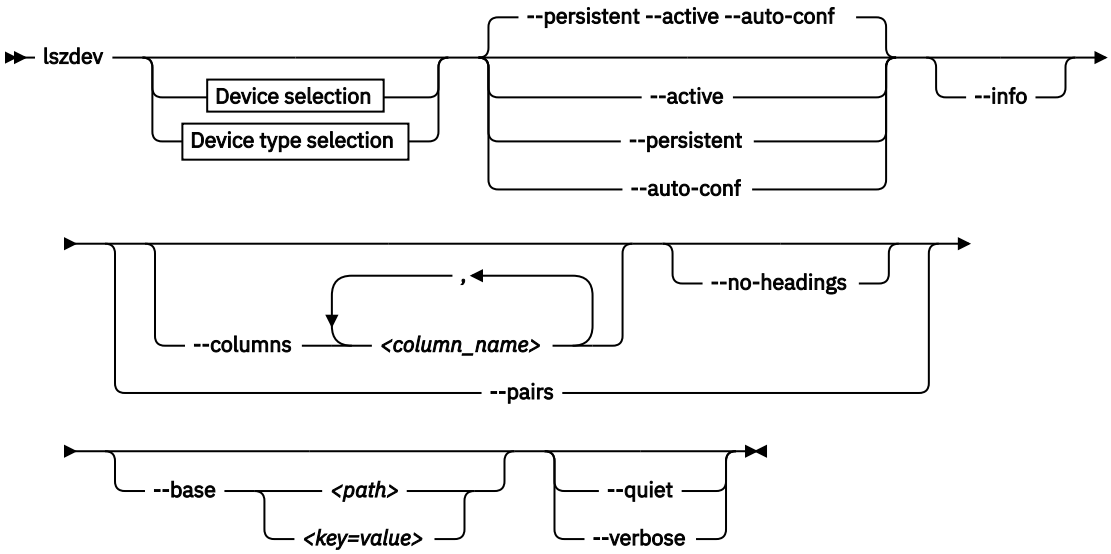
Use the **lszdev** command to display the configuration of devices and device drivers that are specific to IBM Z. Supported device types include storage devices (DASD and zFCP) and networking devices (QETH, CTC, and LCS).

Configuration information is taken from three sources: the active configuration of the currently running system, the persistent configuration stored in configuration files, and the auto-configuration from the Support Element (SE). By default, **lszdev** displays information from all available sources. If no auto-configuration data is displayed, the mainframe model does not support such data, or none has been specified. For details about the auto-configuration, see Chapter 3, “Device auto-configuration for Linux in LPAR mode,” on page 21.

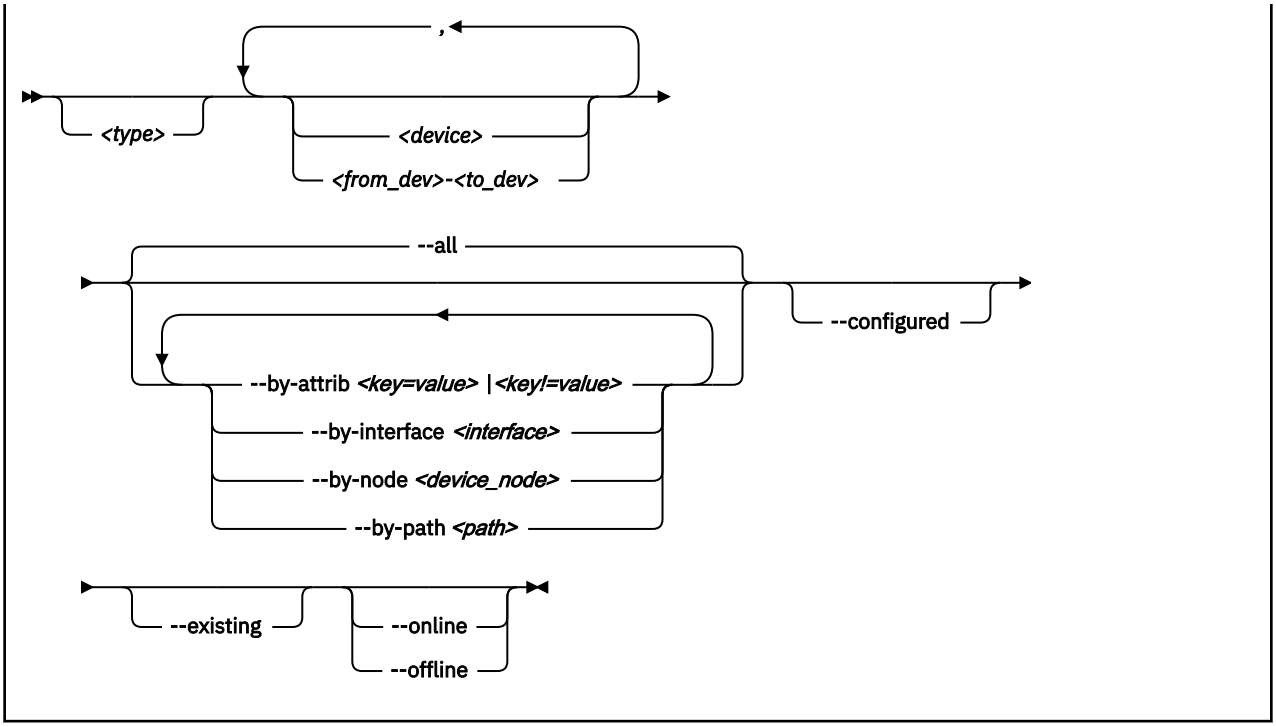
The **lszdev** command supports two different views:

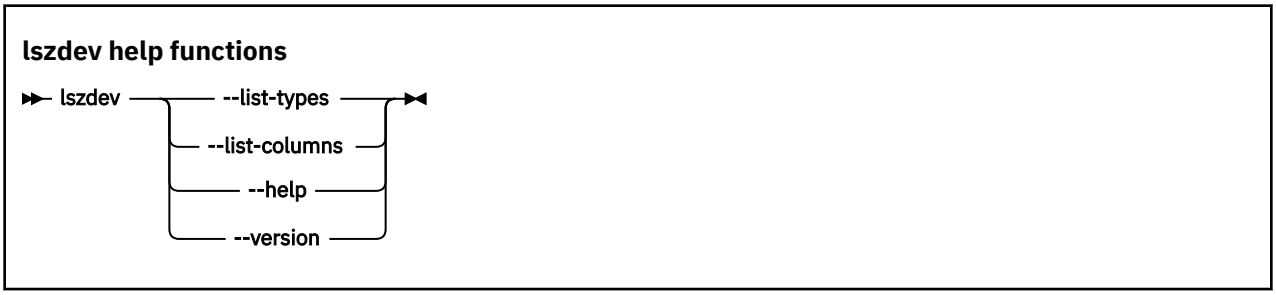
- The list view provides overview information for selected devices in list form with configurable columns (default)
- The details view provides detailed per-device information

lszdev main syntax



Device selection





Where:

<type>
restricts the output to the specified device type. A device type typically corresponds to a device driver. Multiple device types are sometimes provided for the same driver, for example, both "dasd-eckd" and "dasd-fba" are related to the DASD device driver. You can work with types in the following ways:

- To display data for devices with matching type and ID only, specify a device type and a device ID, for example:

```
# lszdev dasd 0.0.8000
```

- To display the configuration of the device type itself, specify a device type together with the `--type` option, for example:

```
# lszdev dasd --type
```

To get a list of supported device types, use the `--list-types` option.

<device>
limits the output to information about a single device or a range of devices by device ID. To select a range of devices, specify the ID of the first and the last device in the range separated by a hyphen (-). Specify multiple IDs or ID ranges by separating IDs with a comma (,).

--all
lists all existing and configured devices. This option is the default.

--by-attrib *<key=value> | <key!=value>*
selects devices with a specified attribute, *<key>* that has a value of *<value>*. When specified as *<key!=value>*, lists all devices that do not provide an attribute named *<key>* with a value of *<value>*.

Tip: You can use the `--list-attributes` option to display a list of available attributes and the `--help-attribute` to get more detailed information about a specific attribute.

--by-interface *<interface>*
selects devices by network interface, for example, encf5a0. The *<interface>* parameter must be the name of an existing networking interface.

--by-node *<node>*
selects devices by device node, for example, `/dev/sda`. The *<node>* must be the path to a block device or character device special file.

Note: If *<node>* is the device node for a logical device (such as a device mapper device), **`lszdev`** tries to resolve the corresponding physical device nodes. The **`lsblk`** tool must be available for this resolution to work.

--by-path *<path>*
selects devices by file-system path, for example, `/usr`. The *<path>* parameter can be the mount point of a mounted file system, or a path on that file system.

Note: If the file system that provides *<path>* is stored on multiple physical devices (such as supported by btrfs), **`lszdev`** tries to resolve the corresponding physical device nodes. The **`lsblk`** tool must be available and the file system must provide a valid UUID for this resolution to work.

--configured
narrows the selection to those devices for which a persistent configuration exists.

--existing
narrows the selection to devices that are present in the active configuration.

--online
narrows the selection to devices that are enabled in the active configuration.

--offline
narrows the selection to devices that are disabled in the active configuration.

-a or --active
lists information obtained from the active configuration, that is, information from the running system.

-p or --persistent
lists information from the persistent configuration.

--auto-conf
lists information from the auto-configuration, see Chapter 3, “Device auto-configuration for Linux in LPAR mode,” on page 21.

-i or --info
displays detailed information about the configuration of the selected device or device type. Use `-ii` for even more information.

-c or --columns *<columns>*
specifies a comma-separated list of columns to display.

Example:

```
# lszdev --columns TYPE,ID
```

Tip: To get a list of supported column names, use the `--list-columns` option.

-n or --no-headings
suppresses column headings for list output.

--pairs
produces output in *<key="value">* format. Use this option to generate output in a format more suitable for processing by other programs. In this format, column values are prefixed with the name of the corresponding column. Values are enclosed in double quotation marks. The **lszdev** command automatically escapes quotation marks and slashes that are part of the value string.

--base *<path>* | *<key=value>*
changes file system paths that are used to access files. If *<path>* is specified without an equal sign (=), it is used as base path for accessing files in the active and persistent configuration. If the specified parameter is in *<key=value>* format, only those paths that begin with *<key>* are modified. For these paths, the initial *<key>* portion is replaced with *<value>*.

Example:

```
# lszdev --active --base /etc=/mnt/etc
```

-t or --type *<device_type>*
lists information about a device type. Use this option to display configuration information of a device type instead of a device.

-q or --quiet
prints only minimal run-time information.

-V or --verbose
prints additional run-time information.

-L or --list-types
lists all available device types that you can use with the `--type` option.

-l or --list-columns
lists all available columns that you can use with the `--columns` option.

-h or --help
displays help information for the command.

-v or --version
displays the version number of **lszdev**, then exits.

Input files

The **lszdev** command uses these input files:

/etc/udev/rules.d/
lszdev reads udev rules that represent the persistent configuration of devices from this directory. The udev rules are named 41-*<device subtype>*-*<id>*.rules.

/etc/modprobe.d/
lszdev reads modprobe configuration files that represent the persistent configuration of certain device types from this directory. File names start with `s390x-`.

/usr
lszdev reads udev rules that represent the auto-configuration of devices from this directory.

Examples

- To display a list of all devices in the currently active configuration:

```
# lszdev --active
```

- To return type and ID of root device in machine-readable format in the currently active configuration:

```
# lszdev --active --columns TYPE,ID --by-path /
```

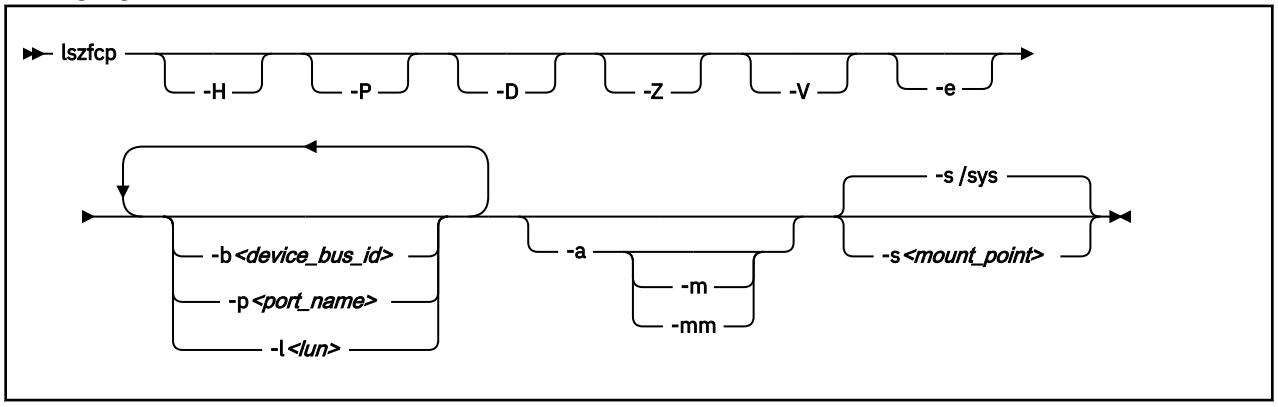
- To display DASD driver settings in the currently active configuration:

```
# lszdev --active --type dasd
```

lszfcp - List zfcp devices

Use the **lszfcp** command to gather information about zfcp devices, ports, units, and their associated class devices from sysfs and to display it in a summary format.

lszfcp syntax



Where:

-H or --hosts
shows information about hosts.

-P or --ports
shows information about ports.

-D or --devices
shows information about SCSI devices.

-a or --attributes
shows all attributes (implies **-V**).

-V or --verbose
shows sysfs paths of associated class and bus devices.

-e or --extended
generates extended output.

-b *<device_bus_id>* or --busid=*<device_bus_id>*
limits the output to information about the specified device.

-p *<port_name>* or --wwpn=*<port_name>*
limits the output to information about the specified port name.

-l *<lun>* or --lun=*<lun>*
limits the output to information about the specified LUN.

-a or --attributes
shows the main attributes of the specified objects.

-m or --moreattrs
shows more attributes of the specified objects.

Specify twice (`-mm`) to show even more attributes for SCSI devices.

-s *<mount_point>* or --sysfs=*<mount_point>*
specifies the mount point for sysfs.

-v or --version
displays version information.

-h or --help
displays a short help text.

Examples

- This command displays information about all available hosts, ports, and SCSI devices.

```
# lszfcp -H -D -P
0.0.3d0c host0
0.0.500c host1
...
0.0.3c0c host5
0.0.3d0c/0x500507630300c562 rport-0:0-0
0.0.3d0c/0x50050763030bc562 rport-0:0-1
0.0.3d0c/0x500507630303c562 rport-0:0-2
0.0.500c/0x50050763030bc562 rport-1:0-0
...
0.0.3c0c/0x500507630303c562 rport-5:0-2
0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:1077035024
0.0.3d0c/0x500507630300c562/0x4010403300000000 0:0:0:1077100560
0.0.3d0c/0x50050763030bc562/0x4010403200000000 0:0:1:1077035024
0.0.3d0c/0x500507630303c562/0x4010403200000000 0:0:2:1077035024
0.0.500c/0x50050763030bc562/0x4010403200000000 1:0:0:1077035024
...
0.0.3c0c/0x500507630303c562/0x4010403200000000 5:0:2:1077035024
```

- This command shows SCSI devices and limits the output to the devices that are attached through the FCP device with bus ID 0.0.3d0c:

```
# lszfcp -D -b 0.0.3d0c
0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:1077035024
0.0.3d0c/0x500507630300c562/0x4010403300000000 0:0:0:1077100560
0.0.3d0c/0x50050763030bc562/0x4010403200000000 0:0:1:1077035024
0.0.3d0c/0x500507630303c562/0x4010403200000000 0:0:2:1077035024
```

mon_fsstatd – Monitor z/VM guest file system size

The **mon_fsstatd** command is a user space daemon that collects physical file system size data from Linux on z/VM.

The daemon periodically writes the data as defined records to the z/VM monitor stream using the monwriter character device driver.

You can start the daemon with the systemd service unit `mon_fsstatd`. When the daemon is started as a service unit, it reads the configuration file `/etc/sysconfig/mon_fsstatd`.

Before you begin:

- Install the monwriter device driver and set up z/VM to start the collection of monitor sample data. See Chapter 36, “Writing z/VM monitor records,” on page 419 for information about the setup for and usage of the monwriter device driver.
- Customize the configuration file `/etc/sysconfig/mon_fsstatd` if you plan to call it as a systemd service.

The following publications provide general information about DCSSs, DIAG x'DC', CP commands, and APPLDATA:

- See *z/VM: Saved Segments Planning and Administration*, SC24-6322 for general information about DCSSs.
- See *z/VM: CP Programming Services*, SC24-6272 for information about the DIAG x'DC' instruction.
- See *z/VM: CP Commands and Utilities Reference*, SC24-6268 for information about the CP commands.
- See *z/VM: Performance*, SC24-6301 for information about monitor APPLDATA.

You can run the **mon_fsstatd** command in two ways:

- Calling mon_fsstatd as a systemd service. This method reads the configuration file `/etc/sysconfig/mon_fsstatd`.
- Calling mon_fsstatd from a command line.

mon_fsstatd systemd service syntax

If you run the **mon_fsstatd** daemon as a systemd service unit, you configure the daemon through specifications in a configuration file.

Where:

start
starts monitoring of guest file system size, using the configuration in `/etc/sysconfig/mon_fsstatd`.

stop
stops monitoring of guest file system size.

status
shows current status of guest file system size monitoring.

restart
stops and restarts monitoring.

reload
reloads the configuration. Use **reload** to re-read the configuration file when it was changed.

enable
starts the service automatically at boot time.

disable
disables automatic start of the service at boot time.

Configuration file keywords

FSSTAT_INTERVAL="*<n>*"
specifies the wanted sampling interval in seconds.

Examples of systemd service unit use

This example configuration file for mon_fsstatd (`/etc/sysconfig/mon_fsstatd`) sets the sampling interval to 30 seconds:

```
FSSTAT_INTERVAL="30"
```

Example of mon_fsstatd use. Note that your output can look different and include messages for other daemons, such as mon_procd:

- To start guest file system size monitoring:

```
# systemctl start mon_fsstatd
```

- To display the status:

```
# systemctl status mon_fsstatd
| mon_fsstatd.service - Monitor z/VM guest file system size
...
   Active: active (running) since Wed 2018-02-21 14:52:11 CET; 4s ago
```

- To stop guest file system size monitoring:

```
# systemctl stop mon_fsstatd
```

- To display the status again and check that monitoring is now stopped:

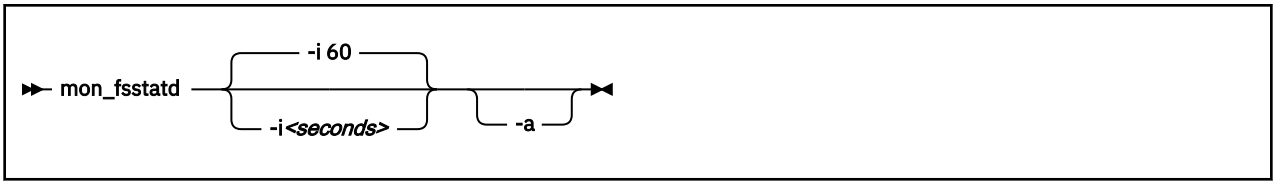
```
# systemctl status mon_fsstatd
| mon_fsstatd.service - Monitor z/VM guest file system size
...
   Active: inactive (dead)
...
```

- To restart the daemon and re-read the configuration file:

```
# systemctl restart mon_fsstatd
```

mon_fsstatd command-line syntax

If you call the **mon_fsstatd** daemon from the command line, you configure the daemon through command parameters.



Where:

-i or --interval *<seconds>*
specifies the wanted sampling interval in seconds.

-a or --attach
runs the daemon in the foreground.

-h or --help
displays help information for the command.

-v or --version
displays version information for the command.

Examples of command-line use

- To start mon_fsstatd with default setting:

```
> mon_fsstatd
```

- To start mon_fsstatd with a sampling interval of 30 seconds:

```
> mon_fsstatd -i 30
```

- To start mon_fsstatd and have it run in the foreground:

```
> mon_fsstatd -a
```

- To start mon_fsstatd with a sampling interval of 45 seconds and have it run in the foreground:

```
> mon_fsstatd -a -i 45
```

Processing monitor data

The mon_fsstatd daemon writes physical file system size data for Linux on z/VM to the z/VM monitor stream.

The following is the format of the file system size data that is passed to the z/VM monitor stream. One sample monitor record is written for each physical file system that is mounted at the time of the sample interval. The monitor data in each record contains a header consisting of a time stamp, the length of the data, and an offset. The header is followed by the file system data (as obtained from statvfs). The file system data fields begin with "fs_".

Table 79. File system size data format

Type	Name	Description
__u64	time_stamp	Time at which the file system data was sampled.
__u16	data_len	Length of data that follows the header.
__u16	data_offset	Offset from start of the header to start of file system data (that is, to the fields that begin with fs_).

Table 79. File system size data format (continued)

Type	Name	Description
__u16	fs_name_len	Length of the file system name. The file system name can be too long to fit in the monitor record. If so, this length is the portion of the name that is contained in the monitor record.
char [fs_name_len]	fs_name	The file system name. If the name is too long to fit in the monitor record, the name is truncated to the length in the fs_name_len field.
__u16	fs_dir_len	Length of the mount directory name. The mount directory name can be too long to fit in the monitor record. If so, this length is the portion of the name that is contained in the monitor record.
char[fs_dir_len]	fs_dir	The mount directory name. If the name is too long to fit in the monitor record, the name is truncated to the length in the fs_dir_len field.
__u16	fs_type_len	Length of the mount type. The mount type can be too long to fit in the monitor record. If so, this length is the portion that is contained in the monitor record.
char[fs_type_len]	fs_type	The mount type (as returned by getmntent). If the type is too long to fit in the monitor record, the type is truncated to the length in the fs_type_len field.
__u64	fs_bsize	File system block size.
__u64	fs_frsize	Fragment size.
__u64	fs_blocks	Total data blocks in file system.
__u64	fs_bfree	Free blocks in fs.
__u64	fs_bavail	Free blocks avail to non-superuser.
__u64	fs_files	Total file nodes in file system.
__u64	fs_ffree	Free file nodes in fs.
__u64	fs_favail	Free file nodes available to non-superuser.
__u64	fs_flag	Mount flags.

Use the time_stamp to correlate all file systems that were sampled in a given interval.

Reading the monitor data

All records that are written to the z/VM monitor stream begin with a product identifier.

The product ID is a 16-byte structure of the form pppppppffnvvrrmm, where for records that are written by mon_fsstatd, these values are:

ppppppp
is a fixed ASCII string LNXAPPL.

ff
is the application number for mon_fsstatd = x'0001'.

n
is the record number = x'00'.

vv
is the version number = x'0000'.

rr
is reserved for future use and should be ignored.

mm
is reserved for mon_fsstatd and should be ignored.

Note: Though the mod_level field (mm) of the product ID varies, there is no relationship between any particular mod_level and file system. The mod_level field should be ignored by the reader of this monitor data.

There are many tools available to read z/VM monitor data. One such tool is the Linux monreader character device driver. For more information about monreader, see Chapter 37, “Reading z/VM monitor records,” on page 423.

mon_procd – Monitor Linux on z/VM

The **mon_procd** command is a user space daemon that gathers system summary information and information about up to 100 concurrent processes on Linux on z/VM.

The daemon writes this data to the z/VM monitor stream using the monwriter character device driver. You can start the daemon as a systemd service `mon_procd` or call it manually. When it is called as a systemd service unit, it reads the configuration file `/etc/sysconfig/mon_procd`.

Before you begin:

- Install the monwriter device driver and set up z/VM to start the collection of monitor sample data. See Chapter 36, “Writing z/VM monitor records,” on page 419 for information about the setup for and usage of the monwriter device driver.
- Customize the configuration file `/etc/sysconfig/mon_procd` if you plan to call it as a systemd service unit.
- The Linux instance on which the mon_procd deamon runs requires a z/VM guest virtual machine with the OPTION APPLMON statement in the CP directory entry.

The following books provide general information about DCSSs, CP commands, and APPLDATA:

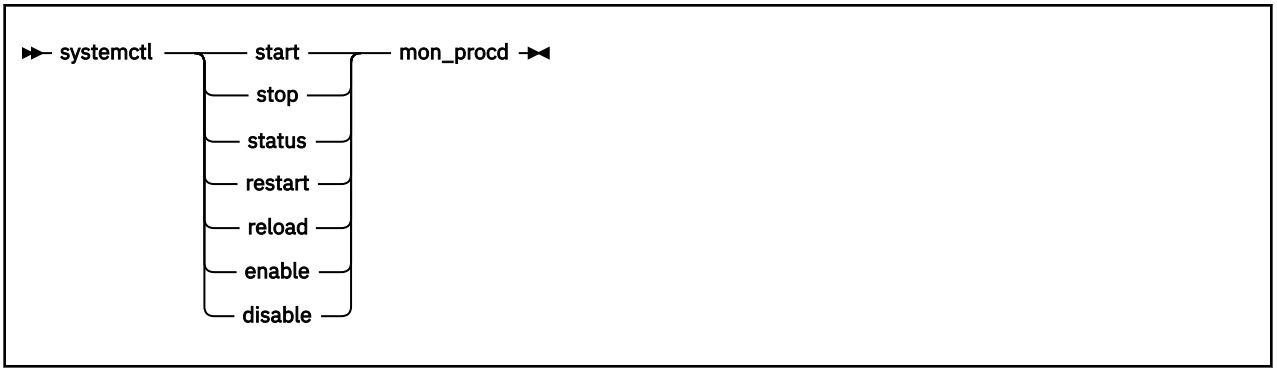
- See *z/VM: Saved Segments Planning and Administration*, SC24-6322 for general information about DCSSs.
- See *z/VM: CP Commands and Utilities Reference*, SC24-6268 for information about the CP commands.
- See *z/VM: Performance*, SC24-6301 for information about monitor APPLDATA.

You can run the **mon_procd** command in two ways:

- Calling **mon_procd** as a systemd service. This method reads the configuration file `/etc/sysconfig/mon_procd`.
- Calling **mon_procd** manually from a command line.

mon_procd systemd service syntax

If you run the **mon_procd** daemon as a systemd service unit, you configure the daemon through specifications in a configuration file.



Where:

start
starts monitoring of guest process data, using the configuration in `/etc/sysconfig/mon_procd`.

stop
stops monitoring of guest process data.

status
shows current status of guest process data monitoring.

restart
stops and restarts guest process data monitoring.

reload
reloads the configuration. Use **`reload`** to re-read the configuration file when it was changed.

enable
starts the service automatically at boot time.

disable
disables automatic start of the service at boot time.

Configuration file keywords

PROC_INTERVAL="*<n>*"
specifies the wanted sampling interval in seconds.

PROC="yes | no"
specifies whether to enable the mon_procd daemon. Set to "yes" to enable the daemon. Anything other than "yes" is interpreted as "no".

Examples of systemd service unit use

This example configuration file for mon_procd (`/etc/sysconfig/mon_procd`) sets the process monitoring interval to 60 seconds:

```
PROC_INTERVAL=60
```

Examples of mon_procd use:

- To start guest process data monitoring:

```
# systemctl start mon_procd
```

- To display the status:

```
# systemctl status mon_procd
| mon_procd.service - Monitor Linux on z/VM
...
   Active: active (running) since Mon 2018-02-26 12:16:00 CET; 4s ago
```

- To stop guest process data monitoring:

```
# systemctl stop mon_procd
```

- To display the status again and check that monitoring is now stopped:

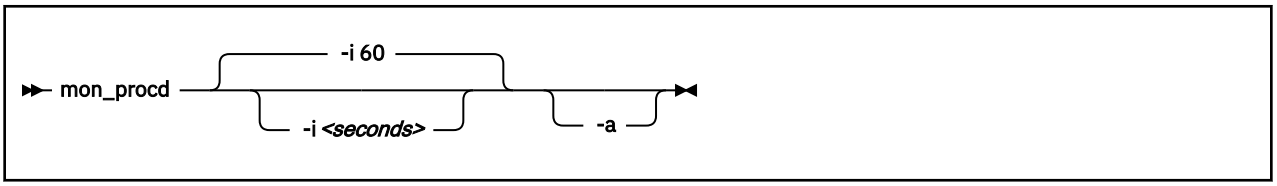
```
# systemctl status mon_procd
| mon_procd.service - Monitor Linux on z/VM
...
   Active: inactive (dead)
...
```

- To restart the daemon and re-read the configuration file:

```
# systemctl restart mon_procd
```

mon_procd command-line syntax

If you call the **mon_procd** daemon from the command line, you configure the daemon through command parameters.



Where:

-i *<seconds>* or --interval *<seconds>*
specifies the wanted sampling interval in seconds.

-a or --attach
runs the daemon in the foreground.

-h or --help
displays help information for the command.

-v or --version
displays version information for the command.

Examples of command-line use

- To start mon_procd with default setting:

```
> mon_procd
```

- To start mon_procd with a sampling interval of 30 seconds:

```
> mon_procd -i 30
```

- To start mon_procd and have it run in the foreground:

```
> mon_procd -a
```

- To start mon_procd with a sampling interval of 45 seconds and have it run in the foreground:

```
> mon_procd -a -i 45
```

Processing monitor data

The mon_procd daemon writes process data to the z/VM monitor stream.

The data includes system summary information and information of each process for up to 100 processes currently being managed by an instance of Linux on z/VM to the z/VM monitor stream. At the time of the sample interval, one sample monitor record is written for system summary data. Then, one sample monitor record is written for each process for up to 100 processes currently being managed by the Linux instance. If more than 100 processes exist in a Linux instance at a given time, processes are sorted by the sum of CPU and memory usage percentage values. Only the top 100 processes' data is written to the z/VM monitor stream.

The monitor data in each record begins with a header (a time stamp, the length of the data, and the offset). The data after the header depends on the field "record number" of the 16-bit product ID and can be summary data or process data. See “Reading the monitor data” on page 704 for details. The following is the format of system summary data that is passed to the z/VM monitor stream.

Table 80. System summary data format

Type	Name	Description
__u64	time_stamp	Time at which the process data was sampled.
__u16	data_len	Length of data that follows the header.
__u16	data_offset	Offset from start of the header to the start of the process data.
__u64	uptime	Uptime of the Linux instance.
__u32	users	Number of users on the Linux instance.
char[6]	loadavg_1	Load average over the last 1 minute.
char[6]	loadavg_5	Load average over the last 5 minutes.
char[6]	loadavg_15	Load average over the last 15 minutes.
__u32	task_total	total number of tasks on the Linux instance.
__u32	task_running	Number of running tasks.
__u32	task_sleeping	Number of sleeping tasks.
__u32	task_stopped	Number of stopped tasks.
__u32	task_zombie	Number of zombie tasks.
__u32	num_cpus	Number of CPUs.
__u16	puser	A number that represents (100 * percentage of total CPU time used for normal processes executing in user mode).
__u16	pnice	A number that represents (100 * percentage of total CPU time used for niced processes executing in user mode).
__u16	psystem	A number that represents (100 * percentage of total CPU time used for processes executing in kernel mode).
__u16	pidle	A number that represents (100 * percentage of total CPU idle time).
__u16	piowait	A number that represents (100 * percentage of total CPU time used for I/O wait).
__u16	pirq	A number that represents (100 * percentage of total CPU time used for interrupts).
__u16	psoftirq	A number that represents (100 * percentage of total CPU time used for softirqs).

Table 80. System summary data format (continued)		
Type	**Name**	**Description**
__u16	psteal	A number that represents (100 * percentage of total CPU time spent in stealing).
__u64	mem_total	Total memory in KB.
__u64	mem_used	Used memory in KB.
__u64	mem_free	Free memory in KB.
__u64	mem_buffers	Memory in buffer cache in KB.
__u64	mem_pgpgin	Data that is read from disk in KB.
__u64	mem_pgpgout	Data that is written to disk in KB
__u64	swap_total	Total swap memory in KB.
__u64	swap_used	Used swap memory in KB.
__u64	swap_free	Free swap memory in KB.
__u64	swap_cached	Cached swap memory in KB.
__u64	swap_pswpin	Pages that are swapped in.
__u64	swap_pswpout	Pages that are swapped out.

The following is the format of a process information data that is passed to the z/VM monitor stream.

Table 81. Process data format		
Type	**Name**	**Description**
__u64	time_stamp	Time at which the process data was sampled.
__u16	data_len	Length of data following the header.
__u16	data_offset	Offset from start of the header to the start of the process data.
__u32	pid	ID of the process.
__u32	ppid	ID of the process parent.
__u32	euid	Effective user ID of the process owner.
__u16	tty	Device number of the controlling terminal or 0.
__s16	priority	Priority of the process
__s16	nice	Nice value of the process.
__u32	processor	Last used processor.
__u16	pcpu	A number that represents (100 * percentage of the elapsed cpu time used by the process since last sampling).
__u16	pmem	A number that represents (100 * percentage of physical memory used by the process).
__u64	total_time	Total cpu time the process used.
__u64	ctotal_time	Total cpu time the process and its dead child processes used.
__u64	size	Total virtual memory that is used by the task in KB.
__u64	swap	Swapped out portion of the virtual memory in KB.

Table 81. Process data format (continued)		
Type	**Name**	**Description**
__u64	resident	Non-swapped physical memory that is used by the task in KB.
__u64	trs	Physical memory that is devoted to executable code in KB.
__u64	drs	Physical memory that is devoted to other than executable code in KB.
__u64	share	Shared memory that is used by the task in KB.
__u64	dt	Dirty page count.
__u64	maj_flt	Number of major page faults occurred for the process.
char	state	Status of the process.
__u32	flags	The process current scheduling flags.
__u16	ruser_len	Length of real user name of the process owner and should not be larger than 64.
char[ruser_len]	ruser	Real user name of the process owner. If the name is longer than 64, the name is truncated to the length 64.
__u16	euser_len	Length of effective user name of the process owner and should not be larger than 64.
char[euser_len]	euser	Effective user name of the process owner. If the name is longer than 64, the name is truncated to the length 64.
__u16	egroup_len	Length of effective group name of the process owner and should not be larger than 64.
char[egroup_len]	egroup	Effective group name of the process owner. If the name is longer than 64, the name is truncated to the length 64.
__u16	wchan_len	Length of sleeping in function's name and should not be larger than 64.
char[wchan_len]	wchan_name	Name of sleeping in function or '-'. If the name is longer than 64, the name is truncated to the length 64.
__u16	cmd_len	Length of command name or program name that is used to start the process and should not be larger than 64.
char[cmd_len]	cmd	Command or program name that is used to start the process. If the name is longer than 64, the name is truncated to the length 64.
__u16	cmd_line_len	Length of command line that is used to start the process and should not be larger than 1024.
char [cmd_line_len]	cmd_line	Command line that is used to start the process. If the name is longer than 1024, the name is truncated to the length 1024.

Use the time_stamp to correlate all process information that were sampled in a given interval.

Reading the monitor data

All records written to the z/VM monitor stream begin with a product identifier.

The product ID is a 16-byte structure of the form pppppppffnvvrrmm, where for records that are written by mon_procd, these values are:

ppppppp
is a fixed ASCII string LNXAPPL.

ff
is the application number for mon_procd = x'0002'.

n
is the record number as follows:

- x'00' indicates summary data.
- x'01' indicates process data.

vv
is the version number = x'0000'.

rr
is the release number, which can be used to mark different versions of process APPLDATA records.

mm
is reserved for mon_procd and should be ignored.

Note: Though the mod_level field (mm) of the product ID varies, there is no relationship between any particular mod_level and process. The mod_level field should be ignored by the reader of this monitor data.

This item uses at most 101 monitor buffer records from the monwriter device driver. A maximum number of buffers is set when a monwriter module is loaded. Because of this, the maximum number of buffers must not be less than the sum of buffer records that are used by all monwriter applications.

There are many tools available to read z/VM monitor data. One such tool is the Linux monreader character device driver. For more information about monreader, see Chapter 37, “Reading z/VM monitor records,” on page 423.

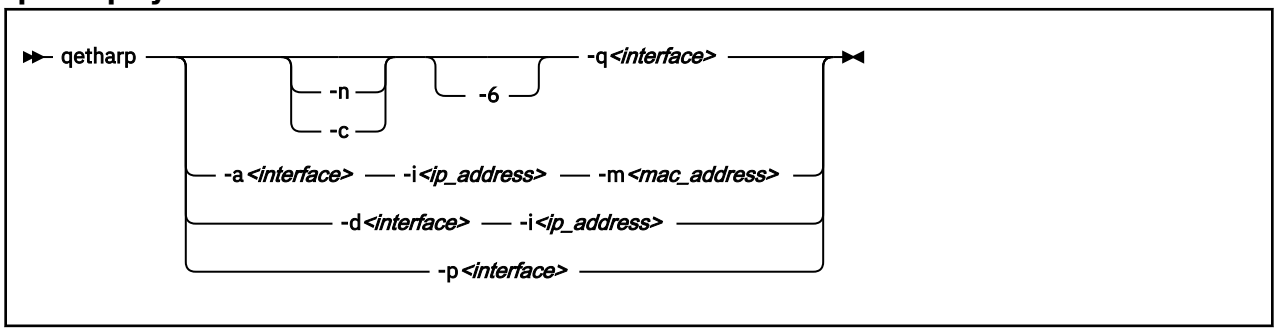
qetharp - Query and purge OSA and HiperSockets ARP data

Use the **qetharp** command to query and purge address data such as MAC and IP addresses from the ARP cache of the OSA and HiperSockets hardware.

Before you begin:

- The **qetharp** command applies only to devices in layer 3 mode (see "Layer 2 and layer 3" on page 247).
- The **qetharp** command supports IPv6 only for real HiperSockets and z/VM guest LAN HiperSockets.
- ForHiperSockets, z/VM guest LAN and VSWITCH interfaces, the **qetharp** command supports only the **--query** option.

qetharp syntax



Where:

-q or --query
shows the address resolution protocol (ARP) information about the specified network interface. Depending on the device that the interface was assigned to, this information is obtained from an OSA feature's ARP cache or a HiperSockets ARP cache.

The default command output shows symbolic host names and includes only numerical addresses for host names that cannot be resolved. Use the **-n** option to show numerical addresses instead of host names.

By default, qetharp omits IPv6 related information. Use the **-6** option to include IPv6 information for HiperSockets.

-n or --numeric
shows numeric addresses instead of trying to determine symbolic host names. This option can be used only with the **-q** option.

-c or --compact
limits the output to numeric addresses only. This option can be used only with the **-q** option.

-6 or --ipv6
includes IPv6 information for HiperSockets. For real HiperSockets, shows the IPv6 addresses. For guest LAN HiperSockets, shows the IPv6 to MAC address mappings. This option can be used only with the **-q** option.

<interface>
specifies the qeth interface to which the command applies.

-a or --add
adds a static ARP entry to the OSA adapter. Static entries can be deleted with **-d**.

-d or --delete
deletes a static ARP entry from the OSA adapter. Static entries are created with **-a**.

-p or --purge
flushes the ARP cache of the OSA. The cache contains dynamic ARP entries, which the OSA adapter creates through ARP queries. After flushing the cache, the OSA adapter creates new dynamic entries. This option works only with OSA devices. qetharp returns immediately.

-i *<ip_address>* or --ip *<ip_address>*
specifies the IP address to be added to or removed from the OSA adapter.

-m *<mac_address>* or --mac *<mac_address>*
specifies the MAC address to be added to the OSA adapter.

-v or --version
shows version information and exits

-h or --help
displays usage information and exits. To view the man page, enter **man qetharp**.

Examples

- Show all ARP entries of the OSA defined as encf500:

```
# qetharp -q encf500
```

- Show all ARP entries of the HiperSockets interface that is defined as enca1c0 including IPv6 entries:

```
qetharp -6q enca1c0
```

- Show all ARP entries of the OSA defined as encf500, without resolving host names:

```
# qetharp -nq encf500
```

- Show all ARP entries, including IPv6 entries, of the HiperSockets interface that is defined as enca1c0 without resolving host names:

```
qetharp -n6q enca1c0
```

- Flush the OSA ARP cache for encf500:

```
# qetharp -p encf500
```

- Add a static entry for encf500 and IP address 1.2.3.4 to the OSA ARP cache, with MAC address aa:bb:cc:dd:ee:ff:

```
# qetharp -a encf500 -i 1.2.3.4 -m aa:bb:cc:dd:ee:ff
```

- Delete the static entry for encf500 and IP address 1.2.3.4 from the OSA ARP cache.

```
# qetharp -d encf500 -i 1.2.3.4
```

qethconf - Configure qeth devices

Use the **qethconf** command to configure IP address takeover, virtual IP address (VIPA), and proxy ARP for layer3 qeth devices.

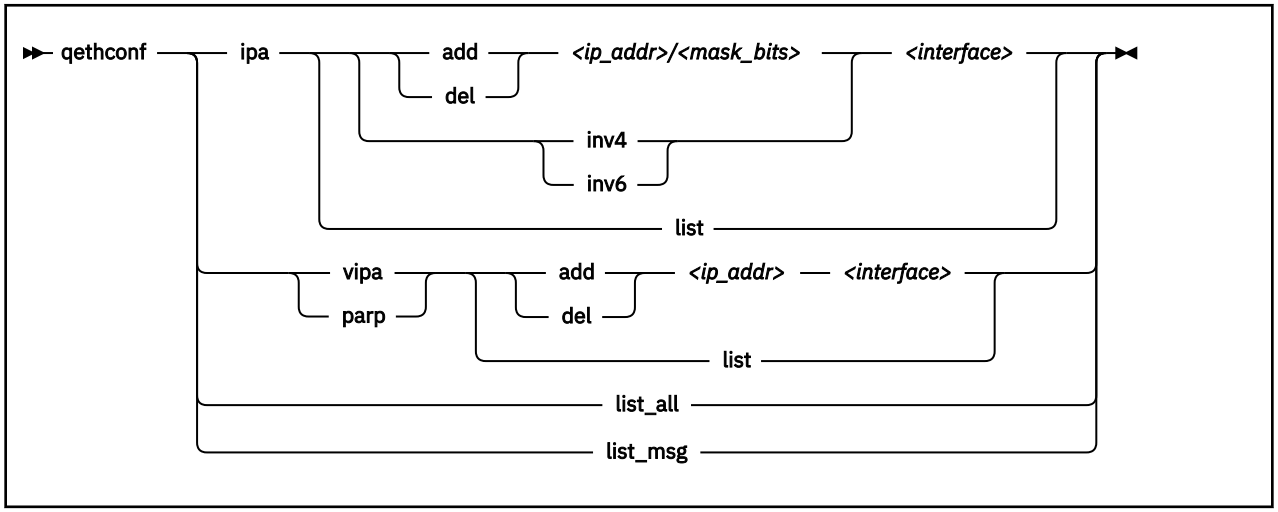
See Chapter 16, “qeth device driver for OSA-Express (QDIO) and HiperSockets,” on page 241 for details about the following concepts:

- IP address takeover
- VIPA (virtual IP address)
- Proxy ARP

You cannot use this command with the layer2 option.

From the arguments that are specified, **qethconf** assembles the function command and redirects it to the corresponding sysfs attributes. You can also use **qethconf** to list the already defined entries.

qethconf syntax



The **qethconf** command has these function keywords:

ipa
configures qeth for IP address takeover (IPA).

vipa
configures qeth for virtual IP address (VIPA).

parp or rxip
configures qeth for proxy ARP.

The **qethconf** command has these action keywords:

add
adds an IP address or address range.

del
deletes an IP address or address range.

inv4
inverts the selection of address ranges for IPv4 address takeover. This inversion makes the list of IP addresses that was specified with `qethconf add` and `qethconf del` an exclusion list.

inv6
inverts the selection of address ranges for IPv6 address takeover. This inversion makes the list of IP addresses that was specified with `qethconf add` and `qethconf del` an exclusion list.

list
lists existing definitions for specified qeth function.

list_all
lists existing definitions for IPA, VIPA, and proxy ARP.

<ip_addr>
IP address. Can be specified in one of these formats:

- IP version 4 format, for example, `192.168.10.38`
- IP version 6 format, for example, `FE80::1:800:23e7:f5db`
- 8- or 32-character hexadecimals prefixed with `-x`, for example, `-xc0a80a26`

<mask_bits>
specifies the number of bits that are set in the network mask. Enables you to specify an address range.

Example: A *<mask_bits>* of 24 corresponds to a network mask of `255.255.255.0`.

<interface>
specifies the name of the interface that is associated with the specified address or address range.

list_msg
lists **qethconf** messages and explanations.

-h or --help
displays help information.

-v or --version
displays version information.

Examples

- List existing proxy ARP definitions:

```
# qethconf parp list
parp add 1.2.3.4 encf500
```

- Assume responsibility for packages that are destined for 1.2.3.5:

```
# qethconf parp add 1.2.3.5 encf500
qethconf: Added 1.2.3.5 to /sys/class/net/encf500/device/rxip/add4.
qethconf: Use "qethconf parp list" to check for the result
```

Confirm the new proxy ARP definitions:

```
# qethconf parp list
parp add 1.2.3.4 encf500
parp add 1.2.3.5 encf500
```

- Configure encf500 for IP address takeover for all addresses that start with 192.168.10:

```
# qethconf ipa add 192.168.10.0/24 encf500
qethconf: Added 192.168.10.0/24 to /sys/class/net/encf500/device/ipa_takeover/add4
qethconf: Use "qethconf ipa list" to check for the result
```

Display the new IP address takeover definitions:

```
# qethconf ipa list
ipa add 192.168.10.0/24 encf500
```

- Configure VIPA for ence400:

```
# qethconf vipa add 10.99.3.3 ence400
qethconf: Added 10.99.3.3 to /sys/class/net/ence400/device/vipa/add4.
qethconf: Use "qethconf vipa list" to check for the result
```

Display the new VIPA definitions:

```
# qethconf vipa list
vipa add 10.99.3.3 ence400
```

- List all existing IPA, VIPA, and proxy ARP definitions.

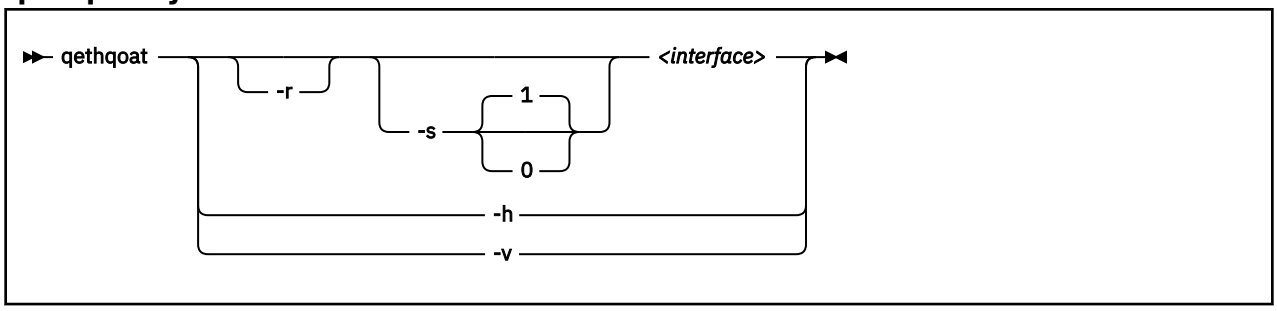
```
# qethconf list_all
parp add 1.2.3.4 encf500
parp add 1.2.3.5 encf500
ipa add 192.168.10.0/24 encf500
vipa add 10.99.3.3 ence400
```

qethqoat - Query OSA address table

Purpose

Use the **qethqoat** command to query the OSA address table and display physical and logical device information.

qethqoat syntax



where:

-r or --raw
writes raw data to stdout.

-s or --scope
defines the scope of the query. The following values are valid:

0
queries the level of the OSA address table

1
interface (this is the default)

<interface>
specifies the interface for which you want to display information.

-h or --help
displays help information. To view the man page, enter **man qethqoat**.

-v or --version
displays version information.

Examples

To display physical and logical device information for interface encf500, issue:

```
# qethqoat encf500
PCHID: 0x0310
CHPID: 0xa9
Manufacturer MAC address: 6c:ae:8b:48:0b:68
Configured MAC address: 00:00:00:00:00:00
Data device sub-channel address: 0xf402
CULA: 0x00
Unit address: 0x02
Physical port number: 0
Number of output queues: 1
Number of input queues: 1
Number of active input queues: 0
CHPID Type: OSD
Interface flags: 0x0a000000
OSA Generation: OSA-Express7S
Port speed/mode: 25 Gb/s / full duplex

Port media type: multi mode (SR/SX)
Jumbo frames: yes
Firmware: 0x00000c9a

IPv4 router: no
IPv6 router: no
IPv4 vmac router: no
IPv6 vmac router: no
Connection isolation: not active
Connection isolation VEPA: no
IPv4 assists enabled: 0x00111c77
IPv6 assists enabled: 0x00f15c60
IPv4 outbound checksum enabled: 0x0000003a
IPv6 outbound checksum enabled: 0x00000000
IPv4 inbound checksum enabled: 0x0000003a
IPv6 inbound checksum enabled: 0x00000000

IPv4 Multicast Address:                 MAC Address:
-----------------------                 ------------
224.0.0.1                               01:00:5e:00:00:01

IPv6 Address:                           IPA Flags:
-------------                           ----------
fe80::6cae:8b00:748:b68                 0x00000000

IPv6 Multicast Address:                 MAC Address:
-----------------------                 ------------
ff01::1                                 33:33:00:00:00:01
ff02::1                                 33:33:00:00:00:01
ff02::1:ff48:b68                        33:33:ff:48:0b:68
ff02::1:3                               33:33:00:01:00:03
```

This example uses scope 0 to query the supported OAT level and descriptor header types.

```
# qethqoat -s 0 encf500
Supported Scope mask: 0x00000001
Supported Descriptor hdr types: 0x0001070f
```

This example shows how the binary output from **qethqoat** can be processed in another tool. Here it is displayed in a hexdump viewer:

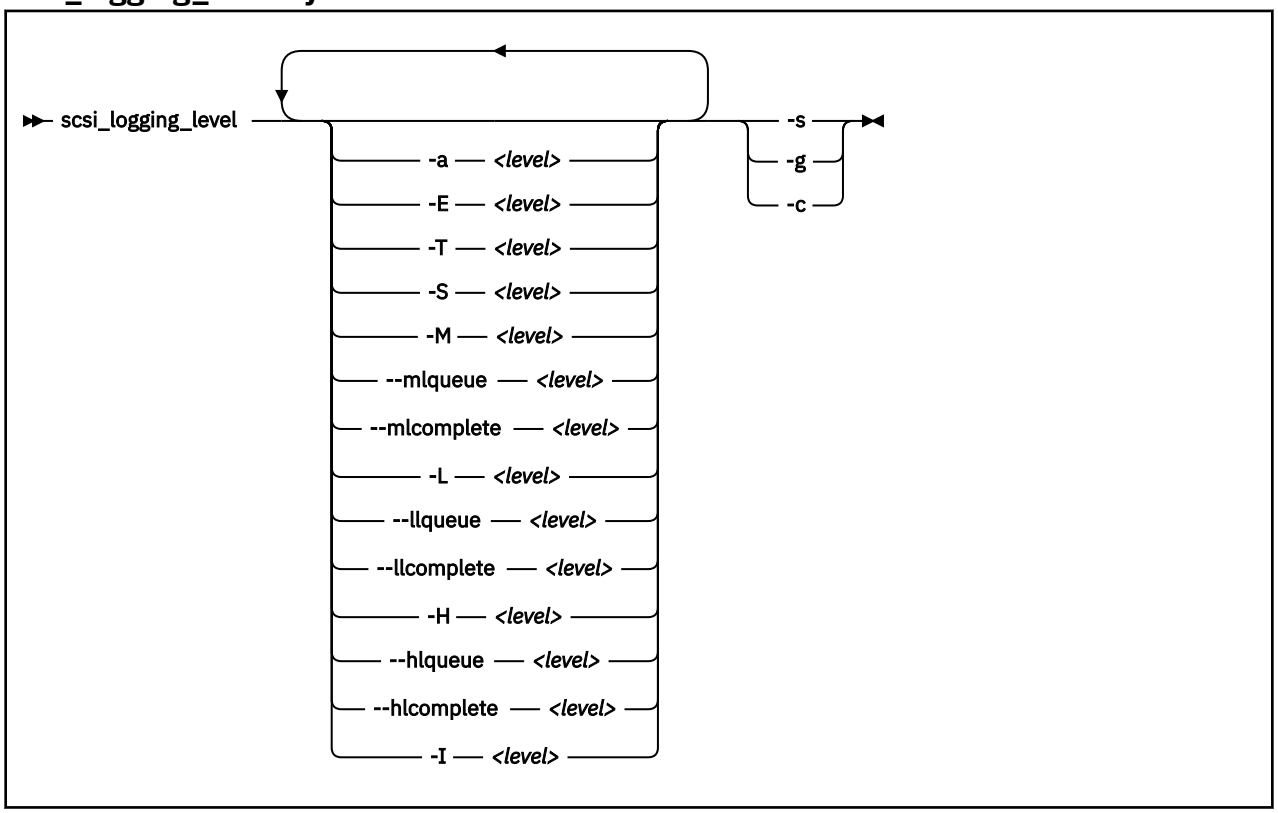
```
# qethqoat -r encf500 | hexdump
0000000 0158 0000 0008 0000 0000 0101 0000 0000
0000010 0000 0001 0000 0000 0000 0000 0000 0000
0000020 0004 0050 0001 0000 0000 0000 d7c8 4040
0000030 0120 0094 001a 643b 8a22 0000 0000 0000
0000040 e102 0002 0000 0004 0001 0000 0800 0000
0000050 0100 0480 0000 0766 0000 0000 0000 0000
0000060 0000 0000 0000 0000 0000 0000 0000 0000
0000070 0008 0060 0001 0000 0000 0000 d3c8 4040
0000080 0000 0000 0000 0000 0000 0000 0000 0000
0000090 0000 0000 0000 0000 0000 0000 0011 1c77
00000a0 0021 5c60 0000 001a 0000 0000 0000 001a
00000b0 0000 0000 0000 0000 0000 0000 0000 0000
00000c0 0002 0000 0000 0000 0000 0000 0000 0000
00000d0 0010 0030 0001 0000 0000 0000 c4c8 f4d4
00000e0 0000 0002 0000 0000 0000 0001 0000 0010
00000f0 0001 0001 0000 0000 0000 0000 0000 0000
0000100 e000 0001 0100 5e00 0001 0000 0000 0000
0000110 0010 0030 0001 0000 0000 0000 c4c8 f6d4
0000120 0000 0008 0000 0000 0000 0001 0000 0018
0000130 0001 0001 0000 0000 0000 0000 0000 0000
0000140 ff02 0000 0000 0000 0000 0000 0000 0001
0000150 3333 0000 0001 0000
0000158
```

scsi_logging_level - Set and get the SCSI logging level

Use the **scsi_logging_level** command to create, set, or get the SCSI logging level.

The SCSI logging feature is controlled by a 32-bit value – the SCSI logging level. This value is divided into 3-bit fields that describe the log level of a specific log area. Due to the 3-bit subdivision, setting levels or interpreting the meaning of current levels of the SCSI logging feature is not trivial. The scsi_logging_level script helps with both tasks.

scsi_logging_level syntax



Where:

-a *<level>* or --all *<level>*
specifies value for all SCSI_LOG fields.

-E *<level>* or --error *<level>*
specifies SCSI_LOG_ERROR.

-T *<level>* or --timeout *<level>*
specifies SCSI_LOG_TIMEOUT.

-S *<level>* or --scan *<level>*
specifies SCSI_LOG_SCAN.

-M *<level>* or --midlevel *<level>*
specifies SCSI_LOG_MLQUEUE and SCSI_LOG_MLCOMPLETE.

--mlqueue *<level>*
specifies SCSI_LOG_MLQUEUE.

--mlcomplete *<level>*
specifies SCSI_LOG_MLCOMPLETE.

-L *<level>* or --lowlevel *<level>*
specifies SCSI_LOG_LLQUEUE and SCSI_LOG_LLCOMPLETE.

--llqueue *<level>*
specifies SCSI_LOG_LLQUEUE.

--llcomplete *<level>*
specifies SCSI_LOG_LLCOMPLETE.

-H or --highlevel *<level>*
specifies SCSI_LOG_HLQUEUE and SCSI_LOG_HLCOMPLETE.

--hlqueue *<level>*
specifies SCSI_LOG_HLQUEUE.

--hlcomplete *<level>*
specifies SCSI_LOG_HLCOMPLETE.

-I *<level>* or --ioctl *<level>*
specifies SCSI_LOG_IOCTL.

-v or --version
displays version information.

-h or --help
displays help text.

-s or --set
creates and sets the logging level as specified on the command line.

-g or --get
gets the current logging level.

-c or --create
creates the logging level as specified on the command line.

You can specify several SCSI_LOG fields by using several options. When multiple options specify the same SCSI_LOG field, the most specific option has precedence.

Examples

- This command prints the logging word of the SCSI logging feature and each logging level.

```
#> scsi_logging_level -g
Current scsi logging level:
dev.scsi.logging_level = 0
SCSI_LOG_ERROR=0
SCSI_LOG_TIMEOUT=0
SCSI_LOG_SCAN=0
SCSI_LOG_MLQUEUE=0
SCSI_LOG_MLCOMPLETE=0
SCSI_LOG_LLQUEUE=0
SCSI_LOG_LLCOMPLETE=0
SCSI_LOG_HLQUEUE=0
SCSI_LOG_HLCOMPLETE=0
SCSI_LOG_IOCTL=0
```

- This command is useful to find issues with LUN discovery and SCSI error handling (recovery), such caused by dirty fibre optics, and has only negligible impact on regular I/O:

```
#> scsi_logging_level -s --mlcomplete 1 -T 7 -E 5 -S 7 -I 0 -a 0
New scsi logging level:
dev.scsi.logging_level = 4605
SCSI_LOG_ERROR=5
SCSI_LOG_TIMEOUT=7
SCSI_LOG_SCAN=7
SCSI_LOG_MLQUEUE=0
SCSI_LOG_MLCOMPLETE=1
SCSI_LOG_LLQUEUE=0
SCSI_LOG_LLCOMPLETE=0
SCSI_LOG_HLQUEUE=0
SCSI_LOG_HLCOMPLETE=0
SCSI_LOG_IOCTL=0
```

- This command sets all logging levels to 3:

```
#> scsi_logging_level -s -a 3
New scsi logging level:
dev.scsi.logging_level = 460175067
SCSI_LOG_ERROR=3
SCSI_LOG_TIMEOUT=3
SCSI_LOG_SCAN=3
SCSI_LOG_MLQUEUE=3
SCSI_LOG_MLCOMPLETE=3
SCSI_LOG_LLQUEUE=3
SCSI_LOG_LLCOMPLETE=3
SCSI_LOG_HLQUEUE=3
SCSI_LOG_HLCOMPLETE=3
SCSI_LOG_IOCTL=3
```

- This command sets SCSI_LOG_HLQUEUE=3, SCSI_LOG_HLCOMPLETE=2 and assigns all other SCSI_LOG fields the value 1.

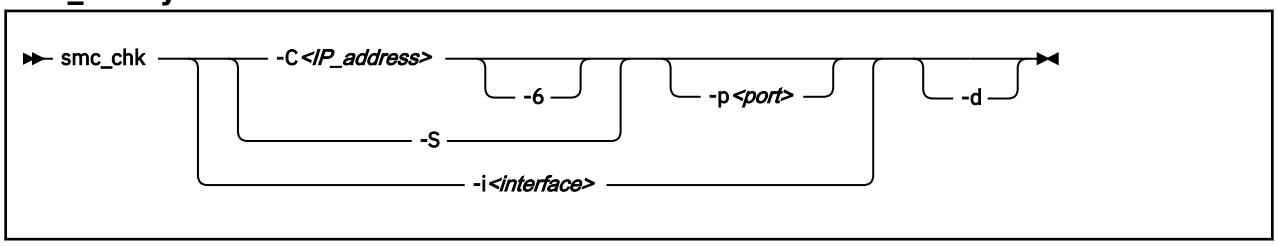
```
# scsi_logging_level --hlqueue 3 --highlevel 2 --all 1 -s
New scsi logging level:
dev.scsi.logging_level = 174363209
SCSI_LOG_ERROR=1
SCSI_LOG_TIMEOUT=1
SCSI_LOG_SCAN=1
SCSI_LOG_MLQUEUE=1
SCSI_LOG_MLCOMPLETE=1
SCSI_LOG_LLQUEUE=1
SCSI_LOG_LLCOMPLETE=1
SCSI_LOG_HLQUEUE=3
SCSI_LOG_HLCOMPLETE=2
SCSI_LOG_IOCTL=1
```

smc_chk - Verify SMC setups

Use the **smc_chk** command to verify that SMC-D or SMC-R connectivity is set up correctly and is operable.

The command runs a connectivity test, and displays errors if the test fails.

smc_chk syntax



Where:

-C or --connect *<IP_address>*
Obtains diagnostic information about the SMC support of a service that runs at a specified IP address. Use the -p option to specify a port.

-S or --server
Starts a server for manual tests. Use the -p option to specify a port.

-p or --port *<port>*
Specifies the port to use for tests. When you start a server by using -S, and if the specified port is in use, the next free port is chosen.

-i or --pnetid *<interface>*
Prints the PNET ID of the specified interface and exits.

-6 or --ipv6
The IP address that is provided is in IPv6 format.

-d or --debug
Shows debug messages.

-v or --version
Displays version information.

-h or --help
Displays a brief **smc_chk** usage information.

Examples

- To check whether a z/OS instance with IP address 192.168.37.1 is enabled for SMC using the 3270 console service that is running on port 23:

```
# smc_chk -C 192.168.37.1 -p 23
Test with target IP 192.168.37.1 and port 23
  Live test (SMC-D and SMC-R)
    Success, using SMC-D
```

- To print the PNET ID of interface encf5f0:

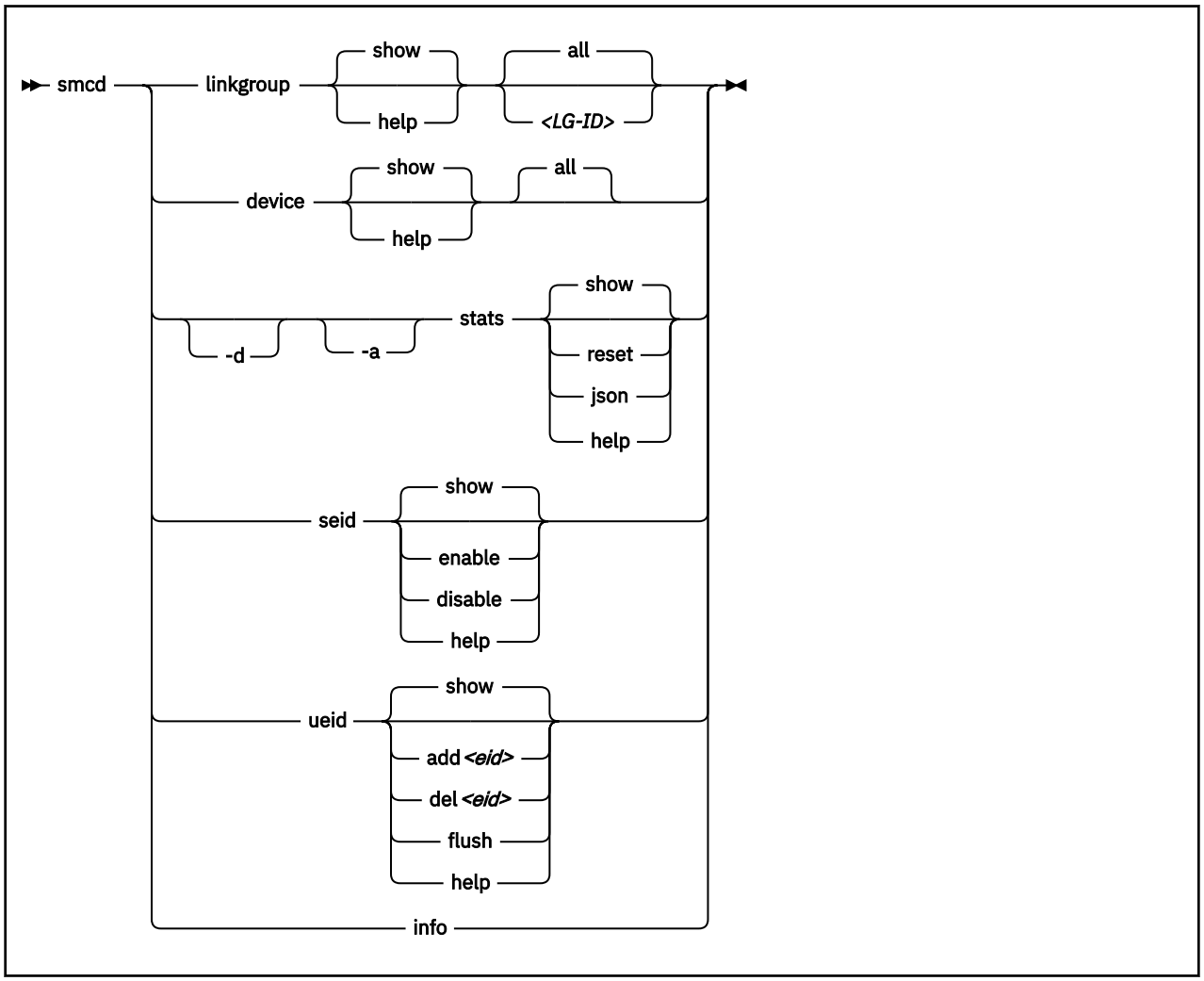
```
# smc_chk -i encf5f0
PNET5
```

- To start a server on port 45901 (or the next available free port):

```
smc_chk -S -p 45901
```

smcd - Display information about SMC-D link groups and devices

SMC-D connections are based on ISM devices, see Chapter 23, “Internal shared memory device driver,” on page 353.



Where:

Generic options
These options are available for all subcommands.

show
displays information about link groups, devices, or statistics, the system enterprise ID (EID), or user defined EIDs. This is the default option.

-h or --help
displays help information for the specified sub-command. To view the man page, enter **man smcd**.

linkgroup options
The **linkgroup** subcommand displays information about SMC-D link groups and links.

all
displays information about all link groups or devices.

<LG-ID>
limits the information to the specified link group.

device options
The **device** subcommand displays information about SMC-D devices.

all
displays information about all SMC-D devices. This is the default.

stats options
The **stats** subcommand displays statistics for SMC-D.

-d or --details
displays detailed SMC-D statistics.

-a or --absolute
ignores any counter resets and displays statistics beginning with smc module load.

reset
displays the current statistics and resets all SMC-D statistics counters to zero.

json
displays the current statistics in JSON format.

seid otions
The **seid** subcommand controls the system EID.

enable
uses the system EID for your Linux instance.

disable
does not use the system EID for your Linux instance. At least one user defined EID must exists. A disabled system EID is automatically enabled when the last user defined EID entry is deleted.

ueid options
the **ueid** subcommand manages user defined EIDs.

add *<ueid>*
add a user defined EID.

For *<ueid>*, specify up 32 uppercase alphabetic (A-Z) characters, numerals (0-9), hyphens (-), and dots (.). The first character must be alphanumeric, and dots must not be consecutive.

del *<ueid>*
delete the specified user defined EID.

flush
delete all user defined EIDs.

info
displays a summary of the SMC levels supported in the Linux kernel, and the capabilities of the hardware:

Kernel Capabilities
Shows the Linux kernel's SMC capabilities independently of any hardware prerequisites.

Hardware Capabilities
Shows the hardware's capabilities independently of support for it in Linux.

-v or --version
displays version information.

Output columns for linkgroup and device

In the output table, the columns headers have the following meanings:

FID
Function ID of the PCI device.

Type
Type of the underlying PCI device. For SMC-D connections, the type is ISM.

PCI-ID
ID of the PCI device.

PCHID
Physical channel ID of the PCI device.

InUse
Shows whether the ISM device is in use. The value can be:

Yes
At least one link group runs on the ISM device.

No
No link group runs on the ISM device.

#LGs
Number of link groups on the device.

PNET-ID
PNET ID of the device. A leading asterisk (*) means that the PNET ID is set by the user. For example, "*NET1".

Output of stats

For the output of the **stats** sub-command, see “Obtaining statistics for SMC connections” on page 345.

Examples

- To display all SMC-D devices, issue:

```
# smcd device
FID   Type PCI-ID        PCHID  InUse #LGs  PNET-ID
02e1  ISM  0002:00:00.0  07c2   No       0  NET1
```

  The same output results with the command: **smcd device show all**.
- To show SMC-D statistics:

```
# smcd stats
```

- To show detailed SMC-D statistics and reset SMC-D statistics counters:

```
# smcd -d stats reset
```

- To ignore any counter resets and show detailed SMC-D statistics since module load in JSON format:

```
# smcd -da stats json
```

- To show all user defined EIDs.

```
# smcd ueid
BUILDING-19
BUILDING-04
```

- To show the system EID.

```
# smcd seid IBM-SYSZ-ISMSEID000000002E488561 [enabled]
```

- To disable the system EID.

```
# smcd seid disable
```

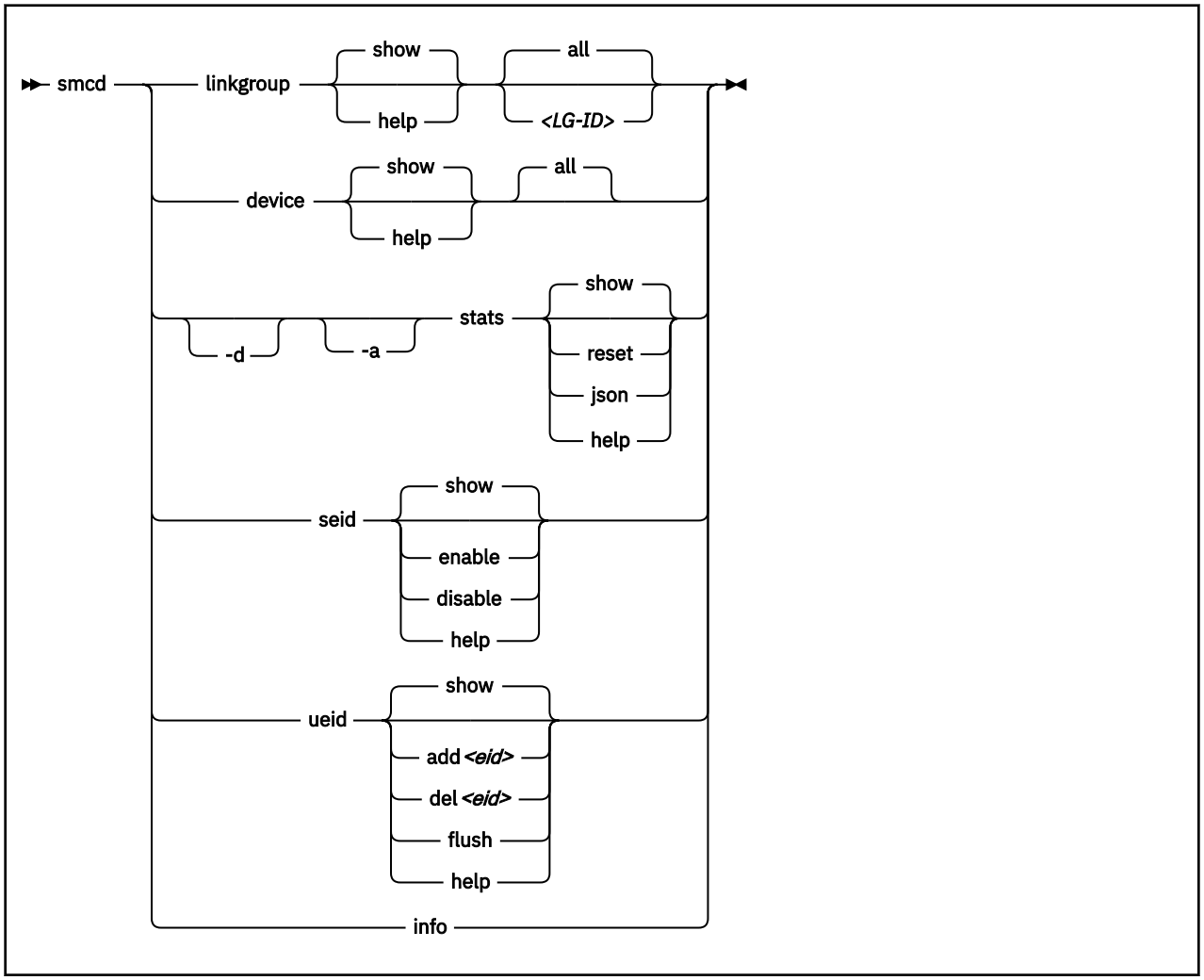
- To display a summary of SMC capabilities, issue:

```
# smcd info
   Kernel Capabilities
   SMC Version:      2.0
   SMC Hostname:     myHost
   SMC-D Features:   v1 v2
   SMC-R Features:   v1

   Hardware Capabilities
   SEID:             IBM-SYSZ-ISMSEID00000000XYZ
   ISM:              v1 v2
   RoCE:             n/a
```

The example shows that SMC-D version 1 and SMC-D version 2 could be used, as the kernel and ISM supports both versions. However, SMC-R would not be available: While the kernel supports it, the hardware is missing.

smcr - Display information about SMC-R link groups, links and devices

SMC-R connections are based on RoCE devices, see Chapter 22, “RDMA over Converged Ethernet,” on page 349.



Where:

Generic options
These options are available for all subcommands.

show
depending on the subcommand, displays information about link groups, devices, statistics, or user-defined enterprise IDs (EIDs). This is the default option.

-h or --help
displays help information for the specified sub-command. To view the man page, enter **man smcr**.

These options have the same meaning for all applicable subcommands.

-d or --details
displays detailed information about SMC-R link groups, devices, or statistics.

-dd or --ddetails
displays more details about SMC-R link groups and devices.

linkgroup options
The **linkgroup** subcommand displays information about SMC-R link groups and links.

link-show
displays information about a specified link, or all.

all
displays information about all link groups.

<LG-ID>
limits the displayed information to the link group with the specified ID. Combined with the `link-show` option, information is narrowed to links for the specified link group.

device options
The **`device`** subcommand displays information about SMC-R devices.

all
displays information about all SMC-R devices. This is the default.

ibdev <dev>
limits the command output to the device port with the specified RoCE device name.

netdev <dev>
limits the command output to the device with the specified network device name.

stats options
The **`stats`** subcommand displays statistics for SMC-R.

-a or --absolute
ignores any counter resets and displays statistics beginning with smc module load.

reset
displays the current statistics and resets all SMC-R statistics counters to zero.

json
displays the current statistics in JSON format.

ueid options
The **`ueid`** subcommand manages user defined EIDs.

add *<ueid>*
add a user defined EID.

For *<ueid>*, specify up 32 uppercase alphabetic characters (A-Z), numerals (0-9), hyphens (-), and dots (.). The first character must be alphanumeric, and dots must not be consecutive.

del *<ueid>*
delete the specified user defined EID.

flush
delete all user defined EIDs.

info
displays a summary of the SMC levels supported in the Linux kernel, and the capabilities of the hardware:

Kernel Capabilities
Shows the Linux kernel's SMC capabilities independently of any hardware prerequisites.

Hardware Capabilities
Shows the hardware's capabilities independently of support for it in Linux.

-v or --version
displays version information.

Output columns for linkgroup and device

In the output tables for **`smcr linkgroup`** and **`smcr device`**, the columns headers have the following meanings:

Net-Dev
Network device name.

IB-Dev
RoCE (InfiniBand) device name.

IB-P
InfiniBand port of the RoCE device. The port count starts with 1. Consequently, devices where each port is represented as a separate device indicate the port as the first port for all ports.

IB-State
State of the RoCE device port. The state can be INACTIVE or ACTIVE.

Type
Type of the underlying PCI device. For SMC-R, the type can be:

- RoCE_Express
- RoCE_Express 2

Crit
Show whether the device is critical, that is, without a failover possibility. The value can be:

Yes
At least one link group runs on the device with state "SINGLE" or locally "ASYMMETRIC", which means that one or more link groups lack a failover device.

No
No link group running on the device with state "SINGLE" or locally "ASYMMETRIC", which means that the link group or groups all have a fallback device.

FID
Function ID of the PCI device.

PCI-ID
ID of the PCI device.

PCHID
Physical channel ID of the PCI device.

#Links
Number of links on the device.

PNET-ID
PNET ID of the device. A leading asterisk (*) means that the PNET ID is set by the user. For example, "*NET1".

Output of stats

For the output of the **stats** sub-command, see “Obtaining statistics for SMC connections” on page 345.

Examples

- To display all SMC-R link groups, issue:

```
# smcr linkgroup show all
LG-ID    LG-Role LG-Type  VLAN #Conns  PNET-ID
00000100 CLNT    SYM         0      1  NET1
```

- To display all SMC-R links, issue:

```
# smcr linkgroup link-show all
LG-ID    LG-Role LG-Type Net-Dev    Link-State   #Conns
00000100 CLNT    SYM     ens281     LINK_ACTIVE       1
00000100 CLNT    SYM     enP1s282   LINK_ACTIVE       0
```

- To display SMC-R devices, issue:

```
# smcr device show all
Net-Dev  IB-Dev  IB-P   IB-State  Type          Crit #Links  PNET-ID
ens281   mlx4_0     1   ACTIVE    RoCE_Express   No       1  NET1
ens281d1 mlx4_0     2   INACTIVE  RoCE_Express   No       1  NET2
```

- To limit the output to device ens281, issue:

```
# smcr device show netdev ens281
Net-Dev IB-Dev   IB-P   IB-State  Type         Crit  #Links  PNET-ID
ens281  mlx4_0      1   ACTIVE    RoCE_Express   No       1  NET1
```

- To show SMC-R statistics:

```
# smcr stats
```

- To show detailed SMC-R statistics and reset SMC-R statistics counters:

```
# smcr -d stats reset
```

- To ignore any counter resets and show detailed SMC-R statistics since module load in JSON format:

```
# smcr -da stats json
```

- To add a user defined EID, GROUP-1.5.

```
# smcr ueid add GROUP-1.5
```

- To show all user defined EIDs.

```
# smcr ueid
ASSEMBLY5
GROUP-1.5
GROUP-7
```

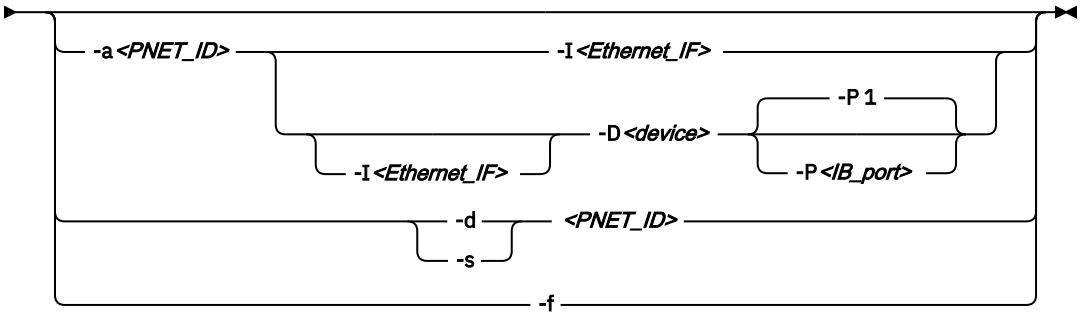
smc_pnet - Create network mapping table

Use the **smc_pnet** command to map a RoCE adapter port or ISM device to a standard Ethernet interface.

The SMC-R protocol requires grouping of standard Ethernet and RoCE networks or ISM devices. Such groups are called physical networks (PNETs). Within the same Converged Ethernet fabric, any available Ethernet interface can be combined with an available RDMA-capable network interface card or a DMA-capable ISM device.

Note: The mapping of a RoCE adapter port or ISM device to a standard Ethernet interface can be defined in the IOCDS or it can be defined as an entry in a PNET table. Only use the **smc_pnet** command if the IOCDS does not contain the required PNET IDs. IOCDS specifications override PNET table entries that are created with **smc_pnet**.

smc_pnet syntax



Enter **smc_pnet** without parameters to display all entries in the PNET table.

-a or --add *<PNET_ID>*
creates a new entry in the PNET table and allocates the specified ID, if it does not already exist. Only one entry can be defined for a specific Ethernet interface and a specific Infiniband device port or ISM device. A PNET ID consists of up to 16 alphanumeric uppercase characters without blanks.

-I or --interface *<Ethernet_IF>*
specifies the name of the Ethernet interface for a new PNET.

-D or --ibdevice *<device>*
specifies the name of the Infiniband device or ISM device for a new PNET.

-P or --ibport *<IB_port>*
Optional: specifies the port number of the Infiniband device port. Valid values are 1 or 2. The default value is 1.

-s or --show *<PNET_ID>*
displays the PNET table entry with the specified ID.

-d or --delete *<PNET_ID>*
deletes the PNET table entry with the specified ID.

-f or --flush
removes all entries from the PNET table.

-h or --help
displays help information for the command.

-v or --version
displays the version number of **smc_pnet**.

Examples

- To create a PNET with ID ABC for the Ethernet interface names encf500 and bond0, and add Infiniband device with ID 0001:00:0.0 on port number 2 and ISM device with ID 0004:00:00.0 on port 1:

```
# smc_pnet –a ABC –I encf500
# smc_pnet –a ABC –I bond0
# smc_pnet –a ABC –D 0001:00:00:00.0 –P 2
# smc_pnet –a ABC –D 0004:00:00:00.0
```

- To show all PNET entries:

```
# smc_pnet
ABC encf500 n/a 255
ABC bond0 n/a 255
ABC n/a 0001:00:00.0 2
ABC n/a 0004:00:00.0 1
```

- To define PNET ID XYZ for the Ethernet interface name vlan0201 and the InfiniBand device ID 0001:00:00.0 on port 1:

```
# smc_pnet –a XYZ –I vlan0201 –D 0001:00:00.0 –P 1
```

- To show all entries for PNET ID XYZ:

```
# smc_pnet –s XYZ
XYZ vlan0201 n/a 255
XYZ n/a 0001:00:00.0 1
```

- To delete a PNET table entry with PNET ID ABC:

```
# smc_pnet -d ABC
```

- To delete all entries in the PNET table:

```
# smc_pnet -f
```

For command return codes, see the man page.

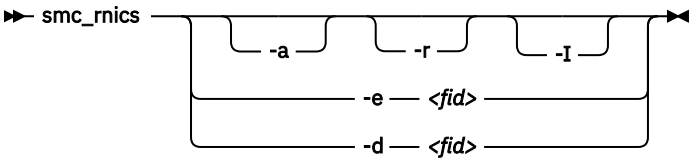
smc_rnics - list RoCE Express PCI functions and control their online state

Use **smc_rnics** to list RoCE Express PCI functions and to set these PCI functions online or offline.

Setting a PCI function offline on Linux in LPAR mode or in a DPM partition also deconfigures it in the partition.

Syntax

smc_rnics syntax



where:

-a or --all
lists all PCI functions, regardless of their online state. By default, only online PCI functions are listed.

-r or --rawids
displays the type as raw vendor and device IDs.

-I or --IB-dev
lists the PCI functions with their RDMA properties. Because RDMA is based on the InfiniBand (IB) communications standard, this parameter and output table columns with RDMA information use IB terminology. The default list shows the network device properties.

-e or --enable *<fid>*
sets the specified PCI function online. In the command, *<fid>* is the function ID in hexadecimal notation. Leading zeroes can be omitted.

-d or --disable *<fid>*
sets the specified PCI function offline. In the command, *<fid>* is the function ID in hexadecimal notation. Leading zeroes can be omitted.

-h or --help
displays help information for the **smc_rnics** command. To view the man page, issue **man smc_rnics**.

-v or --version
displays the version of the **smc_rnics** command.

Examples

- This example lists the online PCI functions.

```
# smc_rnics
  FID Power PCI_ID       PCHID Type          PPrt PNET_ID Net-Dev
------------------------------------------------------------------
  8ca 1     0008:00:00.0 01c8  RoCE_Express2 0    NET25   eno8
  8ea 1     0009:00:00.0 01c8  RoCE_Express2 1    NET26   eno9
```

- This example lists online and offline PCI functions.

```
# smc_rnics -a
  FID Power PCI_ID       PCHID Type          PPrt PNET_ID Net-Dev
------------------------------------------------------------------
  50a 0
  8ca 1     0008:00:00.0 01c8  RoCE_Express2 0    NET25   eno8
  8ea 1     0009:00:00.0 01c8  RoCE_Express2 1    NET26   eno9
```

- This example sets the PCI function with FID `0x050a` online.

```
# smc_rnics -e 50a
```

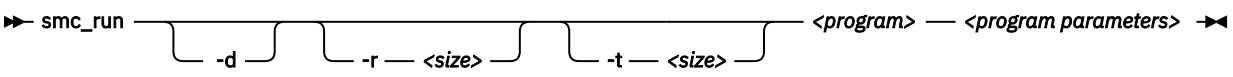
- This example lists the online PCI functions with their RDMA properties.

```
# smc_rnics -I
  FID Power PCI_ID       PCHID Type          IPrt PNET_ID IB-Dev
------------------------------------------------------------------
  50a 1     000a:00:00.0 01c8  RoCE_Express2 1    NET26   mlx5_0
  8ca 1     0008:00:00.0 01c8  RoCE_Express2 1    NET25   mlx5_1
  8ea 1     0009:00:00.0 01c8  RoCE_Express2 1    NET26   mlx5_2
```

smc_run - Run a TCP socket program with the SMC protocol using a preloaded library

Use the **smc_run** command to start a TCP socket program that uses SMC as the networking protocol.

smc_run syntax



Where:

smc_run *<program> <program_parameters>*
Starts the specified TCP socket program with the specified parameters, using the SMC protocol.

-d
Optional: Display diagnostic messages while the program is running.

-h
displays help information for the command.

-r *<size>*
requests a receive buffer with a specific size. Specify the size in bytes, use a suffix (k or K) for kilobytes, or use a suffix (m or M) for megabytes.

-t *<size>*
requests a transmit buffer with a specific size. Specify the size in bytes, use a suffix (k or K) for kilobytes, or use a suffix (m or M) for megabytes.

-v
displays the version of the **smc_run** command.

Examples

- To start a program called iperf3 with parameters "-s -p 12345":

```
# smc_run iperf3 -s -p 12345
-----------------------------------------------------------
Server listening on 12345
-----------------------------------------------------------
...
```

- To start a program called iperf3 with parameters "-s -p 12345" and diagnostic messages:

```
# smc_run -d iperf3 -s -p 12345
```

- To start a program called iperf3 with parameters "-s -p 12345", and with a request for a receive buffer of 16 KB and a transmit buffer of 512 KB:

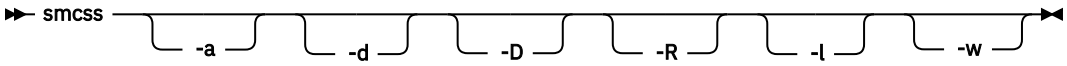
```
# smc_run -r 16384 -t 512k iperf3 -s -p 12345
```

For command return codes, see the man page.

smcss - Display information about the AF_SMC sockets and link groups

Use the **smcss** command to display information about the AF_SMC sockets and link groups.

smcss syntax



Entering **smcss** without any parameters displays a list of connecting, closing, or connected SMC sockets.

-a or --all
lists all SMC sockets: listening, opening, closing, and connected.

-d or --debug
displays debug information, such as the shutdown state.

-D or --smcd
lists SMC-D sockets only. Displays additional SMC-D specific information.

-R or --smcr
lists SMC-R sockets only. Displays additional SMC-R specific information.

-l or --listening
lists listening sockets only. These are omitted in the default listing.

-w or --wide
prevents truncation of IP addresses.

-h or --help
displays help information for the command.

-v or --version
displays the version number of **smcss**.

Meaning of the output fields

Entry	Values and meaning
State	**INIT** The SMC socket is being initialized. It is not connected nor listening yet. **CLOSED** The SMC socket is closed. It is not connected nor listening anymore. **LISTEN** The SMC socket is a listening socket, waiting for incoming connection requests. **ACTIVE** The SMC socket has an established connection. In this state, the TCP connection is fully established, rendezvous processing has been completed, and SMC peers can exchange data via RDMA. **PEERCLW1** No further data will be sent to the peer. **PEERCLW2** No further data will be sent to or received from the peer. **APPLCLW1** No further data will be received from the peer.

Entry	Values and meaning
	APPLCLW2 No further data will be received from or sent to the peer. **APPLFINCLW** The peer has closed the socket. **PEERFINCLW** The socket is closed locally. **PEERABORTW** The socket was abnormally closed locally. **PROCESSABORT** The peer has closed the socket abnormally.
Inode	denotes the inode of the SMC socket.
UID	denotes the unique ID of the SMC socket.
Local Address	denotes address and port number of the local end of the SMC socket. Trailing dots indicate a truncated address. Use the `-w` option to display full addresses.
Peer Address	denotes address and port number of the remote end of the socket.
Intf	denotes that if the socket is explicitly bound with **`setsockopt`** option SO_BINDTODEVICE, "Intf" shows the interface number of the Ethernet device to which the socket is bound.
Mode	can have the following values: **SMCD** The SMC socket uses SMC-D for data exchange. **SMCR** The SMC socket uses SMC-R for data exchange. **TCP** An SMC connection could not be established. The SMC socket uses the TCP protocol for data exchange.
ShutD	(shutdown) can take the following values: **<->** The SMC socket has not been shut down. **R->** The SMC socket is shut down one-way and cannot receive data. **<-W** The SMC socket is shut down one-way and cannot send data. **R-W** The SMC socket is shut down and cannot receive or send data.
Token	is a unique ID of the SMC socket connection.
Sndbuf	denotes the size of the to-be-sent window of the SMC socket connection.
Rcvbuf	denotes the size of the receiving window of the SMC socket connection (filled by peer).
Peerbuf	denotes the size of the peer receiving window of the SMC socket connection (to fill during data-transfer).
rxprod-Cursor	Describes the current cursor location of the Rcvbuf for data to be received from the peer.

Entry	Values and meaning
rxcons-Cursor	Describes the current cursor location of the Peerbuf for data sent to peer and confirmed by the peer.
rxFlags	SMC socket connection flags set by and received from the peer.
txprod-Cursor	Describes the current cursor location of the Peerbuf for data sent to peer.
txcons-Cursor	Describes the current cursor location of the Rcvbuf for data received from the peer and confirmed to the peer.
txFlags	SMC socket connection flags set locally and sent to the peer.
txprep-Cursor	Describes the current cursor location of the Sndbuf for data to be sent. The data is to be moved to the Peerbuf.
txsent-Cursor	Describes the current cursor location of the Sndbuf for data sent. The data was moved to the Peerbuf.
txfin-Cursor	Describes the current cursor location of the Sndbuf for data sent and send completion confirmed. The data was moved to the Peerbuf and completion was confirmed.
Role	"Role" can take the following values: **CLNT** The link group of the SMC socket is used for client connections. **SERV** The link group of the SMC socket is used for server connections.
IB-Device	Name of the RoCE device used by the link group to which the SMC socket belongs.
Port	Port of the RoCE device used by the link group to which the SMC socket belongs.
Linkid	unique link ID of the link within the link group to which the SMC socket belongs.
GID	Group identifier of the RoCE port used by the link group to which the SMC socket belongs.
Peer-GID	GID of the foreign RoCE port used by the link group to which the SMC socket belongs.

Examples

- To display information about all SMC sockets on the server:

```
[root@myserver]# smcss -a
State          UID   Inode   Local Address           Peer Address            Intf Mode
INIT           00000 0000000
ACTIVE         00000 0060177 ::ffff:10.100.80..:6668 ::ffff:10.100.8..:40812 0000 SMCD
ACTIVE         00000 0060173 ::ffff:10.100.80..:6668 ::ffff:10.100.8..:40804 0000 SMCD
LISTEN         00000 0059058 :::6668
```

- To list listening sockets on the server:

```
root@myserver>#  smcss -l
State          UID   Inode   Local Address           Peer Address            Intf Mode
LISTEN         00000 0059058 :::6668
```

- To display debug information about all SMC sockets on the server:

```
[root@myserver]# smcss -d
State          UID   Inode   Local Address           Peer Address            Intf Mode Shutd Token ...
ACTIVE         00000 0060177 ::ffff:10.100.80..:6668 ::ffff:10.100.8..:40812 0000 SMCD  <->  00...
ACTIVE         00000 0060173 ::ffff:10.100.80..:6668 ::ffff:10.100.8..:40804 0000 SMCD  <->  00...
...
```

For command return codes, see the man page.

tape390_crypt - Manage tape encryption

Use the **tape390_crypt** command to enable and disable tape encryption for a channel attached tape device. You can also to specify key encrypting keys (KEK) by using labels or hashes.

For 3592 tape devices, it is possible to write data in an encrypted format. The encryption keys are stored on an encryption key manager (EKM) server, which can run on any machine with TCP/IP and Java support. The EKM communicates with the tape drive over the tape control unit by using TCP/IP. The control unit acts as a proxy and forwards the traffic between the tape drive and the EKM. This type of setup is called out-of-band control-unit based encryption.

The EKM creates a data key that encrypts data. The data key itself is encrypted with KEKs and is stored in so called external encrypted data keys (EEDKs) on the tape medium.

You can store up to two EEDKs on the tape medium. With two EEDKs, one can contain a locally available KEK and the other can contain the public KEK of the location or company to where the tape is to be transferred. Then, the tape medium can be read in both locations.

When the tape device is mounted, the tape drive sends the EEDKs to the EKM. The EKM tries to unwrap one of the two EEDKs and sends back the extracted data key to the tape drive.

Linux can address KEKs by specifying either hashes or labels. Hashes and labels are stored in the EEDKs.

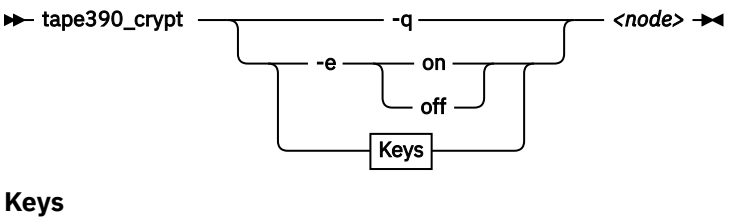
Note: If a tape is encrypted, it cannot be used for IPL.

Before you begin:

To use tape encryption, you need:

- A 3592 crypto-enabled tape device and control unit that is configured as system-managed encryption.
- A crypto-enabled 3590 channel-attached tape device driver. See Chapter 15, “Channel-attached tape device driver,” on page 229.
- A key manager. See *Encryption Key Manager Component for the Java(TM) Platform Introduction, Planning, and User's Guide*, GA76-0418 for more information.

tape390_crypt syntax



Notes:

[1] The -k or --key operand can be specified maximally twice.

Where:

-q or --query
displays information about the tape's encryption status. If encryption is active and the medium is encrypted, additional information about the encryption keys is displayed.

-e or --encryption
sets tape encryption on or off.

-k or --key
sets tape encryption keys. You can only specify the `-k` option if the tape medium is loaded and rewound. While processing the `-k` option, the tape medium is initialized and all previous data contained on the tape medium is lost.

You can specify the `-k` option twice because the tape medium can store two EEDKs. If you specify the `-k` option once, two identical EEDKs are stored.

<value>
specifies the key encrypting key (KEK), which can be up to 64 characters long. The keywords `label` or `hash` specify how the KEK in *<value>* is to be stored on the tape medium. The default store type is `label`.

-d or --delimiter
specifies the character that separates the KEK in *<value>* from the store type (`label` or `hash`). The default delimiter is ":" (colon).

<char>
is a character that separates the KEK in *<value>* from the store type (`label` or `hash`).

-f or --force
specifies that no prompt message is to be issued before writing the KEK information and initializing the tape medium.

<node>
specifies the device node of the tape device.

-h or --help
displays help text. For more information, enter the command `man tape390_crypt`.

-v or --version
displays information about the version.

Examples

The following scenarios illustrate the most common use of tape encryption. In all examples `/dev/ntibm0` is used as the tape device.

Querying a tape device before and after encryption is turned on

This example shows a query of tape device `/dev/ntibm0`. Initially, encryption for this device is off. Encryption is then turned on, and the status is queried again.

```
tape390_crypt -q /dev/ntibm0
ENCRYPTION: OFF
MEDIUM: NOT ENCRYPTED

tape390_crypt -e on /dev/ntibm0

tape390_crypt -q /dev/ntibm0
ENCRYPTION: ON
MEDIUM: NOT ENCRYPTED
```

Then, two keys are set, one in label format and one in hash format. The status is queried and there is now additional output for the keys.

```
tape390_crypt -k my_first_key:label -k my_second_key:hash /dev/ntibm0
--->> ATTENTION! <<---
All data on tape /dev/ntibm0 will be lost.
Type "yes" to continue: yes
SUCCESS: key information set.

tape390_crypt -q /dev/ntibm0
ENCRYPTION: ON
MEDIUM: ENCRYPTED
KEY1:
    value:     my_first_key
    type:     label
    ontape:     label
KEY2:
    value:     my_second_key
    type:     label
    ontape:     hash
```

Using default keys for encryption

1. Load the cartridge. If the cartridge is already loaded:
   - Switch off encryption:

     ```
     tape390_crypt -e off /dev/ntibm0
     ```

   - Rewind:

     ```
     mt -f /dev/ntibm0 rewind
     ```

2. Switch encryption on:

   ```
   tape390_crypt -e on /dev/ntibm0
   ```

3. Write data.

Using specific keys for encryption

1. Load the cartridge. If the cartridge is already loaded, rewind:

   ```
   mt -f /dev/ntibm0 rewind
   ```

2. Switch encryption on:

   ```
   tape390_crypt -e on /dev/ntibm0
   ```

3. Set new keys:

   ```
   tape390_crpyt -k key1 -k key2 /dev/ntibm0
   ```

4. Write data.

Writing unencrypted data

1. Load the cartridge. If the cartridge is already loaded, rewind:

   ```
   mt -f /dev/ntibm0 rewind
   ```

2. If encryption is on, switch off encryption:

   ```
   tape390_crypt -e off /dev/ntibm0
   ```

3. Write data.

Appending new files to an encrypted cartridge

1. Load the cartridge

2. Switch encryption on:

```
tape390_crypt -e on /dev/ntibm0
```

3. Position the tape.
4. Write data.

Reading an encrypted tape

1. Load the cartridge
2. Switch encryption on:

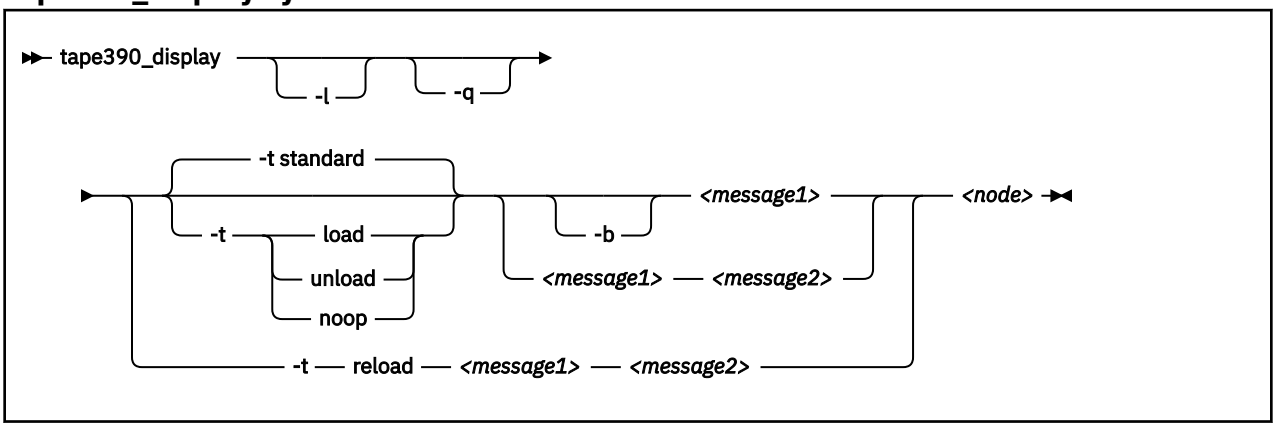
```
tape390_crypt -e on /dev/ntibm0
```

3. Read data.

tape390_display - Display messages on tape devices and load tapes

Use the **tape390_display** command to show messages on the display unit of a physical tape device, optionally in conjunction with loading a tape.

tape390_display syntax



Where:

-l or --load
instructs the tape unit to load the next indexed tape from the automatic tape loader (if installed). Ignored if no loader is installed or if the loader is not in "system" mode. The loader "system" mode allows the operating system to handle tape loads.

-t or --type
The possible values have the following meanings:

standard
displays the message or messages until the physical tape device processes the next tape movement command.

load
displays the message or messages until a tape is loaded; if a tape is already loaded, the message is ignored.

unload
displays the message or messages while a tape is loaded; if no tape is loaded, the message is ignored.

reload
displays the first message while a tape is loaded and the second message when the tape is removed. If no tape is loaded, the first message is ignored and the second message is displayed immediately. The second message is displayed until the next tape is loaded.

noop
is intended for test purposes only. It accesses the tape device but does not display the message or messages.

-b or --blink
causes *<message1>* to be displayed repeatedly for 2 seconds with a half-second pause in between.

<message1>
is the first or only message to be displayed. The message can be up to 8 byte.

<message2>
is a second message to be displayed alternately with the first, at 2-second intervals. The message can be up to 8 byte.

<node>
is a device node of the target tape device.

-q or --quiet
suppresses all error messages.

-h or --help
displays help text.

-v or --version
displays information about the version.

Note:

1. Symbols that can be displayed include:

   Alphabetic characters:
   A through Z (uppercase only) and spaces. Lowercase letters are converted to uppercase.

   Numeric characters:
   0 1 2 3 4 5 6 7 8 9

   Special characters:
   @ $ # , . / ' ( ) * & + - = % : _ < > ? ;

   The following are included in the 3490 hardware reference but might not display on all devices: | ¢
2. If only one message is defined, it remains displayed until the tape device driver next starts to move or the message is updated.
3. If the messages contain spaces or shell-sensitive characters, they must be enclosed in quotation marks.

Examples

The following examples assume that you are using standard devices nodes and not device nodes that are created by udev:

- Alternately display "BACKUP" and "COMPLETE" at 2-second intervals until device `/dev/ntibm0` processes the next tape movement command:

```
tape390_display BACKUP COMPLETE /dev/ntibm0
```

- Display the message "REM TAPE" while a tape is in the physical tape device followed by the message"NEW TAPE" until a new tape is loaded:

```
tape390_display --type reload "REM TAPE" "NEW TAPE" /dev/ntibm0
```

- Attempts to unload the tape and load a new tape automatically, the messages are the same as in the previous example:

```
tape390_display -l -t reload "REM TAPE" "NEW TAPE" /dev/ntibm0
```

tunedasd - Adjust low-level DASD settings

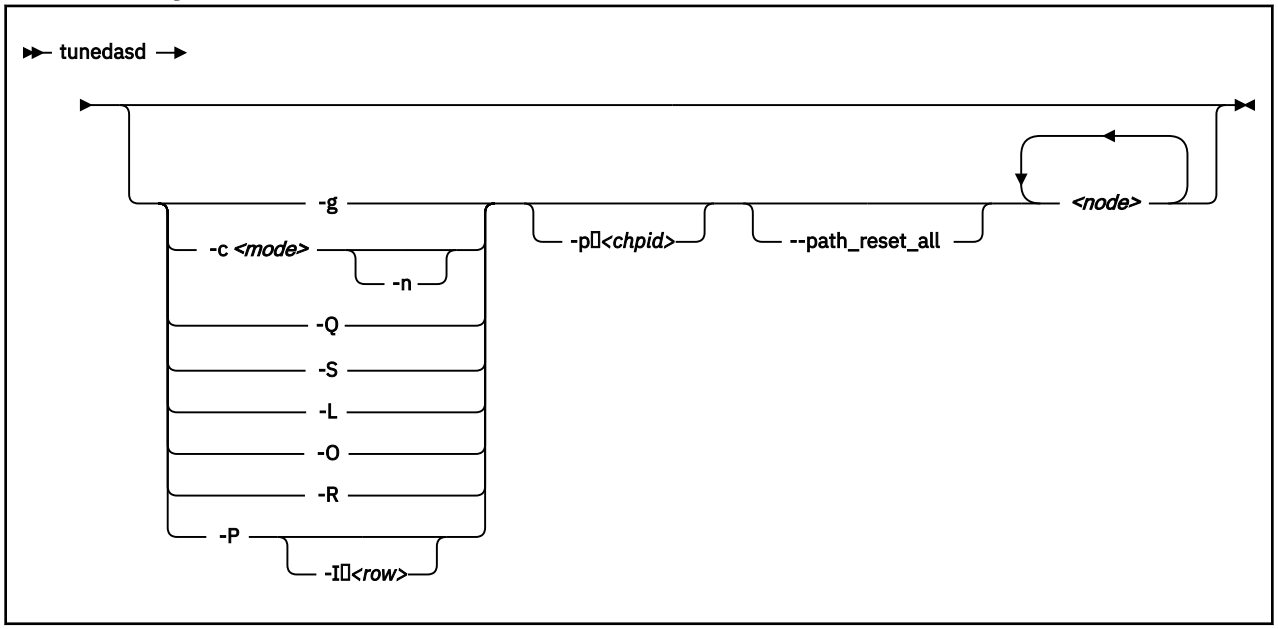
Use the **tunedasd** command to adjust performance relevant settings and other low-level DASD device settings.

In particular, you can perform these tasks:

- Query and set a DASD's cache mode
- Display and reset DASD performance statistics
- Reserve and release DASD
- Break the lock of an online DASD (to learn how to access a boxed DASD that is not yet online, see "Accessing DASD by force" on page 151)

Before you begin: For the performance statistics, data gathering must be turned on by writing "on" to `/proc/dasd/statistics`.

tunedasd syntax



Where:

<node>
specifies a device node for the DASD to which the command is to be applied.

-g or --get_cache
gets the current caching mode of the storage controller. This option applies to ECKD only.

-c *<mode>* or --cache *<mode>*
sets the caching mode on the storage controller to *<mode>*. This option applies to ECKD only.

Today's ECKD devices support the following behaviors:

normal
for normal cache replacement.

bypass
to bypass cache.

inhibit
to inhibit cache.

sequential
for sequential access.

prestage
for sequential prestage.

record
for record access.

For details, see *IBM TotalStorage™ Enterprise Storage Server® System/390® Command Reference 2105 Models E10, E20, F10, and F20*, SC26-7295.

-n *<cylinders>* or --no_cyl *<cylinders>*
specifies the number of cylinders to be cached. This option applies to ECKD only.

-Q or --query_reserve
queries the reserve status of the device. The status can be:

none
the device is not reserved.

implicit
the device is not reserved, but there is a contingent or implicit allegiance to this Linux instance.

other
the device is reserved to another operating system instance.

reserved
the device is reserved to this Linux instance.

For details, see the "Storage Control Reference" of the attached storage server.

This option applies to ECKD only.

-S or --reserve
reserves the device. This option applies to ECKD only.

-L or --release
releases the device. This option applies to ECKD only.

-O or --slock
reserves the device unconditionally. This option applies to ECKD only.

Note: This option is to be used with care as it breaks any existing reserve by another operating system.

-R or --reset_prof
resets the profile information of the device.

-P or --profile
displays a usage profile of the device.

-I *<row>* or --prof_item *<row>*
prints the usage profile item that is specified by *<row>*. *<row>* can be one of:

reqs
number of DASD I/O requests.

sects
number of 512-byte sectors.

sizes
histogram of sizes.

total
histogram of I/O times.

totsect
histogram of I/O times per sector.

start
histogram of I/O time until ssch.

irq
histogram of I/O time between ssch and irq.

irqsect
histogram of I/O time between ssch and irq per sector.

end
histogram of I/O time between irq and end.

queue
number of requests in the DASD internal request queue at enqueueing.

-p *<chpid>* or --path_reset *<chpid>*
resets a channel path *<chpid>* of a selected device. A channel path might be suspended due to high IFCC error rates or a High Performance FICON failure. Use this option to resume considering the channel path for I/O.

--path_reset_all
resets all channel paths of a selected device. The channel paths might be suspended due to high IFCC error rates or a High Performance FICON failure. Use this option to resume considering all defined channel paths for I/O.

-v or --version
displays version information.

-h or --help
displays help information.

Examples

- The following sequence of commands first checks the reservation status of a DASD and then reserves it:

```
# tunedasd -Q /dev/dasdzzz
none
# tunedasd -S /dev/dasdzzz
Reserving device </dev/dasdzzz>...
Done.
# tunedasd -Q /dev/dasdzzz
reserved
```

- This example first queries the current setting for the cache mode of a DASD with device node `/dev/dasdzzz` and then sets it to one cylinder "prestage".

```
# tunedasd -g /dev/dasdzzz
normal (0 cyl)
# tunedasd -c prestage -n 2 /dev/dasdzzz
Setting cache mode for device </devdasdzzz>...
Done.
# tunedasd -g /dev/dasdzzz
prestage (2 cyl)
```

- In this example two device nodes are specified. The output is printed for each node in the order in which the nodes where specified.

```
# tunedasd -g /dev/dasdzzz /dev/dasdzzy
prestage (2 cyl)
normal (0 cyl)
```

- The following command prints the usage profile of a DASD.

```
# tunedasd -P /dev/dasdzzz

19617 dasd I/O requests
with 4841336 sectors(512B each)

   __<4    ___8    __16    __32    __64    _128    _256    _512    __1k    __2k    __4k    __8k    _16k    _32k    _64k    128k
   _256    _512    __1M    __2M    __4M    __8M    _16M    _32M    _64M    128M    256M    512M    __1G    __2G    __4G    _>4G
Histogram of sizes (512B secs)
      0       0     441      77      78      87     188   18746       0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O times (microseconds)
      0       0       0       0       0       0       0     235     150     297   18683     241       3       4       4
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O times per sector
      0       0       0   18736     333     278      94      78      97       1       0       0       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O time till ssch
  19234      40      32       0       2       0       0       3      40      53     128      85       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O time between ssch and irq
      0       0       0       0       0       0       0       0     387     208     250   18538     223       3       4       4
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O time between ssch and irq per sector
      0       0       0   18803     326     398      70      19       1       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O time between irq and end
  18520     735     246      68      43       4       1       0       0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
# of req in chanq at enqueuing (1..32)
      0   19308     123      30      25     130       0       0       0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
```

- The following command prints a row of the usage profile of a DASD. The output is on a single line as indicated by the (cont...) (... cont) in the illustration:

```
# tunedasd -P -I irq /dev/dasdzzz
         0|         0|         0|         0|         0|         0|         0|         0|       503|       271|(cont...)
 (... cont)       267|     18544|       224|         3|         4|         4|         0|         0|         0|(cont...)
 (... cont)         0|         0|         0|         0|         0|         0|         0|         0|         0|(cont...)
 (... cont)         0|         0|         0|         0|
```

- The following command resets a failed channel path with CHPID 45:

```
# tunedasd -p 45 /dev/dasdc
```

vmcp - Send CP commands to the z/VM hypervisor

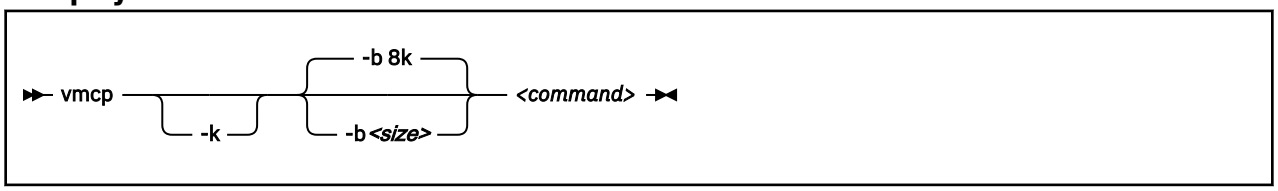
Use the **vmcp** command to send control program (CP) commands to the z/VM hypervisor and display the response from z/VM.

The **vmcp** command expects the command line as a parameter and returns the response to stdout. Error messages are written to stderr.

You can issue **vmcp** commands using the /dev/vmcp device node (see Chapter 41, "z/VM CP interface device driver," on page 447) or from a command prompt in a terminal session. In both cases, you must load the vmcp module.

Before you begin: Ensure that vmcp is loaded by issuing: **modprobe vmcp**.

vmcp syntax



Where:

-k or --keepcase
preserves the case of the characters in the specified command string. By default, the command string is converted to uppercase characters.

-b *<size>* or --buffer=*<size>*
specifies the buffer size in bytes for the response from z/VM CP. Valid values are from 4096 (or 4k) up to 1048756 (or 1M). By default, **vmcp** allocates an 8192 byte (8k) buffer. You can use k and M to specify kilo- and megabytes. The suffixes are not case sensitive, so k is equivalent to K and m is equivalent to M.

<command>
specifies the command that you want to send to CP.

-h or --help
displays help information.

-v or --version
displays version information.

If the command completes successfully, **vmcp** returns 0. Otherwise, **vmcp** returns one of the following values:

1. CP returned a non-zero response code.
2. The specified buffer was not large enough to hold CP's response. The command was run, but the response was truncated. You can use the **--buffer** option to increase the response buffer.
3. Linux reported an error to **vmcp**. See the error message for details.
4. The options that are passed to **vmcp** were erroneous. See the error messages for details.

Examples

- To get your user ID issue:

```
# vmcp query userid
```

- To attach the device 1234 to your guest, issue:

```
# vmcp attach 1234 \*
```

- If you add the following line to `/etc/sudoers`:

```
ALL ALL=NOPASSWD:/sbin/vmcp indicate
```

  every user on the system can run the **indicate** command by using:

```
# sudo vmcp indicate
```

- If you need a larger response buffer, use the `--buffer` option:

```
# vmcp --buffer=128k q 1-ffff
```

vmur - Work with z/VM spool file queues

Use the **vmur** command to work with z/VM spool file queues.

The **vmur** command provides these main functions:

Receive
Read data from the z/VM reader file queue. The command performs the following steps:

- Places the reader queue file to be received at the top of the queue.
- Changes the reader queue file attribute to NOHOLD.
- Closes the z/VM reader after the file is received.

The **vmur** command detects z/VM reader queue files in:

- VMDUMP format as created by CP VMDUMP.
- NETDATA format as created by CMS SENDFILE or TSO XMIT.

Punch or print
Write data to the z/VM punch or printer file queue and transfer it to another user's virtual reader, optionally on a remote z/VM node. The data is sliced up into 80-byte or 132-byte chunks (called *records*) and written to the punch or printer device. If the data length is not an integer multiple of 80 or 132, the last record is padded.

List
Display detailed information about one or all files on the specified spool file queue.

Purge
Remove one or all files on a spool file queue.

Order
Position a file at the top of a spool file queue.

Before you begin: To use the receive, punch, and print functions, the vmur device driver must be loaded and the corresponding unit record devices must be set online.

Serialization

The **vmur** command provides strict serialization of all its functions other than list, which does not affect a file queue's contents or sequence. Thus concurrent access to spool file queues is blocked to prevent unpredictable results or destructive conflicts.

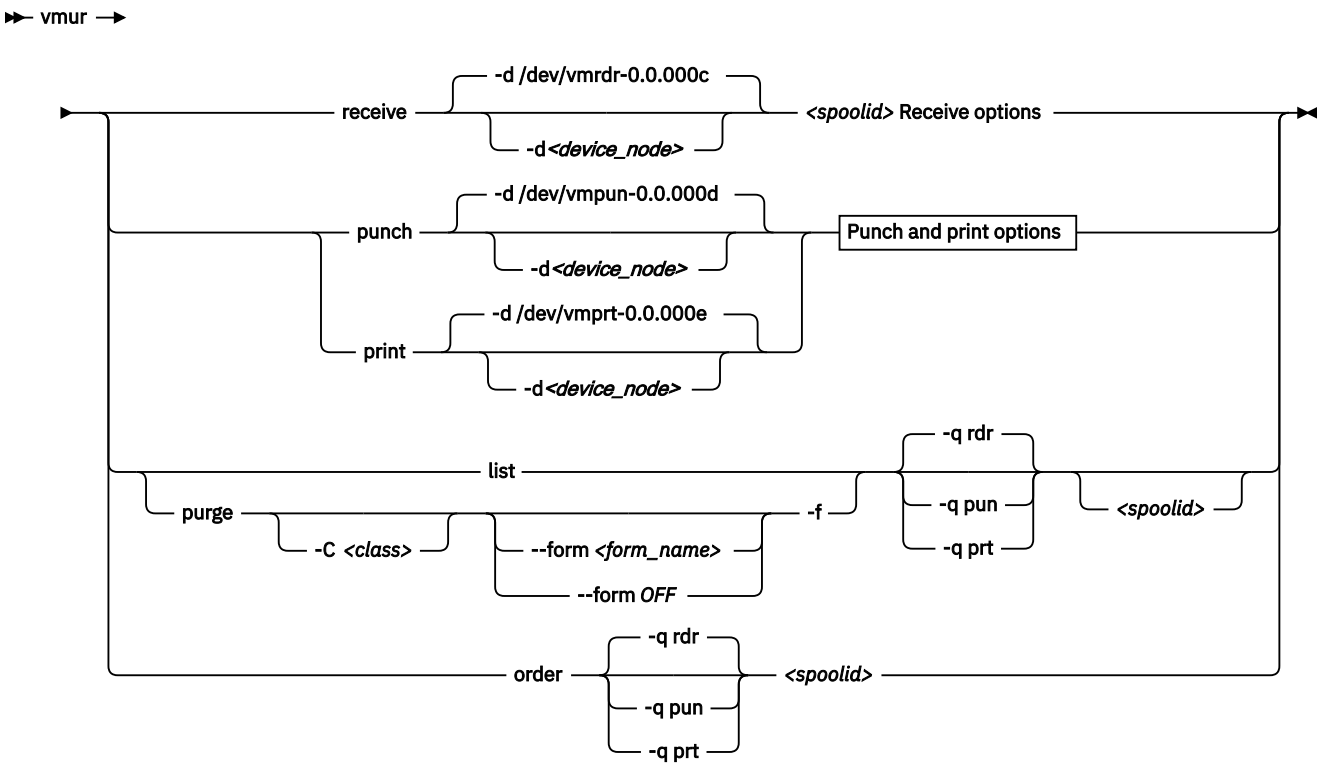
For example, this serialization prevents a process from issuing **vmur purge -f** while another process is running **vmur receive 1234**. However, **vmur** is not serialized against concurrent CP commands that are issued through **vmcp**: if one process is running **vmur receive 1234** and another process issues **vmcp purge rdr 1234**, then the received file might be incomplete. To avoid such unwanted effects, always use **vmur** to work with z/VM spool file queues.

Spooling options

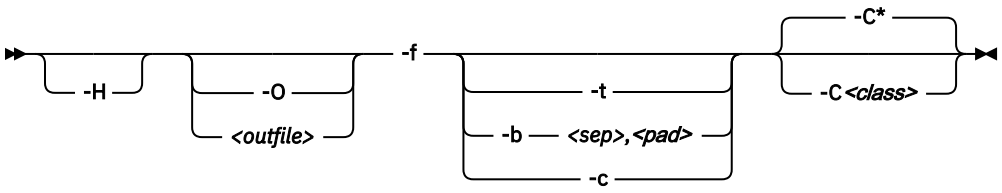
With the **vmur** command, you can temporarily override the z/VM settings for the CLASS, DEST, FORM, and DIST spooling options for virtual unit record devices. The **vmur** command restores the original settings before it returns control.

For details about the spooling options, see the z/VM product information. In particular, see the sections about the z/VM CP SPOOL, QUERY VIRTUAL RDR, QUERY VIRTUAL PUN, and QUERY VIRTUAL PRT commands in *z/VM: CP Commands and Utilities Reference*, SC24-6268.

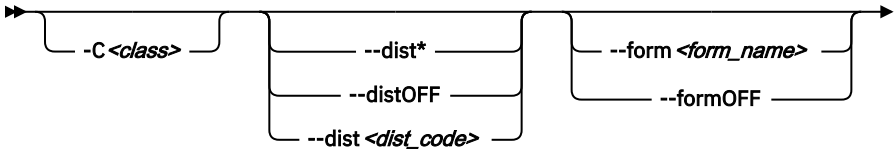
vmur syntax

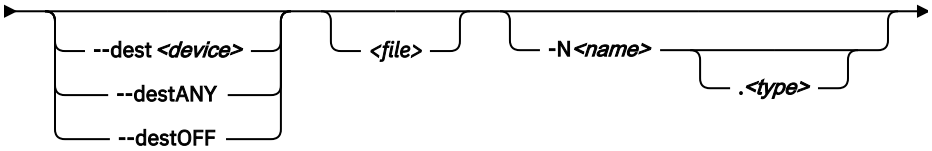


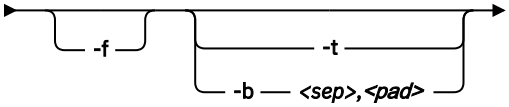
Receive options

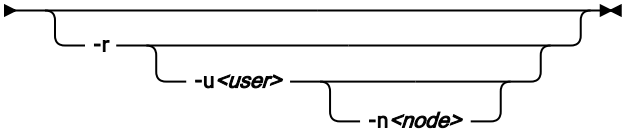


Punch and print options









Where these are the main command options:

re or receive
receives a file from the z/VM reader queue.

pun or punch
writes to the z/VM punch queue.

pr or print
writes to the z/VM printer queue.

li or list
lists information about one or all files on a z/VM spool file queue.

pur or purge
purges one or all files from a z/VM spool file queue.

or or order
places a file on a z/VM spool file queue at the top of the queue.

Note: The short forms that are given for receive, punch, print, list, purge, and order are the shortest possible abbreviations. In keeping with z/VM style, you can abbreviate commands by dropping any number of letters from the end of the full keywords until you reach the short form. For example, **vmur re**, **vmur rec**, or **vmur rece** are all equivalent.

The remaining specifications are listed alphabetically by switch. Variable specifications that do not require a switch are listed first.

<file>
specifies a file, in the Linux file system, with data to be punched or printed. If this specification is omitted, the data is read from standard input.

<outfile>
specifies a file, in the Linux file system, to receive data from the reader spool file. If neither a file name nor **--stdout** are specified, the name and type of the spool file to be received (see the NAME and TYPE columns in **vmur list** output) are used to build an output file name of the form *<name>.<type>*. If the spool file to be received is an unnamed file, an error message is issued.

Use the `--force` option to overwrite existing files without a confirmation prompt.

<spoolid>
specifies the spool ID of a file on the z/VM reader, punch, or printer queue. Spool IDs are decimal numbers in the range 0-9999.

For the list or purge function: omitting the spool ID lists or purges all files in the queue.

-b *<sep>,<pad>* or --blocked *<sep>,<pad>*
receives or writes a file in blocked mode, where *<sep>* specifies the separator and *<pad>* specifies the padding character in hexadecimal notation. Example: *<sep>*

```
--blocked 0xSS,0xPP
```

Use this option to use character sets other than IBM037 and ISO-8859-1 for conversion.

- For the receive function: All trailing padding characters are removed from the end of each record that is read from the virtual reader and the separator character is inserted afterward. The receive function's output can be piped to **iconv** by using the appropriate character sets. Example:

```
# vmur rec 7 -b 0x25,0x40 -O | iconv -f EBCDIC-US -t ISO-8859-1 > myfile
```

- For the punch or print function: The separator is used to identify the line end character of the file to punch or print. If a line has fewer characters than the record length of the used unit record device, the residual of the record is filled up with the specified padding byte. If a line exceeds the record size, an error is printed. Example:

```
# iconv test.txt -f ISO-8859-1 -t EBCDIC-US | vmur pun -b 0x25,0x40 -N test
```

-c or --convert
converts a VMDUMP spool file into a format appropriate for further analysis with crash.

-C *<class>* or --class *<class>*
specifies a spool class.

- For the receive function: The file is received only if it matches the specified class.
- For the purge function: Only files with the specified class are purged.
- For the punch or printer function: Sets the spool class for the virtual reader or virtual punch device. Output files inherit the spool class of the device.

The class is designated by a single alphanumeric character. For receive, it can also be an asterisk (*) to match all classes. Lowercase alphabetic characters are converted to uppercase.

See also "Spooling options" on page 747.

--dest *<device>*
sets the destination device for spool files that are created on the virtual punch or printer device. The value can be ANY, OFF, or it must be a valid device as defined on z/VM.

See also "Spooling options" on page 747.

-d or --device
specifies the device node of the virtual unit record device.

- If omitted in the receive function, `/dev/vmrdr-0.0.000c` is assumed.
- If omitted in the punch function, `/dev/vmpun-0.0.000d` is assumed.
- If omitted in the print function, `/dev/vmprt-0.0.000e` is assumed.

--dist *<distcode>*
sets the distribution code for spool files that are created on the virtual punch or printer device. The value can be an asterisk (*), OFF, or it must be a valid distribution code as defined on z/VM.

OFF and * are equivalent. Both specifications reset the distribution code to the value that is set in the user directory.

See also "Spooling options" on page 747.

-f or --force
suppresses confirmation messages.

- For the receive function: overwrites an existing output file without prompting for a confirmation.
- For the punch or print option: automatically converts the Linux input file name to a valid spool file name without any error message.
- For the purge function: purges the specified spool files without prompting for a confirmation.

--form *<form_name>*
sets the form name for spool files that are created on the virtual punch or printer device. The value can be OFF, to use the system default, or it must be a valid z/VM form name.

See also "Spooling options" on page 747.

-h or --help
displays help information for the command. To view the man page, enter **`man vmur`**.

-H or --hold
keeps the spool file to be received in the reader queue. If omitted, the spool file is purged after it is received.

-n *<node>* or --node *<node>*
specifies the node name of the z/VM system to which the data is to be transferred. Remote Spooling Communications Subsystem (RSCS) must be installed on the z/VM systems and the specified node must be defined in the RSCS machine's configuration file.

The default node is the local z/VM system. The node option is valid only with the `-u` option.

-N *<name>.<type>* or --name *<name>.<type>*
specifies a name and, optionally, a type for the z/VM spool file to be created by the punch or print option. To specify a type after the file name, enter a period followed by the type. For example:

```
# vmur pun -r /boot/parmfile -N myname.mytype
```

Both the name and the type must comply with z/VM file name rules, for example, they must be 1 - 8 characters long.

If omitted, a spool file name is generated from the Linux input file name, if applicable.

Use the **--force** option to suppress warning messages about automatically generated file names or about specified file names that do not adhere to the z/VM file naming rules.

-O or --stdout
writes the reader file content to standard output.

-q or --queue
specifies the z/VM spool file queue to be listed, purged, or ordered. If omitted, the reader file queue is assumed.

-r or --rdr
transferres a punch or print file to a reader.

-t or --text
converts the encoding between EBCDIC and ASCII according to character sets IBM037 and ISO-8859-1.

- For the receive function: receives the reader file as text file. That is, it converts EBCDIC to ASCII and inserts an ASCII line feed character (0x0a) for each input record that is read from the z/VM reader. Trailing EBCDIC blanks (0x40) in the input records are stripped.
- For the punch or print function: punches or prints the input file as text file. That is, converts ASCII to EBCDIC and pads each input line with trailing blanks to fill up the record. The record length is 80 for a punch and 132 for a printer. If an input line length exceeds 80 for punch or 132 for print, an error message is issued.

The **--text** and the **--blocked** attributes are mutually exclusive.

-u *<user>* or --user *<user>*
specifies the z/VM user ID to whose reader the data is to be transferred. If omitted, the data is transferred to your own machine's reader. The user option is valid only with the **-r** option.

-v or --version
displays version information.

Examples

These examples illustrate common scenarios for unit record devices.

In all examples the following device nodes are used:

- `/dev/vmrdr-0.0.000c` as virtual reader.
- `/dev/vmpun-0.0.000d` as virtual punch.

The vmur commands access the reader device, which has to be online. To set it online, it needs to be freed from cio_ignore. Example:

```
# cio_ignore -r c
# chccwdev -e c
Setting device 0.0.000c online
Done
```

Besides the vmur device driver and the **vmur** command, these scenarios require that the **vmcp** and **vmconvert** commands from the s390utils package are available.

Creating and reading a guest memory dump

You can use the **vmur** command to read a guest memory dump that was created, for example, with the **vmcp** command.

Procedure

1. Produce a memory dump of the z/VM guest virtual machine memory:

```
# vmcp vmdump
```

   Depending on the memory size this command might take some time to complete.
2. List the spool files for the reader to find the spool ID of the dump file, VMDUMP.

   In the example, the spool ID of VMDUMP is 463.

```
# vmur li

ORIGINID FILE CLASS RECORDS  CPY HOLD DATE  TIME     NAME   TYPE DIST
T6360025 0463 V DMP 00020222 001 NONE 06/11 15:07:42 VMDUMP FILE T6360025
```

3. Read and convert the VMDUMP spool file to a file in the current working directory of the Linux file system:

```
# vmur rec 463 -c linux_dump
```

Using FTP to receive and convert a dump file

Use the **--convert** option together with the **--stdout** option to receive a VMDUMP spool file straight from the z/VM reader queue, convert it, and send it to another host with FTP.

Procedure

1. Establish an FTP session with the target host and log in.
2. Enter the FTP command `binary`.
3. Enter the FTP command:

```
put |"vmur re <spoolid> -c -O" <filename_on_target_host>
```

Log and read the z/VM guest virtual machine console

You can use the **vmur** command to read a console transcript that has been spooled; for example, with the **vmcp** command.

Procedure

1. Begin console spooling:

```
# vmcp sp cons start
```

2. Produce output to the z/VM console (for example, with CP TRACE).
3. Stop console spooling, close the file with the console output, and transfer the file to the reader queue. In the resulting CP message, the spool ID follows the FILE keyword. In the example, the spool ID is 398:

```
# vmcp sp cons stop close \* rdr

RDR FILE 0398 SENT FROM T6360025 CON WAS 0398 RECS 1872 CPY 001 T NOHOLD NOKEEP
```

4. Read the file with the console output into a file in the current working directory on the Linux file system:

```
# vmur re -t 398 linux_cons
```

Preparing the z/VM reader as an IPL device for Linux

You can use the **vmur** command to transfer all files for booting Linux to the z/VM reader. You can also arrange the files such that the reader can be used as an IPL device.

Procedure

1. Send the kernel parameter file, `parmfile`, to the z/VM punch device and transfer the file to the reader queue.

   The resulting message shows the spool ID of the parameter file.

```
# vmur pun -r /boot/parmfile

Reader file with spoolid 0465 created.
```

2. Send the kernel image file to the z/VM punch device and transfer the file to the reader queue.

   The resulting message shows the spool ID of the kernel image file.

```
# vmur pun -r /boot/vmlinuz -N image

Reader file with spoolid 0466 created.
```

3. Optional: Check the spool IDs of `image` and `parmfile` in the reader queue. In this example, the spool ID of `parmfile` is 465 and the spool ID of `image` is 466.

```
# vmur li

ORIGINID FILE CLASS RECORDS  CPY HOLD DATE  TIME     NAME      TYPE     DIST
T6360025 0463 V DMP 00020222 001 NONE 06/11 15:07:42 VMDUMP    FILE     T6360025
T6360025 0465 A PUN 00000002 001 NONE 06/11 15:30:31 parmfile           T6360025
T6360025 0466 A PUN 00065200 001 NONE 06/11 15:30:52 image              T6360025
```

4. Move `image` to the first and `parmfile` to the second position in the reader queue:

```
# vmur or 465
# vmur or 466
```

5. Configure the z/VM reader as the re-IPL device:

```
# echo 0.0.000c > /sys/firmware/reipl/ccw/device
```

6. Boot Linux from the z/VM reader:

```
# reboot
```

Sending a file to different z/VM guest virtual machines

You can use the **vmur** command to send files to other z/VM guest virtual machines.

About this task

This scenario describes how to send a file called `lnxprofile.exec` from the file system of an instance of Linux on z/VM to other z/VM guest virtual machines. For example, `lnxprofile.exec` could contain the content of a PROFILE EXEC file with CP and CMS commands to customize z/VM guest virtual machines for running Linux.

Procedure

1. Send `lnxprofile.exec` to two z/VM guest virtual machines:
   z/VM user ID t2930020 at node boet2930 and z/VM user ID t6360025 at node boet6360.

```
vmur pun lnxprofile.exec -t -r -u t2930020 -n boet2930 -N PROFILE
vmur pun lnxprofile.exec -t -r -u t6360025 -n boet6360 -N PROFILE
```

2. Log on to t2930020 at boet2930, IPL CMS, and issue the CP command:

```
QUERY RDR ALL
```

   The command output shows the spool ID of PROFILE in the FILE column.
3. Issue the CMS command:

```
RECEIVE <spoolid> PROFILE EXEC A (REPL
```

   In the command, *<spoolid>* is the spool ID of PROFILE found in step “2” on page 754.
4. Repeat steps “2” on page 754 and “3” on page 754 for t6360025 at boet6360.

Sending a file to a z/VSE instance

You can use the **vmur** command to send files to a z/VSE instance.

Procedure

To send `lserv.job` to user ID vseuser at node vse01sys, issue:

```
vmur pun lserv.job -t -r -u vseuser -n vse01sys -N LSERV
```

zdsfs - Mount a z/OS DASD

Use the **zdsfs** command to mount z/OS DASDs as a Linux file system.

With the **zdsfs** command, you can configure DASD volumes for data set conversion. The default conversion for configured DASDs uses code pages EBCDIC CP1047 to UTF-8, but you can configure other code pages for specific data sets.

The zdsfs file system converts the z/OS data sets, which are stored on the DASDs in records of arbitrary or even variable size, into Linux semantics.

Through the zdsfs file system, applications on Linux can read z/OS physical sequential data sets (PS) and partitioned data sets (PDS) on the DASD. In the Linux file system, physical sequential data sets are represented as files. Partitioned data sets are represented as directories that contain the PDS members as files. Other z/OS data set formats, such as extended format data sets or VSAM data sets, are not supported. zdsfs is optimized for sequential read access.

The zdsfs file system requires the FUSE library. Red Hat Enterprise Linux automatically installs this library. The zdsfs command is available from the s390utils-zdsfs RPM.

Attention:

For Red Hat Enterprise Linux earlier than 8.4:

- To avoid data inconsistencies, set the DASDs offline in z/OS before you mount them in Linux.
- Through the zdsfs file system, the whole DASDs are accessible to Linux, but the access is not controlled by z/OS auditing mechanisms.

  To avoid security problems, you might want to dedicate the z/OS DASDs only for providing data for Linux.

Per default, only the Linux user who mounts the zdsfs file system has access to it. Configure file access behavior by using the `allow_other`, `default_permissions`, `umask`, `uid`, and `gid` options.

Tip: If you want to grant a user group access to the zdsfs file system, mount it with the fuse options `default_permissions, allow_other,` and `gid`.

To unmount file systems that you mounted with **zdsfs**, you can use **fusermount**, whether root or non-root user. See the **fusermount** man page for details.

For more information about z/OS data sets, see *z/OS DFSMS Using Data Sets*, SC26-7410.

Controlling read access using a REST server

As of Red Hat Enterprise Linux 8.4, you can use a z/OSMF REST server to control access. A configuration file configures access to the REST server, see “zdsfs configuration file” on page 760.

Using a REST server, the DASD does not have to be offline for Linux to access it. The REST server also enables authorization checking and audit capabilities.

To authenticate with the REST server from Linux, use a `.netrc` file. The `.netrc` file must be in your home directory and contain an entry for the REST server. Example `.netrc` file:

```
machine example.com
login user
password secret
```

Before you begin

- The raw-track access mode of the DASD must be enabled.

  Make sure that the DASD is set offline when you enable the raw-track access mode.

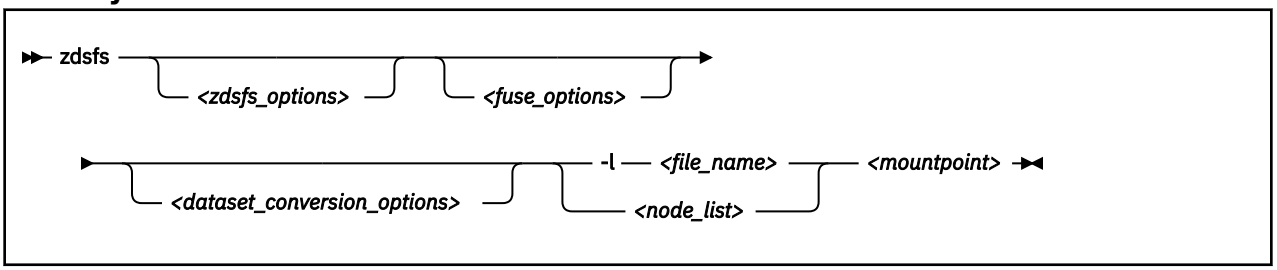
  See “Accessing full ECKD tracks” on page 162 for details.

- The DASD must be online.

  Tip: You can use the **chccwdev** command to enable the raw-track access mode and set the device online afterward in one step.

  Unless you use a REST server, set the DASD offline in z/OS before you set it online in Linux.
- You must have the appropriate read permissions for the device node.
- To use a REST server for read access:
  - A z/OSMF REST server must be up and running in z/OS.
  - A user ID must exist in z/OS that is allowed to access the required data sets. This user's login credentials must be used for **zdsfs**.

zdsfs syntax



where:

<zdsfs_options>
zdsfs-specific options.

-c <config_file>
provides a configuration file for zdsfs. The default is `/etc/zdsfs.conf`.

-o ignore_incomplete
represents all complete data sets in the file system, even if there are incomplete data sets. Incomplete data sets are not represented.

In z/OS, data sets might be distributed over different DASDs. For each incomplete data set, a warning message is issued to the standard error stream. If there are incomplete data sets and this option is not specified, the **zdsfs** command returns with an error.

-o rdw
keeps record descriptor words (RDWs) of data sets that are stored by using the z/OS concept of variable record lengths.

-o restapi
uses z/OSMF REST services for coordinated read-access to data sets. The user credentials are read from the `.netrc` file in the user's home directory, or from the location the NETRC environment variable points to.

-o restserver=*<server_URL>*
specifies up to three server URLs to z/OSMF REST services. If more than one server is specified, the first that responds is used.

-o seekbuffer=*<s>*
sets the maximum seek history buffer size in bytes. The default is 1,048,576 B.

zdsfs saves offset information about a data set in the seek history buffer to speed up the performance of a seek operation.

-o tracks=*<n>*
specifies the track buffer size in tracks. The default is 128 tracks.

zdsfs allocates a track buffer of *<n>*120 KB for each open file to store and extract the user data. Increasing the track buffer size might improve your system performance.

-o check_host_count
checks the host-access open count to ensure that the device is not online to another operating system instance. The operation is canceled if another operating system instance is accessing the volume.

<fuse_options>
Options for FUSE. The following options are supported by the **zdsfs** command. To use an option, it must also be supported by the version of FUSE that is installed.

-d or -o debug
enables debug output (implies **-f**).

-f
runs the command as a foreground operation.

-o allow_other
allows access to other users.

-o allow_root
allows access to root.

-o default_permissions
enables permission checking by the kernel.

-o max_read=*<n>*
sets maximum size of read requests.

-o kernel_cache
caches files in the kernel.

-o [no]auto_cache
enables or disables caching based on modification times.

-o umask=*<mask>*
sets file permissions (octal).

-o uid=*<n>*
sets the file owner.

-o gid=*<n>*
sets the file group.

-o max_write=*<n>*
sets the maximum size of write requests.

-o max_readahead=*<n>*
sets the maximum readahead value.

-o async_read
performs reads asynchronously (default).

-o sync_read
performs reads synchronously.

<dataset_conversion_options>

-x *<dataset_config_file>*
Specifies a configuration file with code-page conversion options for specific data sets. Command-line options override configuration file specifications. For details about the configuration file, see “Dataset conversion configuration file” on page 761.

-o codepage_convert
Enable code-page conversion for all data sets. The default source code-page is EBCDIC CP1047 and the target code-page is UTF-8. Change the default for the source code-page with the `-o codepage_from` option and for the target code-page with the `-o codepage_to` option. To specify source and target code-pages for individual data sets, use a data-set configuration file.

-o codepage_from=*<n>*
Overrides the default code-page for the source data set. This option requires the `-o codepage_to=<n>` option. For a list of valid code pages, issue `iconv -l`.

-o codepage_to=*<n>*
Overrides the default code-page for the target data set. This option requires the `-o codepage_from=<n>` option. For a list of valid code pages, issue `iconv -l`.

<node_list>
one or more device nodes for the DASDs, separated by blanks.

<file_name>
a file that contains a node list.

<mountpoint>
the mount point in the Linux file system where you want to mount the z/OS data sets.

-h or --help
displays help information for the command. To view the man page, enter **`man zdsfs`**.

-v or --version
displays version information for the command.

File characteristics

There are two ways to handle the z/OS characteristics of a file:

- The file `metadata.txt`:

  The `metadata.txt` file is in the root directory of the mount point. It contains one row for each file or directory, where:

  dsn
  specifies

  – the name of the file in the form *<file_name>* for z/OS physical sequential data sets.

  – the name of the directory in the form *<directory_name>*, and the name of a file in that directory in the form *<directory_name>(<file_name>)* for z/OS partitioned data sets.

  dsorg
  specifies the organization of the file. The organization is PO for a directory, and PS for a file.

  lrecl
  specifies the record length of the file.

  recfm
  specifies the z/OS record format of the file. Supported record formats are: V, F, U, B, S, A, and M.

  Example:

```
dsn=FOOBAR.TESTF.TXT,recfm=FB,lrecl=80,dsorg=PS
dsn=FOOBAR.TESTVB.TXT,recfm=VB,lrecl=100,dsorg=PS
dsn=FOOBAR.PDSF.DAT,recfm=F,lrecl=80,dsorg=PO
dsn=FOOBAR.PDSF.DAT(TEST1),recfm=F,lrecl=80,dsorg=PS
dsn=FOOBAR.PDSF.DAT(TEST2),recfm=F,lrecl=80,dsorg=PS
dsn=FOOBAR.PDSF.DAT(TEXT3),recfm=F,lrecl=80,dsorg=PS
```

- Extended attributes:

  user.dsorg
  specifies the organization of the file.

  user.lrecl
  specifies the record length of the file.

  user.recfm
  specifies the z/OS record format of the file.

  You can use the following system calls to work with extended attributes:

  listxattr
  to list the current values of all extended attributes.

  getxattr
  to read the current value of a particular extended attribute.

You can use these system calls through the **getfattr** command. For more information, see the man pages of these commands and of the listxattr and getxattr system calls.

Examples

- Enable the raw-track access mode of DASD device 0.0.7000 and set the device online afterward:

```
# chccwdev -a raw_track_access=1 -e 0.0.7000
```

- To mount the z/OS DASD represented by the file node /dev/dasde and specifying a z/OSMF REST server for coordinated read-access:

```
# zdsfs -o restapi -o restserver=zos1.server.tld/zosmf /dev/dasde /mnt
```

- To mount disks with a REST server in place:

```
# zdsfs /dev/disk/by-path/ccw-0.0.edc0 /dev/disk/by-path/ccw-0.0.edc7 /mnt/
Using z/OSMF REST services on https://example.com/zosmf/
```

  The mount process informs you of which REST server is used.

- Mount the partitioned data set on the DASDs represented by the file nodes /dev/dasde and /dev/dasdf at /mnt:

```
# zdsfs /dev/dasde /dev/dasdf /mnt
```

- As user "myuser", mount the partitioned data set on the DASD represented by the file node /dev/dasde at /home/myuser/mntzos:
  - Access the mounted file system exclusively:

    ```
    # zdsfs /dev/dasde /home/myuser/mntzos
    ```

  - Allow the root user to access the mounted file system:

    ```
    # zdsfs -o allow_root /dev/dasde /home/myuser/mntzos
    ```

  The **ls** command does not reflect these permissions. In both cases, it shows:

```
# ls -al /home/myuser/mntzos
total 121284
dr-xr-x--- 2 root   root          0 Dec  3 15:54 .
drwx------ 3 myuser myuser     4096 Dec  3 15:51 ..
-r--r----- 1 root   root    2833200 Jun 27  2012 EXPORT.BIN1.DAT
-r--r----- 1 root   root    2833200 Jun 27  2012 EXPORT.BIN2.DAT
-r--r----- 1 root   root    2833200 Jun 27  2012 EXPORT.BIN3.DAT
-r--r----- 1 root   root    2833200 Feb 14  2013 EXPORT.BIN4.DAT
dr-xr-x--- 2 root   root   13599360 Aug  9  2012 EXPORT.PDS1.DAT
dr-xr-x--- 2 root   root   13599360 Aug  9  2012 EXPORT.PDS2.DAT
dr-xr-x--- 2 root   root   13599360 Aug  9  2012 EXPORT.PDS3.DAT
dr-xr-x--- 2 root   root   55247400 Aug  9  2012 EXPORT.PDS4.DAT
-r--r----- 1 root   root        981 Dec  3 15:54 metadata.txt

$ ls -al /dev/dasde
brw-rw---- 1 root disk 94, 16 Dec  3 13:58 /dev/dasde
```

- As root user, mount the partitioned data set on the DASD represented by the file node /dev/dasde at /mnt on behalf of the user ID "myuser" (UID=1002), and permit the members of the group ID "zosimport" (GID=1002) file access:

```
# zdsfs /dev/dasde /mnt -o uid=1002,gid=1002,allow_other,default_permissions
```

  The **ls** command indicates the owner "myuser" and the access right for group "zosimport":

```
$ ls -al /mnt
total 121284
dr-xr-x---  2 myuser zosimport       0 Dec  3 14:22 .
drwxr-xr-x 23 root   root         4096 Dec  3 13:59 ..
-r--r-----  1 myuser zosimport      981 Dec  3 14:22 metadata.txt
-r--r-----  1 myuser zosimport  2833200 Jun 27  2012 EXPORT.BIN1.DAT
-r--r-----  1 myuser zosimport  2833200 Jun 27  2012 EXPORT.BIN2.DAT
-r--r-----  1 myuser zosimport  2833200 Feb 14  2013 EXPORT.BIN3.DAT
-r--r-----  1 myuser zosimport  2833200 Jun 27  2012 EXPORT.BIN4.DAT
dr-xr-x---  2 myuser zosimport 13599360 Aug  9  2012 EXPORT.PDS1.DAT
dr-xr-x---  2 myuser zosimport 13599360 Aug  9  2012 EXPORT.PDS2.DAT
dr-xr-x---  2 myuser zosimport 55247400 Aug  9  2012 EXPORT.PDS3.DAT
dr-xr-x---  2 myuser zosimport 13599360 Aug  9  2012 EXPORT.PDS4.DAT
```

- Unmount the partitioned data set that is mounted at `/mnt`:

```
# fusermount -u /mnt
```

- Show the extended attributes of a file, `FB.XMP.TXT`, on a z/OS DASD that is mounted on `/mnt`:

```
# getfattr -d /mnt/FB.XMP.TXT
```

- Show the extended attributes of all files on a z/OS DASD that is mounted on `/mnt`:

```
# cat /mnt/metadata.txt
```

- Access data:

```
# cat /mnt/MYUSER.ZDSFSDV.LARGE.TEST2 > /dev/null
```

- To mount the z/OS disk and enable code-page conversion for all data sets using a custom source and target code page:

```
# zdsfs -o codepage_from=CP037 -o codepage_to=ISO-8859-1 /dev/dasde /mnt/
```

zdsfs configuration file

The default path to the zdsfs configuration file is `/etc/zdsfs.conf`. Specify a different configuration file location with the `-c` *<config_file>* option.

The configuration file can contain the following options:

restapi = 0 | 1
: Enables (1) or disables (0) the use of z/OSMF REST services. If enabled, a valid REST server must be specified with the `restserver=` option, as well as a `.netrc` file with a valid z/OS user ID and password.

restserver = *<URL>*
: Specifies the URL of the z/OSMF REST server that is used for coordinated read-access.

  For failover scenarios: Optionally, provide up to three additional server addresses. These are tried in the specified order if one of the servers cannot be reached during mount. If a server is unresponsive during operation, all specified server are probed again.

keepalive = *<timeout_in_seconds>*
: Changes the keepalive timer for ENQs. By default the keepalive refreshes the access after 540 seconds (9 minutes). The 9 minutes are chosen to prevent a timeout by z/OS after 10 minutes.

An example of a `zdsfs.conf` file could look similar to the following:

```
# rest enabled
restapi = 1

# rest server
restserver = example.com/zosmf/
```

```
# backup rest server
restserver=https://example.com/zosmf/

# keepalive timeout in seconds (default 540)
keepalive = 540
```

Dataset conversion configuration file

You can specify code-page conversion settings for individual datasets. The default configuration file is `/etc/dataset.conf`. Use the `-x` *<dataset_file>* option to specify a different file.

Each config file entry must contain the following options:

<DATASET.TITLE>

Specifies the data-set title or a pattern of titles to which the entry applies. The title can have a trailing asterisk to match all titles that begin with the leading characters.

conv = 0|1|*<codepage_from>,<codepage_to>*

0
disables code-page conversion.

1
performs conversion with the default conversion table.

Specifying source and target code pages explicitly to override the default conversion tables The code-page specifications must be separated by a comma. For a list of valid code pages specifications, issue `iconv -l`.

rdw = 0 | 1

0
omits the record descriptor word from the data stream.

1
keeps the record descriptor word from the data stream.

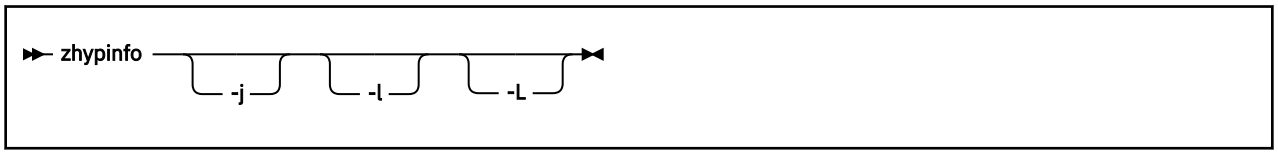
Important: Code-page conversion can render data unreadable if the record-descriptor word is kept.

An example of a data set conversion configuration file could look similar to the following:

```
# default conversion for all files with MYFILE.TXTFILES. prefix
MYFILE.TXTFILES.*
  conv=1
  rdw=0

# no conversion for all files with MYFILE.PDF. prefix
MYFILE.PDF.*
  conv=0
  rdw=0

# no conversion for all files with MYFILE.BINFILE. prefix with RDW kept
MYFILE.BINFILES.*
  conv=0
  rdw=1

# special handling for a specific file
MYFILE.ZDSFSDV.LARGE.FILE4
  conv = EBCDIC-US,ISO-8859-1
  rdw=0
```

zhypinfo - obtain information about virtualization layers on IBM Z

Use the **zhypinfo** command to display information about the virtualization environment of your Linux instance.



Where:

-j or --json
writes all available information including STHYI in JSON format. No character set conversion or escaping is done.

-l or --layers
prints the number of virtualization layers.

Each virtualization layer that can host multiple guests constitutes a virtualization level. Layers include the physical hardware and hypervisors.

-L or --levels
prints the number of virtualization levels.

Output table

The command prints a table with the following columns.

#
Index number of the respective layer.

Layer_Type
Type of virtualization layer.

Lvl
Virtualization level, where each layer of category 'HOST' constitutes a new level.

Categ
Category to which the virtualization layer belongs, virtualization host or guest.

Name
Name of the respective entity.

IFLs
Number of logical IFLs defined for the layer.

CPs
Number of logical CPs defined for the layer.

Total
Sum of logical CPUs in the layer.

Unavailable data is indicated by a dash (-).

Example

- To display information about a Linux instance running as a KVM guest:

```
[kvmguest]# zhypinfo

# Layer_Type       Lvl Categ Name     IFLs   CPs     Total
------------------------------------------------------------
3 KVM-guest          1 GUEST zhyp135    10     0        10
2 KVM-hypervisor     1 HOST     -        9     0         9
1 LPAR               0 GUEST S38LP40     9     0         9
0 CEC                0 HOST  S38       125     4       129
```

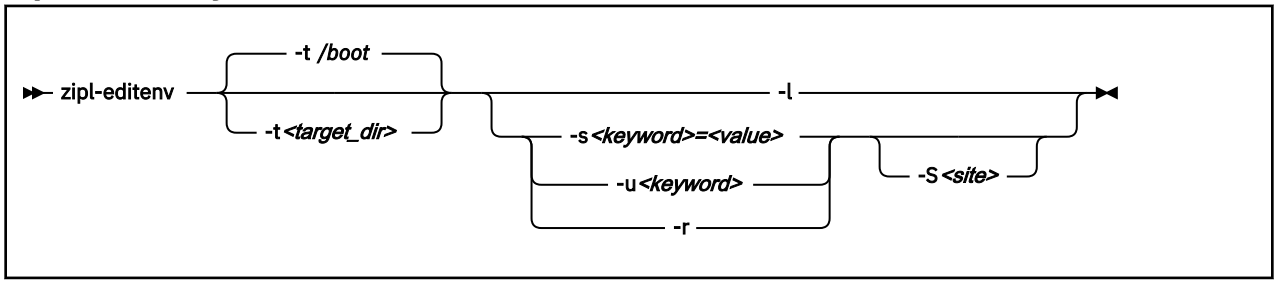
zipl-editenv - Edit the zipl environment block

Use the **zipl-editenv** command to display and change the zipl environment block.

Modifications that are made with **zipl-editenv** are lost after rerunning zipl. Modifications that are made with the **zipl-editenv** command do not affect the zipl environment file. Keep the environment file current to avoid inconsistencies. For more information about the zipl environment file, see "zipl environment - Variables for the kernel command line" on page 80.

CAUTION: Do not modify the zipl environment block by any other means than the **zipl-editenv** command.

zipl-editenv syntax



Where:

-t *<target_dir>* or --target *<target_dir>*
specifies a directory that contains the boot data. The default is `-t /boot`.

-l or --list
prints a list of all keyword-value pairs in the zipl environment block.

-s *<keyword>*=*<value>* or --set *<keyword>*=*<value>*
assigns the specified value to the keyword in the common name space, unless -S is specified. The value can consist of any printable characters, but must not contain functions, such as the new-line symbol. If a keyword does not exist it is added.

The keyword must satisfy the following requirements:

- Consist of uppercase letters A - Z, digits 0 = 9, and the "_" (underscore).
- Must not begin with a digit.

The maximum number of keyword-value pairs per boot partition is 512.

-u *<keyword>* or --unset *<keyword>*
removes the specified keyword from the common name space of the zipl environment block, unless `-S` is specified.

-r or --reset
removes all keywords from a common name space of the zipl environment block, unless `-S` is specified. All variables on the command line then resolve to the empty string.

-S *<site>* or -- site *<site>*
specifies the site as a numeral in the range 0 to 9. Combine the `-S` with the `-s`, `-u`, or the `-r` options to manipulate keywords for the specified site.

-h or --help
displays help text.

-v or --version
displays information about the version.

Examples

- To list the zipl environment block, issue:

```
# zipl-editenv -l
ROOT=/dev/dasda1
PANIC_TIMEOUT=panic=8
```

- To change the value of PANIC_TIMEOUT to `panic=9`, issue:

```
# zipl-editenv -s PANIC_TIMEOUT=panic=9
```

  Use the `--list` option to check that the keywords and their values are now as expected:

```
# zipl-editenv -l
ROOT=/dev/dasda1
PANIC_TIMEOUT=panic=9
```

- Assume that you have created an environment file with two sites, site 1 and site 2. You then run zipl to prepare the boot device for IPL. The installed environment might look for example like this:

```
# zipl-editenv --list
Common variables:
  VAR_A=A
Site 1:
  VAR_A=A1
Site 2:
  VAR_B=B2
```

  In this example, the common name space contains keyword VAR_A with the value A, site 1 contains the same keyword with a different value, A1, and site 2 contains keyword VAR_B with the value B2.

  – To define a value for VAR_A at site 2, issue:

```
# zipl-editenv --set VAR_A=A2 --site 2
```

  – To add a keyword VAR_C in the common name space, issue:

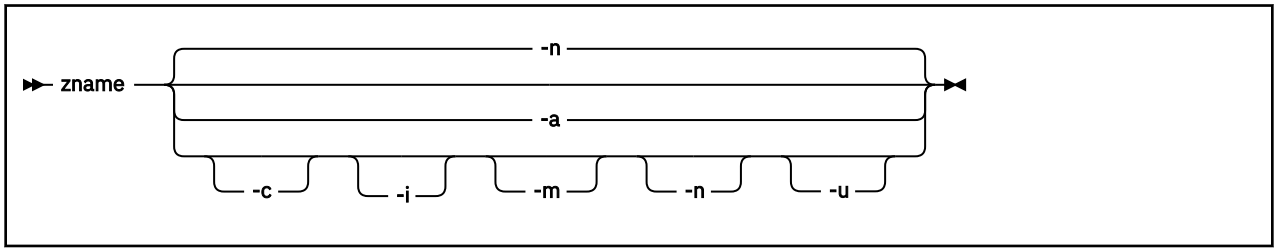
```
# zipl-editenv --set VAR_C=C
```

  List the environment to see the changes:

```
# zipl-editenv --list
Common variables:
  VAR_A=A
  VAR_C=C
Site 1:
  VAR_A=A1
Site 2:
  VAR_A=A2
  VAR_B=B2
```

zname - Obtain information about the IBM Z hardware

Use the **zname** command to display information about the hardware your Linux instance runs on.



Where:

-a or --all
prints all available information in the following order: Name, model, capacity, model, manufacturer.

-c or --capacity
prints capacity information.

-i or --cpuid
prints the CPU identifier.

-m or --model
prints model information.

-n or --name
prints the model name. This is the default.

-u or --manufacturer
prints manufacturer information.

Examples

- To print the default information (the name), issue:

```
# zname
IBM z16
```

- To print all available data, issue:

```
zname -a
IBM z16 A01 701 3931 IBM
```

The sample output is for IBM z16, model A01, with a CPU capacity of 701, machine type 3931, and with IBM as the manufacturer.

znetconf - List and configure network devices

Use the **znetconf** command to list, configure, add, and remove network devices.

The **znetconf** command:

- Lists potential network devices.
- Lists configured network devices.
- Automatically configures and adds network devices.
- Removes network devices.

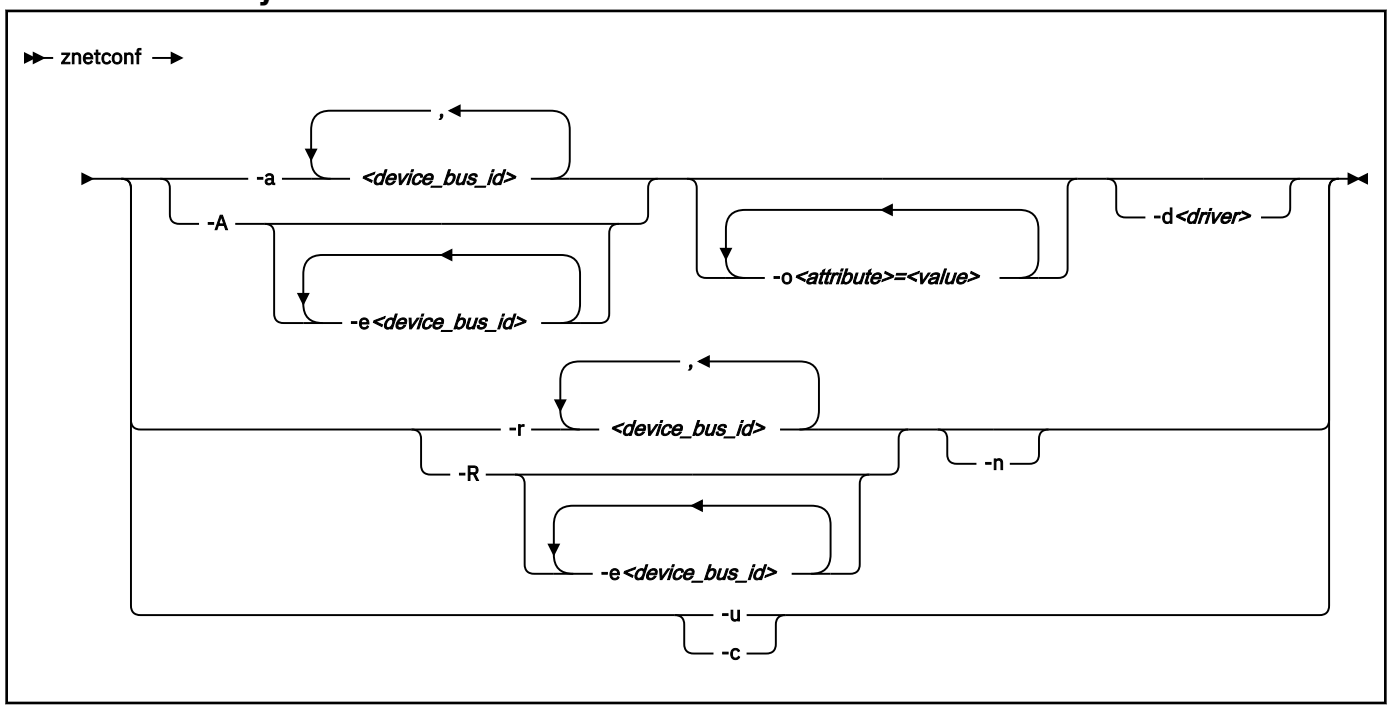
For automatic configuration, **znetconf** first builds a channel command word (CCW) group device from sensed CCW devices. It then configures any specified option through the sensed network device driver and sets the new network device online.

During automatic removal, **znetconf** sets the device offline and removes it.

Attention: Removing all network devices might lead to complete loss of network connectivity. Unless you can access your Linux instance from a terminal server on z/VM (see *How to Set up a Terminal Server Environment on z/VM*, SC34-2596), you might require the HMC or a 3270 terminal session to restore the connectivity.

Before you begin: The qeth, ctcm, or lcs device drivers must be loaded. If needed, the **znetconf** command attempts to load the particular device driver.

znetconf syntax



Where:

-a or --add
configures the network device with the specified device bus-ID. If you specify only one bus ID, the command automatically identifies the remaining bus IDs of the group device. You can enter a list of device bus-IDs that are separated by commas. The **znetconf** command does not check the validity of the combination of device bus-IDs.

<device_bus_id>
specifies the device bus-ID of the CCW devices that constitute the network device. If a device bus-ID begins with "0.0.", you can abbreviate it to the final hexadecimal digits. For example, you can abbreviate 0.0.f503 to f503.

-A or --add-all
configures all potential network devices. After you run **`znetconf -A`**, enter **`znetconf -c`** to see which devices were configured. You can also enter **`znetconf -u`** to display devices that were not configured.

-e or --except
omits the specified devices when configuring all potential network devices or removing all configured network devices.

-o *<attribute>=<value>* or --option *<attribute>=<value>*
configures devices with the specified sysfs option.

-d *<driver name>* or --driver *<driver name>*
configures devices with the specified device driver. Valid values are qeth, lcs, ctc, or ctcm.

-n or --non-interactive
answers all confirmation questions with "Yes".

-r or --remove
removes the network device with the specified device bus-ID. You can enter a list of device bus-IDs that are separated by a comma. You can remove only configured devices as listed by **`znetconf -c`**.

-R or --remove-all
removes all configured network devices. After successfully running this command, all devices that are listed by **`znetconf -c`** become potential devices that are listed by **`znetconf -u`**.

-u or --unconfigured
lists all network devices that are not yet configured.

-c or --configured
lists all configured network devices.

-h or --help
displays short information about command usage. To view the man page, enter **`man znetconf`**.

-v or --version
displays version information.

If the command completes successfully, **`znetconf`** returns 0. Otherwise, 1 is returned.

Examples

- To list all potential network devices:

```
# znetconf -u
Device IDs                 Type    Card Type  CHPID Drv.
------------------------------------------------------------
0.0.f500,0.0.f501,0.0.f502 1731/01 OSA (QDIO) 00    qeth
0.0.f503,0.0.f504,0.0.f505 1731/01 OSA (QDIO) 01    qeth
```

- To configure device 0.0.f503:

```
znetconf -a 0.0.f503
```

or

```
znetconf -a f503
```

- To configure the potential network device 0.0.f500 with the `layer2` option with the value 0:

```
znetconf -a f500 -o layer2=0
```

- To list configured network devices:

```
znetconf -c
Device IDs                 Type    Card Type     CHPID Drv. Name          State
------------------------------------------------------------------------------
0.0.f500,0.0.f501,0.0.f502 1731/01 Virt.NIC QDIO 00    qeth encf500 online
0.0.f503,0.0.f504,0.0.f505 1731/01 Virt.NIC QDIO 01    qeth encf503 online
0.0.f5f0,0.0.f5f1,0.0.f5f2 1731/01 OSD_1000      76    qeth encf5f0 online
```

- To remove network device 0.0.f503:

```
znetconf -r 0.0.f503
```

or

```
znetconf -r f503
```

- To remove all configured network devices except the devices with bus IDs 0.0.f500 and 0.0.f5f0:

```
znetconf -R -e 0.0.f500 -e 0.0.f5f0
```

- To configure all potential network devices except the device with bus ID 0.0.f503:

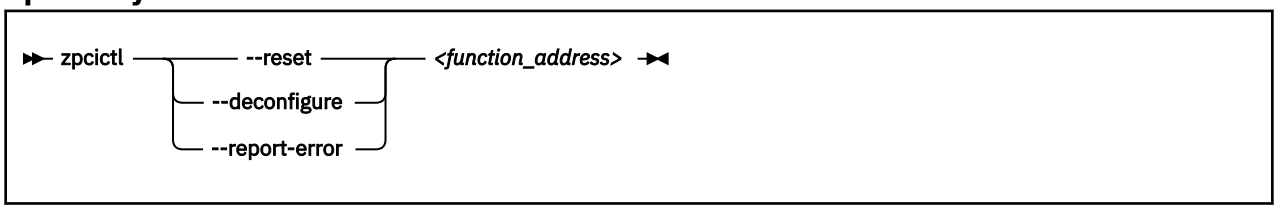
```
znetconf -A -e 0.0.f503
```

zpcictl - Manage defective PCIe devices

Use the **zpcictl** to manage PCI devices on Linux in LPAR mode and Linux on z/VM. In particular, use this command to report defective PCI devices to the Support Element (SE).

Note: With the `smartmontools` package installed, diagnostic data might be collected and sent with the recovery action. Whether diagnostic data is collected and which data is available depends on the PCI device type.

zpcictl syntax



Where:

--reset
resets the device.

--deconfigure
deconfigures the device to prepare for any repair action.

--report-error
reports the device error to the SE.

<function_address>
the PCI function address of the device, for example, `0000:00:00.0`.

-h or --help
displays help information for the command. To view the man page, enter **man zpcictl**.

-v or --version
displays version information.

Example

In this example, an error is reported for a PCI device with slot ID `0000:00:00.0`.

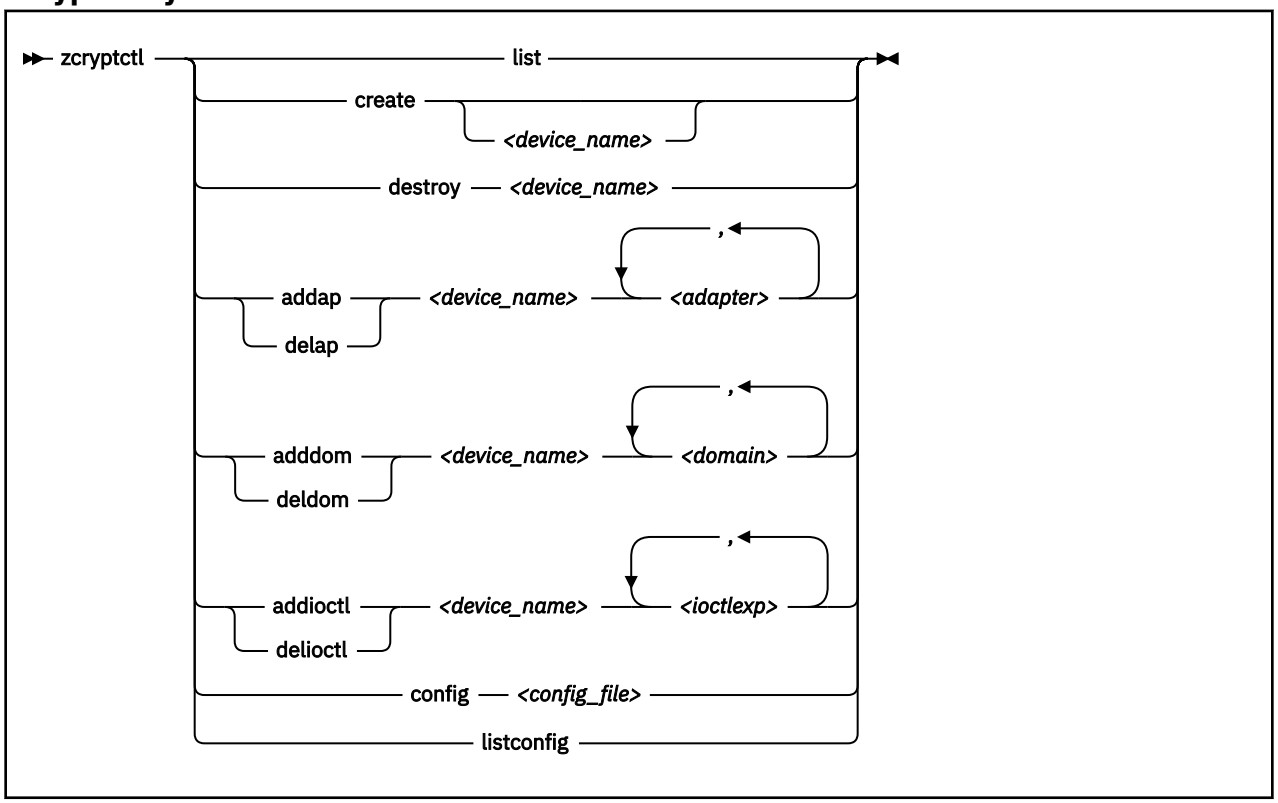
```
# zpcictl --report-error 0000:00:00.0
```

zcryptctl - Control access to AP queues and functions

Use the **zcryptctl** command to control access to AP queues and functions.

For more information about cryptographic device nodes, see “Creating customized device nodes” on page 497.

zcryptctl syntax



Where:

list
lists all zcrypt device nodes.

create *<node_name>*
creates a new zcrypt device node. The *<device_name>* is optional and must be unique. If no node name is provided, the zcrypt device driver creates one with a name of the form: `zcrypt_<n>`, where *<n>* is the next free number. By default no adapter, domain, or IOCTL is allowed on the new device.

By default the device node file is created with permissions 0600 and might need adjustments to be usable by non-root users.

destroy *<device_name>*
destroys a zcrypt device. Marks the given zcrypt device as disposable. The device is removed when the use counter is zero.

addap *<device_name> <adapter>*
adds a cryptographic adapter to be accessible through this device. The adapter argument is a number in the range 0 - 255. Specify ALL to enable all adapters.

delap *<device_name> <adapter>*
removes the adapter from the allowed adapters list. The adapter argument is a number in the range 0 - 255. Specify ALL to remove all adapters.

adddom *<device_name> <domain_nr>*
adds a domain to be accessible through the specified device. The domain argument is a number in the range 0 - 255. Specify ALL to enable all domains.

deldom *<device_name> <domain_nr>*
deletes a domain from the specified device. Specify ALL to delete all domains.

addioctl*<device_name> <ioctl_exp>*
adds an IOCTL. Specify the IOCTL as a symbolic string or the corresponding numeric value in the range 0 - 255. Specify ALL to include all IOCTLs. The IOCTLs and their numbers are listed in `arch/s390/include/uapi/asm/zcrypt.h`.

Set IOCTLs according to the functions you want to support. The following table lists the IOCTLs that are required by the CCA, EP11, and libica library.

Table 82. IOCTLs required by cryptographic libraries

Library	Functions	Required IOCTLs
CCA	Secure key cryptographic functions on CCA coprocessors.	ZSECSENDCPRB
EP11	Secure key cryptographic functions on EP11 coprocessors.	ZSENDEP11CPRB
libica	Clear key cryptographic functions.	ICARSAMODEXPO, ICARSACRT, ZSECSENDCPRB

delioctl *<device_name> <ioctl_exp>*
deletes the specified IOCTL. Specify the IOCTL as a symbolic string or a numeric value in the range 0 - 255. Specify ALL to delete all IOCTLs.

config *<config_file>*
processes a configuration file.

listconfig
lists the current configuration in a format suitable for the config command.

Tip: Use **`listconfig`** to generate a configuration file that can be used as input to the **`config`** command.

Examples

These examples illustrate common uses for **`zcryptctl`**.

- To set up a zcrypt device with access to secure key operations on domain 81 of two CCA adapters, 7 and 10.

```
# zcryptctl new zcrypt_0
# zcryptctl addap zcrypt_0 7
# zcryptctl addap zcrypt_0 10
# zcryptctl adddom zcrypt_0 81
# zcryptctl addioctl zcrypt_0 ZSECSENDCPRB
```

  You might have to change the access rights to the device before a container can use it.
- To list the currently defined devices and their attributes:

```
zcryptctl list
zcdn node name: zcrypt_2
   device node: /dev/zcrypt_2
   major:minor: 250:2
   ioctls: ICARSAMODEXPO,ICARSACRT,ZSECSENDCPRB
   adapter: 4,8,9
   domains: 6,11,81
zcdn node name: zcrypt_0
   device node: /dev/zcrypt_0
   major:minor: 250:0
   ioctls: ZSECSENDCPRB
   adapter: 7,10
   domains: 81
zcdn node name: zcrypt_1
   device node: /dev/zcrypt_1
   major:minor: 250:1
   ioctls: ZSENDEP11CPRB
   adapter: 6,11
   domains: 11
```

- To remove an obsolete device.

```
# zcryptctl destroy zcrypt_0
```

Creating a configuration file

The given configuration file is read line by line and the actions are executed. The syntax is as follows:

- A node=<node_name> line creates a new device node with the given name. The subsequent actions act on this node until another node= line encountered. For example, to create a device node called zcdn_node_1:

```
node = zcdn_node_1
```

- The aps=<list_of_ap_numbers> action adds allowed adapters to the node configuration. The adapters must be separated by space, tab, or commas. For example, to add adapters 1,2,5, and 7:

```

aps = 1, 2, 5, 7
```

- The doms=<list_of_domain_numbers> action adds allowed domains to the node configuration. The domains must be separated by space, tab, or commas. For example, to allow domain 6:

```
doms = 6
```

- The ioctls=<list_of_ioctl_as_number_or_symbolic_name> adds allowed IOCTLs to the node configuration. The IOCTLs must be separated by space, tab, or commas. For example, to allow ZSECSENDCPRB:

```
ioctls = ZSECSENDCPRB
```

  The IOCTL macros, to be used as name, and their numbers are listed in arch/s390/include/uapi/asm/zcrypt.h.

The symbol ALL is also recognized for the aps, doms and ioctls actions.

Empty lines are ignored and the number sign (#) marks the rest of the line as a comment. Each action must fit on one line, multiple lines is not supported. You can use more than one aps, doms, or ioctls line to customize the same node.

Example configuration file

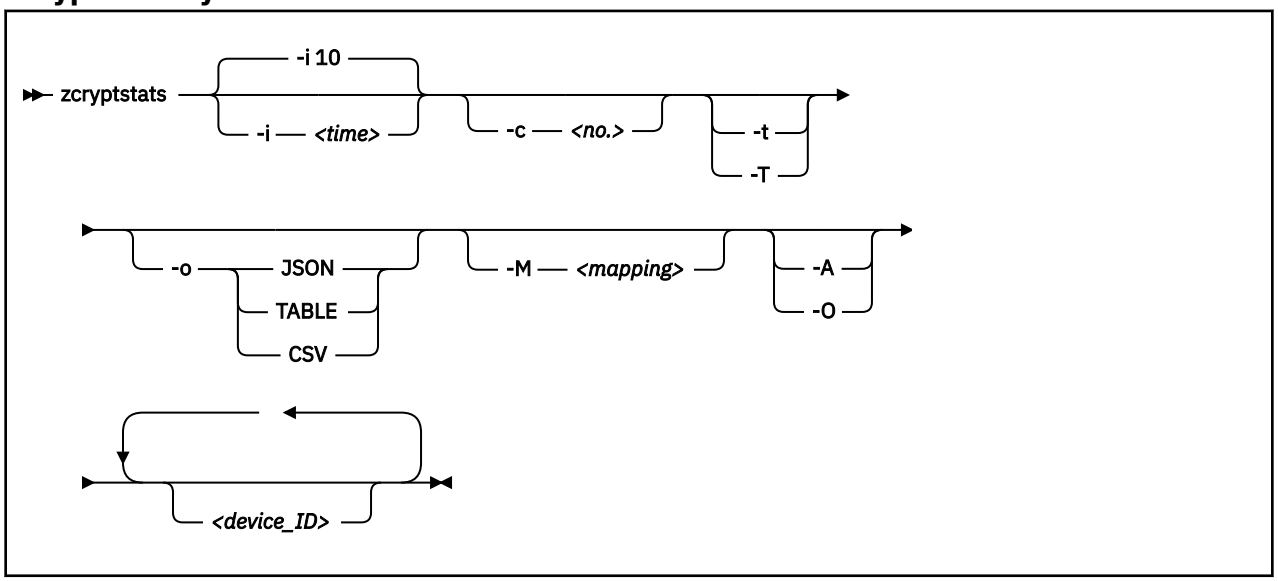
```
##############################################
# Sample zcrypt device node configuration
##############################################
# node 1 for CCA requests on domain 6
node = zcdn_node_1
aps = 1, 2, 5, 7
doms = 6
ioctls = ZSECSENDCPRB

# node 2 for CCA requests on domain 11
node = zcdn_node_2
aps = 1, 2, 5, 7
doms = 11
ioctls = ZSECSENDCPRB

# node 3 for EP11 on domain 6 and 11
node = zcdn_node_3
aps = 3, 6, 11
doms = 6, 11
ioctls = ZSENDEP11CPRB

# node 4 for clear key on everything
node = zcdn_node_4
aps = ALL
doms = ALL
ioctls = ICARSAMODEXPO, ICARSACRT, ZSECSENDCPRB

# node 5 special EP11 on adapter 10, any domain
node = zcdn_node_5
aps = 0x0a
doms = ALL
ioctls = ZSENDEP11CPRB

# node 6 special CCA only on adapter 7, domain 81
node = zcdn_node_6
aps = 7
doms = 0x51
ioctls = ZSECSENDCPRB
```

zcryptstats - Display crypto statistics

Use the **zcryptstats** command to report cryptographic performance measurement data for cryptographic devices at specified intervals.

Device node /dev/chsc must exist. Load the chsh_sch kernel module with **modprobe chsc_sch** to make device node /dev/chsc available.

zcryptstats syntax



Where:

-i or --interval *<time>*
specifies the time interval between reports in seconds. The default is 10 seconds.

-c or --count *<no.>*
specifies the number of reports to be generated. By default the **zcryptstats** command keeps generating reports until it is stopped with Ctrl+C.

-o or --output JSON|TABLE|CSV

displays the statistics in the specified format. By default, a comprehensive report is displayed in a human readable format. Supported output formats are: JSON, TABLE, or CSV.

With TABLE and CSV, only the totals are displayed, that is, TABLE and CSV formats imply the `--only-totals` option.

With JSON or the default display, you can optionally specify one of the `--only-totals` or `-no-totals` options.

-t or --no-totals
omits the totals of all counters of a cryptographic adapter (CARD) or queue device (APQN). This option cannot be specified together with the `--only-totals` option or the `--output TABLE | CSV` option.

-T or --only-totals
omits the individual counters of a cryptographic adapter or a queue device. This option is implied with the `--output TABLE | CSV` option.

-a or --no-apqn
displays only the counters of the cryptographic adapter, but omits the counters of the queue device. This option is implied for mainframes before the October 2018 upgrade of IBM z14.

-M or --map-type *<mapping>*
maps unknown cryptographic device types and modes to known types and modes. Use this option only when new, unknown cryptographic devices are found. You can map unknown devices to known devices and modes, if the new device reports the same counters as the known device. Specify the mapping as a comma-separated list of `FROM-TYPE:FROM-MODE=TO-TYPE:TO-MODE` specifications. The type and mode values must be specified in decimal notation.

-A or --all
displays all adapter devices and queue devices, not only those devices that are available to the LPAR in which Linux runs. Using this option, additional cryptographic devices that are available in the CEC are also monitored. This option cannot be specified together with the `--only-online` option.

-O or --only-online
displays only online cryptographic adapters and queue devices. This option cannot be specified together with the `--all` option.

-V or --verbose
displays additional information messages during processing.

-h or --help
displays help information for the command. To view the man page, enter **`man zcryptstats`**.

-v or --version
displays version information for the command.

<device_ID>

List of cryptographic device IDs, separated by blanks, for which statistics are displayed. Device IDs can either be cryptographic adapter IDs or queue device IDs (*<adapter_ID>.<domain_ID>*). To filter all devices according to a dedicated domain, provide ".*<domain_ID>*". If no IDs are given, all available devices are displayed.

Examples

- To display statistics for the cryptographic adapter with ID 0x02.

```
# zcryptstats 02
```

- Display statistics for domain 0x0005 on adapter 0x02 (APQN 02.0005).

```
# zcryptstats 02.0005
```

- Example of output in default display format:

```
# zcryptstats 06
Linux <version> <system> 16/03/20 s390x
******************************************************************
TIME: 16/03/20 10:48:34 INTERVAL: 1

DEVICE        TYPE                                TIMESTAMP
-----------------------------------------------------------------
06       CARD CEX7A (Accelerator)                 16/03/20 10:48:34
    COUNTER                    OPS         RATE  UTILIZATION    AVG.DURATION
    ---------------------------------------------------------------------------
    RSA 1024 ME                  0         0.00       0.00 %      0.000 usec
    RSA 2048 ME               4149      4148.81      17.46 %     42.074 usec
    RSA 1024 CRT                 0         0.00       0.00 %      0.000 usec
    RSA 2048 CRT              4564      4563.79      82.12 %    179.943 usec
    RSA 4096 ME                  0         0.00       0.00 %      0.000 usec
    RSA 4096 CTR                 0         0.00       0.00 %      0.000 usec
    ---------------------------------------------------------------------------
    Total                     8713      8712.60      99.58 %    114.292 usec

DEVICE        TYPE                                TIMESTAMP
-----------------------------------------------------------------
06.0011 APQN  CEX7A (Accelerator)                 16/01/20 10:48:34

        COUNTER                    OPS         RATE  UTILIZATION    AVG.DURATION
        ---------------------------------------------------------------------------
        RSA 1024 ME                  0         0.00       0.00 %      0.000 usec
        RSA 2048 ME               2166      2165.90      10.06 %     46.428 usec
        RSA 1024 CRT                 0         0.00       0.00 %      0.000 usec
        RSA 2048 CRT              1302      1301.94      23.31 %    179.071 usec
        RSA 4096 ME                  0         0.00       0.00 %      0.000 usec
        RSA 4096 CTR                 0         0.00       0.00 %      0.000 usec
        ---------------------------------------------------------------------------
        Total                     3468      3467.84      33.37 %     96.226 usec

DEVICE        TYPE                                TIMESTAMP
-----------------------------------------------------------------
06.0023 APQN  CEX7A (Accelerator)                 16/03/20 10:48:34
...
```

- Example of the same output in TABLE format:

```
Linux <version> <system> 16/01/20 s390x
TIMESTAMP            DEVICE        OPS         RATE  UTILIZATION    AVG.DURATION
-------------------------------------------------------------------------------------
16/03/20 10:48:34    06           8713      8712.60      99.58 %    114.292 usec
16/03/20 10:48:34    06.0011      3468      3467.84      33.37 %     96.226 usec
16/03/20 10:48:34    06.0023      3267      3266.85      30.15 %     92.278 usec
16/03/20 10:48:34    06.0024      1978      1977.91      36.06 %    182.325 usec
....
```

- Example of the same output in CSV format:

```
TIMESTAMP,DEVICE,OPS,RATE,UTILIZATION,AVG.DURATION
16/03/20 10:48:34,06,8713,8712.60,99.58 %,0.000114292
16/03/20 10:48:34,06.0011,3468,3467.84,33.37 %,0.000096226
16/03/20 10:48:34,06.0023,3267,3266.85,30.15 %,0.000092278
16/03/20 10:48:34,06.0024,1978,1977.91,36.06 %,0.000182325
....
```

- Example of the same output in JSON format:

```
{"zcryptstats": {
  "host": {
    "nodename": "lpar01",
    "sysname": "Linux",
    "release": "5.4",
    "machine": "s390x",
    "date": "16/03/20",
    "statistics": [
      {
        "interval": 1, "timestamp": "16/01/20 10:48:34", "devices": [
          {"device": "06", "type": "CEX7A (Accelerator)",
           "counters": [
            {"counter": "RSA 1024 ME", "ops": 0, "rate": 0.00,
             "utilization": 0.00, "duration": 0.000000000},
            {"counter": "RSA 2048 ME", "ops": 4149, "rate": 4148.81,
             "utilization": 17.46, "duration": 0.000042074},
            {"counter": "RSA 1024 CRT", "ops": 0, "rate": 0.00,
             "utilization": 0.00, "duration": 0.000000000},
            {"counter": "RSA 2048 CRT", "ops": 4564, "rate": 4563.79,
             "utilization": 82.12, "duration": 0.000179943},
            {"counter": "RSA 4096 ME", "ops": 0, "rate": 0.00,
             "utilization": 0.00, "duration": 0.000000000},
            {"counter": "RSA 4096 CTR", "ops": 0, "rate": 0.00,
             "utilization": 0.00, "duration": 0.000000000},
            {"counter": "Total", "ops": 8713, "rate": 8712.60,
             "utilization": 99.58, "duration": 0.000114292}
          ]},
        {"device": "06.0011", "type": "CEX7A (Accelerator)",
         "counters": [
....
```

Chapter 63. Selected kernel parameters

You can use kernel parameters that are beyond the scope of an individual device driver or feature to configure Linux in general.

Device driver-specific kernel parameters are described in the setting up section of the respective device driver.

See Chapter 4, “Kernel and module parameters,” on page 25 for information about specifying kernel parameters.

cio_ignore - List devices to be ignored

Use the `cio_ignore=` kernel parameter to list specifications for I/O devices that are to be ignored.

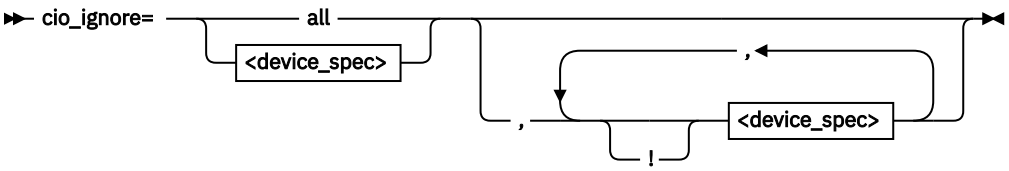
When an instance of Linux on IBM Z boots, it senses and analyzes all available I/O devices. The following applies to ignored devices:

- Ignored devices are not sensed and analyzed. The device cannot be used until it is analyzed.
- Ignored devices are not represented in sysfs.
- Ignored devices do not occupy storage in the kernel.
- The subchannel to which an ignored device is attached is treated as if no device were attached.
- For Linux on z/VM, cio_ignore might hide essential devices such as the console. The console is typically device number 0.0.0009.

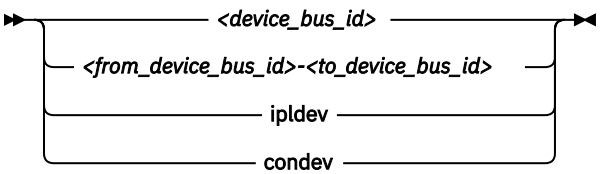
See also “Changing the exclusion list” on page 781.

Format

cio_ignore syntax



<device_spec>



Where:

all
states that all devices are to be ignored.

<device_bus_id>
specifies a device. Device bus-IDs are of the form 0.*<n>*.*<devno>*, where *<n>* is a subchannel set ID and *<devno>* is a device number.

<from_device_bus_id>-<to_device_bus_id>
are two device bus-IDs that specify the first and the last device in a range of devices.

ipldev
specifies the IPL device. Use this keyword with the `!` operator to avoid ignoring the IPL device.

condev
specifies the CCW console. Use this keyword with the `!` operator to avoid ignoring the console device.

!
makes the following term an exclusion statement. This operator is used to exclude individual devices or ranges of devices from a preceding more general specification of devices.

Examples

- This example specifies that all devices in the range 0.0.b100 through 0.0.b1ff, and the device 0.0.a100 are to be ignored.

```
cio_ignore=0.0.b100-0.0.b1ff,0.0.a100
```

- This example specifies that all devices are to be ignored.

```
cio_ignore=all
```

- This example specifies that all devices except the console are to be ignored.

```
cio_ignore=all,!condev
```

- This example specifies that all devices but the range 0.0.b100 through 0.0.b1ff, and the device 0.0.a100 are to be ignored.

```
 cio_ignore=all,!0.0.b100-0.0.b1ff,!0.0.a100
```

- This example specifies that all devices in the range 0.0.1000 through 0.0.1500 are to be ignored, except for devices in the range 0.0.1100 through 0.0.1120.

```
cio_ignore=0.0.1000-0.0.1500,!0.0.1100-0.0.1120
```

  This is equivalent to the following specification:

```
cio_ignore=0.0.1000-0.0.10ff,0.0.1121-0.0.1500
```

- This example specifies that all devices in range 0.0.1000 through 0.0.1100 and all devices in range 0.1.7000 through 0.1.7010, plus device 0.0.1234 and device 0.1.4321 are to be ignored.

```
cio_ignore=0.0.1000-0.0.1100, 0.1.7000-0.1.7010, 0.0.1234, 0.1.4321
```

Changing the exclusion list

Use the **cio_ignore** command or the procfs interface to view or change the list of I/O device specifications that are ignored.

When an instance of Linux on IBM Z boots, it senses and analyzes all available I/O devices. You can use the cio_ignore kernel parameter to list specifications for devices that are to be ignored.

On a running Linux instance, you can view and change the exclusion list through a procfs interface.

After booting Linux you can display the exclusion list by issuing:

```
# cat /proc/cio_ignore
```

To add device specifications to the exclusion list issue a command of this form:

```
# echo add <device_list> > /proc/cio_ignore
```

When you add specifications for a device that is already sensed and analyzed, there is no immediate effect of adding it to the exclusion list. For example, the device still appears in the output of the **lscss** command and can be set online. However, if the device later becomes unavailable, it is ignored when it reappears. For example, if the device is detached in z/VM it is ignored when it is attached again.

To make all devices that are in the exclusion list and that are currently offline unavailable to Linux issue a command of this form:

```
# echo purge > /proc/cio_ignore
```

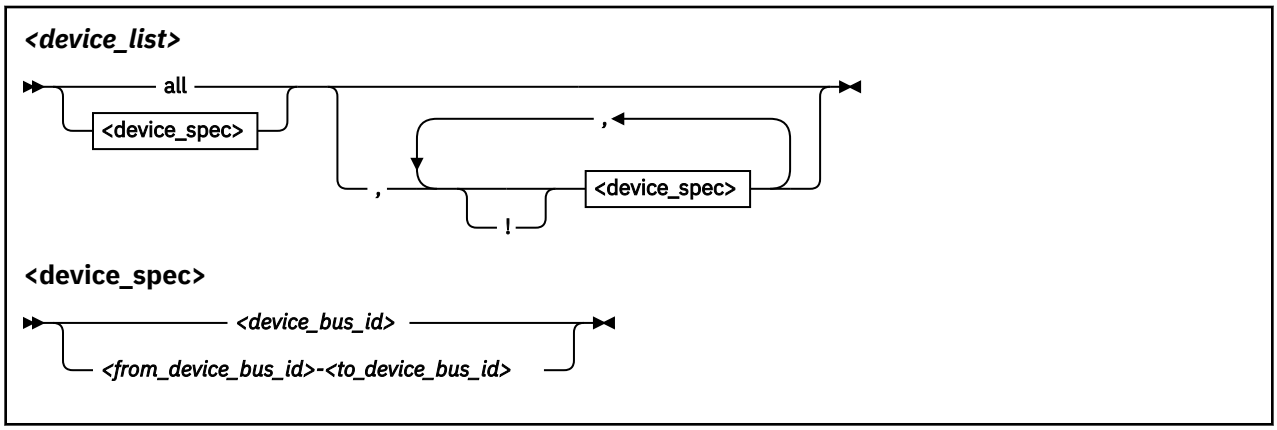
This command does not make devices unavailable if they are online.

To remove device specifications from the exclusion list issue a command of this form:

```
# echo free <device_list> > /proc/cio_ignore
```

When you remove device specifications from the exclusion list, the corresponding devices are sensed and analyzed if they exist. Where possible, the respective device driver is informed, and the devices become available to Linux.

In these commands, *<device_list>* follows this syntax:



Where the keywords and variables have the same meaning as in "Format" on page 780.

Ensure device availability

After the echo command completes successfully, some time might elapse until the freed device becomes available to Linux. To confirm that a device has become available to Linux verify that the sysfs attribute `/sys/bus/ccw/devices/<device_bus_ID>/online` is present.

Results

The dynamically changed exclusion list is only taken into account when a device in this list is newly made available to the system, for example after it is defined to the system. It does not have any effect on setting devices online or offline within Linux.

Examples

- This command removes all devices from the exclusion list.

```
# echo free all > /proc/cio_ignore
```

- This command adds all devices in the range 0.0.b100 through 0.0.b1ff and device 0.0.a100 to the exclusion list.

```
# echo add 0.0.b100-0.0.b1ff,0.0.a100 > /proc/cio_ignore
```

- This command lists the ranges of devices that are ignored by common I/O.

```
# cat /proc/cio_ignore
0.0.0000-0.0.a0ff
0.0.a101-0.0.b0ff
0.0.b200-0.0.ffff
```

- This command removes all devices in the range 0.0.b100 through 0.0.b1ff and device 0.0.a100 from the exclusion list.

```
# echo free 0.0.b100-0.0.b1ff,0.0.a100 > /proc/cio_ignore
```

- This command removes the device with bus ID 0.0.c104 from the exclusion list.

```
# echo free 0.0.c104 > /proc/cio_ignore
```

- This command adds the device with bus ID 0.0.c104 to the exclusion list.

```
# echo add 0.0.c104 > /proc/cio_ignore
```

- This command makes all devices that are in the exclusion list and that are currently offline unavailable to Linux.

```
# echo purge > /proc/cio_ignore
```

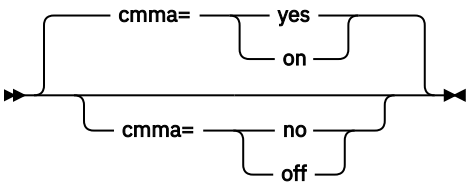
cmma - Reduce hypervisor paging I/O overhead

Use the `cmma=` kernel parameter to reduce hypervisor paging I/O overhead.

You can use Collaborative Memory Management Assist (CMMA, or "cmm2") on all IBM mainframe systems that Red Hat Enterprise Linux 9.1 supports. With this support, the z/VM control program and guest virtual machines can communicate attributes for specific 4K-byte blocks of guest memory. This exchange of information helps both the z/VM host and the guest virtual machines to optimize their use and management of memory.

Format

cmma syntax



Examples

This specification disables the CMMA support:

```
cmma=off
```

Alternatively, you can use the following specification to disable the CMMA support:

```
cmma=no
```

fips - Run Linux in FIPS mode

In Federal Information Processing Standard (FIPS) mode, the kernel enforces FIPS 140-2 security standards. For example, in FIPS mode only FIPS 140-2 approved encryption algorithms can be used (see “FIPS restrictions of the hardware capabilities” on page 532).

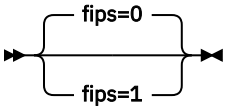
Note: Enabling FIPS mode is not sufficient to make your kernel certified according to FIPS 140-2.

FIPS 140-2 certification is specific to a particular hardware platform and kernel build. Typically, running in FIPS mode is required, but not sufficient to be FIPS 140-2 certified. Check with your distributor to find out whether your kernel is certified according to FIPS 140-2.

For more information about FIPS 140-2, go to csrc.nist.gov/publications/detail/fips/140/2/final.

Format

fips syntax



1 enables the FIPS mode. 0, the default, disables the FIPS mode.

Example

```
fips=1
```

maxcpus - Limit the number of CPUs Linux can use at IPL

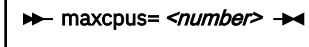
Use the `maxcpus=` kernel parameter to limit the number of CPUs that Linux can use at IPL and that are online after IPL.

If the real or virtual hardware provides more than the specified number of CPUs, these surplus CPUs are initially offline. For example, if five CPUs are available, `maxcpus=2` results in two online CPUs and three offline CPUs after IPL.

Offline CPUs can be set online dynamically unless the `possible_cpus=` parameter is set and specifies a maximum number of online CPUs that is already reached. The `possible_cpus=` parameter sets an absolute limit for the number of CPUs that can be online at any one time (see possible_cpus). If both `maxcpus=` and `possible_cpus=` are set, a lower value for `possible_cpus=` overrides `maxcpus=` and makes it ineffective.

Format

maxcpus syntax



Examples

```
maxcpus=2
```

nokaslr - Disable kernel randomization

By default, Linux uses KASLR. Specify the `nokaslr` kernel parameter to disable kernel randomization, that is, cause the kernel to be loaded at its standard location.

For more information about kernel randomization, see Chapter 10, “KASLR support,” on page 133.

Format

nokaslr kernel parameter syntax

nosmt - Disable simultaneous multithreading

By default, Linux in LPAR mode uses simultaneous multithreading if it is supported by the hardware. Specify the `nosmt` kernel parameter to disable simultaneous multithreading. See also “smt - Reduce the number of threads per core” on page 795.

For more information about simultaneous multithreading, see “Simultaneous multithreading” on page 357.

Format

nosmt syntax

novx - Disable the Vector Extension Facility

By default, Linux uses the Vector Extension Facility if it is supported by the hardware. Specify the novx kernel parameter to disable the Vector Extension Facility.

Do not disable the Vector Extension Facility for regular operations. This parameter is intended for test and diagnostics.

Format

novx syntax

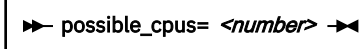
possible_cpus - Limit the number of CPUs Linux can use

Use the `possible_cpus=` parameter to set an absolute limit for the number of CPUs that can be online at any one time. If the real or virtual hardware provides more than the specified maximum, the surplus number of CPUs must be offline. Alternatively, you can use the common code kernel parameter `nr_cpus`.

Use the `maxcpus=` parameter to limit the number of CPUs that are online initially after IPL (see maxcpus).

Format

possible_cpus syntax



Examples

```
possible_cpus=8
```

ramdisk_size - Specify the ramdisk size

Use the `ramdisk_size=` kernel parameter to specify the size of the ramdisk in kilobytes.

Format

ramdisk_size syntax

Examples

```
ramdisk_size=32000
```

rd.zdev=no-auto - Override initial device availability for DPM mode

Use the `rd.zdev=no-auto` kernel parameter to override device availability defaults for Linux in a DPM partition.

Format

rd.zdev syntax

Device exclusion list

As a default for Linux in a DPM partition, auto-configuration data overrides device exclusion through `cio_ignore`. With `rd.zdev=no-auto`, the auto-configuration data is ignored and the exclusion list is enforced unchanged.

For more information about auto-configuration data, see Chapter 3, “Device auto-configuration for Linux in LPAR mode,” on page 21. For `cio_ignore`, see “cio_ignore - Manage the I/O exclusion list” on page 597.

PCIe or cryptographic devices

With the s390-utils package installed, the initial online status of PCIe devices and cryptographic devices for Linux in a DPM partition is "online". Likewise, these devices are online after a reboot even if they were explicitly set offline before the reboot.

Using `rd.zdev=no-auto`, the initial online status of PCIe devices and cryptographic devices depends on defaults of your hardware and firmware levels. The online status of these devices on a running Linux instance is then preserved across reboots.

Linux in traditional LPAR mode

The `rd.zdev=no-auto` parameter does not affect Linux in traditional LPAR mode. After a reboot in this environment, the `cio_ignore` exclusion list is always enforced, and PCIe and cryptographic devices preserve their online status. The initial online status of a PCIe or cryptographic device depends on its hardware definition.

ro - Mount the root file system read-only

Use the `ro` kernel parameter to mount the root file system read-only.

Format

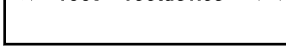
ro syntax

root - Specify the root device

Use the `root=` kernel parameter to tell Linux what to use as the root when mounting the root file system.

Format

root syntax



Examples

This example makes Linux use `/dev/dasda1` when mounting the root file system:

```
root=/dev/dasda1
```

smt - Reduce the number of threads per core

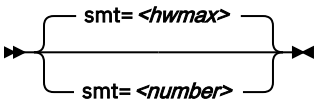
By default, Linux in LPAR mode uses the maximum number of threads per core that is supported by the hardware. Use the `smt=` kernel parameter to use fewer threads. The value can be any integer in the range 1 to the maximum number of threads that is supported by the hardware.

Specifying `smt=1` effectively disables simultaneous multithreading. See also “nosmt - Disable simultaneous multithreading” on page 788.

For more information about simultaneous multithreading, see “Simultaneous multithreading” on page 357.

Format

smt syntax



where *<hwmax>* is the maximum number of threads per core that is supported by the hardware, and *<number>* is an integer in the range 1 - *<hwmax>*.

Examples

```
smt=1
```

vdso - Optimize system call performance

Use the `vdso=` kernel parameter to control the vdso support for the `gettimeofday`, `clock_getres`, and `clock_gettime` system calls.

The virtual dynamic shared object (vdso) support is a shared library that the kernel maps to all dynamically linked programs. The glibc detects the presence of the vdso and uses the functions that are provided in the library.

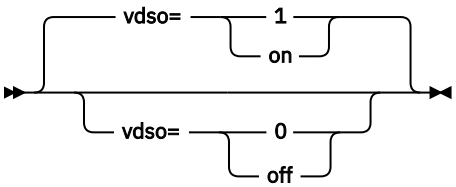
Because the vdso library is mapped to all user-space processes, this change is visible in user space. In the unlikely event that a user-space program does not work with the vdso support, you can disable the support.

The default, which is to use vdso support, works well for most installations. Do not override this default, unless you observe problems.

The vdso support is included in the Linux kernel.

Format

vdso syntax



Examples

This example disables the vdso support:

```
vdso=0
```

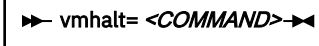
vmhalt - Specify CP command to run after a system halt

Use the `vmhalt=` kernel parameter to specify a command to be issued to CP after a system halt.

This command applies only to Linux on z/VM.

Format

vmhalt syntax



Examples

This example specifies that an initial program load of CMS is to follow the Linux **`halt`** command:

```
vmhalt="CPU 00 CMD I CMS"
```

Note: The command must be entered in uppercase.

vmpanic - Specify CP command to run after a kernel panic

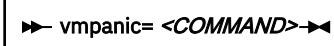
Use the `vmpanic=` kernel parameter to specify a command to be issued to CP after a kernel panic.

This command applies only to Linux on z/VM.

Note: Ensure that the **dumpconf** service is disabled when you use this kernel parameter. Otherwise, **dumpconf** will override the setting.

Format

vmpanic syntax



Examples

This example specifies that a VMDUMP is to follow a kernel panic:

```
vmpanic="VMDUMP"
```

Note: The command must be entered in uppercase.

vmpoff - Specify CP command to run after a power off

Use the `vmpoff=` kernel parameter to specify a command to be issued to CP after a system power off.

Format

vmpoff syntax

Examples

This example specifies that CP is to clear the guest virtual machine after the Linux **`power off`** or **`halt -p`** command:

```
vmpoff="SYSTEM CLEAR"
```

Note: The command must be entered in uppercase.

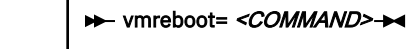
vmreboot - Specify CP command to run on reboot

Use the `vmreboot=` kernel parameter to specify a command to be issued to CP on reboot.

This command applies only to Linux on z/VM.

Format

vmreboot syntax



Examples

This example specifies a message to be sent to the z/VM guest virtual machine OPERATOR if a reboot occurs:

```
vmreboot="MSG OPERATOR Reboot system"
```

Note: The command must be entered in uppercase.

Chapter 64. Linux diagnose code use

Red Hat Enterprise Linux 9.1 issues several diagnose instructions to the hypervisor (LPAR, z/VM, or KVM).

Read `/sys/kernel/debug/diag_stat` to find out which diagnose instructions are called how frequently on your Linux instance.

```
# cat /sys/kernel/debug/diag_stat
                CPU0       CPU1
diag 008:          7          7    Console Function
diag 00c:          0          0    Pseudo Timer
diag 010:          0          0    Release Pages
diag 014:          0          0    Spool File Services
diag 044:          7          5    Voluntary Timeslice End
diag 064:          0          0    NSS Manipulation
diag 09c:         53         92    Relinquish Timeslice
diag 0dc:          0          0    Appldata Control
diag 204:          2          0    Logical-CPU Utilization
diag 210:          3          6    Device Information
diag 224:          0          0    EBCDIC-Name Table
diag 250:          0          0    Block I/O
diag 258:          1          1    Page-Reference Services
diag 26c:          2          0    Certain System Information
diag 288:          0          0    Time Bomb
diag 2c4:          0          0    FTP Services
diag 2fc:          2          0    Guest Performance Data
diag 304:          0          0    Partition-Resource Service
diag 308:          1          1    List-Directed IPL
diag 318:          1          1    CP Name and Version Codes
diag 500:          0          0    Virtio Service
```

The z/VM configuration can modify and restrict the diagnose calls that are available to its guests (see “Function unavailable or degraded in Linux on z/VM” on page 564).

Appendix A. Accessibility

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

Documentation accessibility

The Linux on IBM Z and LinuxONE publications are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF file and want to request a Web-based format for this publication send an email to eservdoc@de.ibm.com or write to:

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany

In the request, be sure to include the publication number and title.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility at

```
www.ibm.com/able
```

Appendix B. Understanding syntax diagrams

This section describes how to read the syntax diagrams in this manual.

To read a syntax diagram follow the path of the line. Read from left to right and top to bottom.

- The ►►—— symbol indicates the beginning of a syntax diagram.
- The ——► symbol, at the end of a line, indicates that the syntax diagram continues on the next line.
- The ►—— symbol, at the beginning of a line, indicates that a syntax diagram continues from the previous line.
- The ——►◄ symbol indicates the end of a syntax diagram.

Syntax items (for example, a keyword or variable) may be:

- Directly on the line (required)
- Above the line (default)
- Below the line (optional)

If defaults are determined by your system status or settings, they are not shown in the diagram. Instead the rule is described together with the option, keyword, or variable in the list following the diagram.

Case sensitivity
Unless otherwise noted, entries are case sensitive.

Symbols
You **must** code these symbols exactly as they appear in the syntax diagram

*****
Asterisk

:
Colon

,
Comma

=
Equal sign

-
Hyphen

//
Double slash

( )
Parentheses

.
Period

+
Add

$
Dollar sign

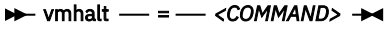
For example:

```
dasd=0.0.7000-0.0.7fff
```

Variables
An *<italicized>* lowercase word enclosed in angled brackets indicates a variable that you must substitute with specific information. For example:

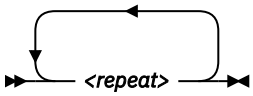
Here you must code **-p** as shown and supply a value for *<interface>*.

An italicized uppercase word in angled brackets indicates a variable that must appear in uppercase:

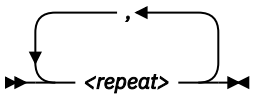


Repetition

An arrow returning to the left means that the item can be repeated.

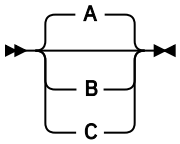


A character within the arrow means you must separate repeated items with that character.



Defaults

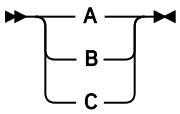
Defaults are above the line. The system uses the default unless you override it. You can override the default by coding an option from the stack below the line. For example:



In this example, A is the default. You can override A by choosing B or C.

Required Choices

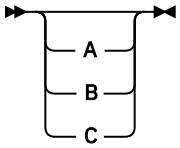
When two or more items are in a stack and one of them is on the line, you **must** specify one item. For example:



Here you must enter either A or B or C.

Optional Choice

When an item is below the line, the item is optional. Only one item **may** be chosen. For example:



Here you may enter either A or B or C, or you may omit the field.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Adobe is either a registered trademark or trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Red Hat® is a trademark or registered trademark of Red Hat, Inc. or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Glossary

This glossary includes IBM product terminology as well as selected other terms and definitions.

Additional information can be obtained in:

- The American National Standard Dictionary for Information Systems , ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036.
- The ANSI/EIA Standard–440-A, Fiber Optic Terminology. Copies may be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue, N.W., Washington, DC 20006.
- The Information Technology Vocabulary developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1).
- The IBM Dictionary of Computing , New York: McGraw-Hill, 1994.
- Internet Request for Comments: 1208, Glossary of Networking Terms
- Internet Request for Comments: 1392, Internet Users' Glossary
- The Object-Oriented Interface Design: IBM Common User Access Guidelines , Carmel, Indiana: Que, 1992.

Numerics

10 Gigabit Ethernet

An Ethernet network with a bandwidth of 10000-Mbps.

3215

IBM console printer-keyboard.

3270

IBM information display system.

3370, 3380 or 3390

IBM direct access storage device (disk).

3480, 3490, 3590

IBM magnetic tape subsystem.

3DES

See Triple Data Encryption Standard.

9336 or 9345

IBM direct access storage device (disk).

A

address space

The range of addresses available to a computer program or process. Address space can refer to physical storage, virtual storage, or both.

auto-detection

Listing the addresses of devices attached to a card by issuing a query command to the card.

C

CEC

(Central Electronics Complex). A synonym for *CPC*.

channel subsystem

The programmable input/output processors of the mainframe, which operate in parallel with the CPU.

checksum

An error detection method using a check byte appended to message data

CHPID

channel path identifier. In a channel subsystem, a value assigned to each installed channel path of the system that uniquely identifies that path to the system.

compatible disk layout

A disk structure for Linux on IBM Z which allows access from other mainframe operating systems. This replaces the older Linux disk layout.

Console

In Linux, an output device for kernel messages.

CPC

(Central Processor Complex). A physical collection of hardware that includes main storage, one or more central processors, timers, and channels. Also referred to as a *CEC*.

CRC

cyclic redundancy check. A system of error checking performed at both the sending and receiving station after a block-check character has been accumulated.

CSMA/CD

carrier sense multiple access with collision detection

CTC

channel to channel. A method of connecting two computing devices.

CUU

control unit and unit address. A form of addressing for mainframe devices using device numbers.

D

DASD

direct access storage device. A mass storage medium on which a computer stores data.

device driver

- A file that contains the code needed to use an attached device.
- A program that enables a computer to communicate with a specific peripheral device; for example, a printer, a videodisc player, or a CD-ROM drive.
- A collection of subroutines that control the interface between I/O device adapters and the processor.

DIAGNOSE

In z/VM, a set of instructions that programs running on z/VM guest virtual machines can call to request CP services.

disconnected device

In Linux on IBM Z, a device that is online, but to which Linux can no longer find a connection. Reasons include:

- The device was physically removed
- The device was logically removed, for example, with a CP DETACH command in z/VM
- The device was varied offline

E

ECKD

extended count-key-data device. A disk storage device that has a data transfer rate faster than some processors can utilize and that is connected to the processor through use of a speed matching buffer. A specialized channel program is needed to communicate with such a device.

ESCON

enterprise systems connection. A set of IBM products and services that provide a dynamically connected environment within an enterprise.

Ethernet

A 10-Mbps baseband local area network that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and delayed retransmission. Ethernet uses CSMA/CD.

F

Fast Ethernet (FENET)

Ethernet network with a bandwidth of 100 Mbps

FBA

fixed block architecture. An architecture for a virtual device that specifies the format of and access mechanisms for the virtual data units on the device. The virtual data unit is a block. All blocks on the device are the same size (fixed size). The system can access them independently.

FDDI

fiber distributed data interface. An American National Standards Institute (ANSI) standard for a 100-Mbps LAN using optical fiber cables.

fibre channel

A technology for transmitting data between computer devices. It is especially suited for attaching computer servers to shared storage devices and for interconnecting storage controllers and drives.

FTP

file transfer protocol. In the Internet suite of protocols, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

G

Gigabit Ethernet (GbE)

An Ethernet network with a bandwidth of 1000-Mbps

H

hardware console

A service-call logical processor that is the communication feature between the main processor and the service processor.

Host Bus Adapter (HBA)

An I/O controller that connects an external bus, such as a Fibre Channel, to the internal bus (channel subsystem).

In a Linux environment HBAs are normally virtual and are shown as an FCP device.

HMC

hardware management console. A console used to monitor and control hardware such as the IBM Z microprocessors.

HFS

hierarchical file system. A system of arranging files into a tree structure of directories.

I

intraensemble data network (IEDN)

A private 10 Gigabit Ethernet network for application data communications within an ensemble. Data communications for workloads can flow over the IEDN within and between nodes of an ensemble. All of the physical and logical resources of the IEDN are configured, provisioned, and managed by the Unified Resource Manager.

intranode management network (INMN)

A private 1000BASE-T Ethernet network operating at 1 Gbps that is required for the Unified Resource Manager to manage the resources within a single zEnterprise node. The INMN connects the Support Element (SE) to the zEnterprise 196 (z196) or zEnterprise 114 (z114) and to any attached zEnterprise BladeCenter Extension (zBX).

ioctl system call

Performs low-level input- and output-control operations and retrieves device status information. Typical operations include buffer manipulation and query of device mode or status.

IOCS

input / output channel subsystem. See channel subsystem.

IP

internet protocol. In the Internet suite of protocols, a connectionless protocol that routes data through a network or interconnected networks and acts as an intermediary between the higher protocol layers and the physical network.

IP address

The unique 32-bit address that specifies the location of each device or workstation on the Internet. For example, 9.67.97.103 is an IP address.

IPIP

IPv4 in IPv4 tunnel, used to transport IPv4 packets in other IPv4 packets.

IPL

initial program load (or boot).

- The initialization procedure that causes an operating system to commence operation.
- The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction.
- The process of loading system programs and preparing a system to run jobs.

IPv6

IP version 6. The next generation of the Internet Protocol.

IUCV

inter-user communication vehicle. A z/VM facility for passing data between virtual machines and z/VM components.

K

kernel

The part of an operating system that performs basic functions such as allocating hardware resources.

kernel module

A dynamically loadable part of the kernel, such as a device driver or a file system.

kernel image

The kernel when loaded into memory.

L

LCS

LAN channel station. A protocol used by OSA.

LDP

Linux Documentation Project. An attempt to provide a centralized location containing the source material for all open source Linux documentation. Includes user and reference guides, HOW TOs, and FAQs. The homepage of the Linux Documentation Project is

```
www.linuxdocs.org
```

Linux

a variant of UNIX which runs on a wide range of machines from wristwatches through personal and small business machines to enterprise systems.

Linux disk layout

A basic disk structure for Linux on IBM Z. Now replaced by compatible disk layout.

Linux on IBM Z

the port of Linux to the IBM mainframe architecture.

LPAR

logical partition of a mainframe. Logical partitions are, in practice, equivalent to separate mainframes; each LPAR runs its own operating system.

LVS (Linux virtual server)

Network sprayer software used to dispatch, for example, http requests to a set of web servers to balance system load.

M

MAC

medium access control. In a LAN this is the sub-layer of the data link control layer that supports medium-dependent functions and uses the services of the physical layer to provide services to the logical link control (LLC) sub-layer. The MAC sub-layer includes the method of determining when a device has access to the transmission medium.

Mbps

million bits per second.

MIB (Management Information Base)

- A collection of objects that can be accessed by means of a network management protocol.

- A definition for management information that specifies the information available from a host or gateway and the operations allowed.

MTU

maximum transmission unit. The largest block which may be transmitted as a single unit.

Multicast

A protocol for the simultaneous distribution of data to a number of recipients, for example live video transmissions.

N

NIC

network interface card. The physical interface between the IBM mainframe and the network.

O

OSD

OSA device for QDIO.

OSA-Express

Abbreviation for Open Systems Adapter-Express networking features. These include 10 Gigabit Ethernet, Gigabit Ethernet, and Fast Ethernet.

OSPF

open shortest path first. A function used in route optimization in networks.

OSX

OSA-Express for zBX. A CHPID type that provides connectivity and access control to the intraensemble data network (IEDN) from z196 or z114 to zBX.

P

POR

power-on reset

POSIX

Portable Operating System Interface for Computer Environments. An IEEE operating system standard closely related to the UNIX system.

Q

QDIO

Queued Direct I/O.

A hardware channel architecture for direct data exchange with I/O devices, where both the I/O device and the program running on the server reference main storage directly through a set of data queues. The QDIO

architecture is used by Open Systems Adapter-Express (OSA-Express), HiperSockets, and Fiber Channel Protocol (FCP) channels.

R

router

A device or process which allows messages to pass between different networks.

S

SE

support element.

- An internal control element of a processor that assists in many of the processor operational functions.
- A hardware unit that provides communications, monitoring, and diagnostic functions to a central processor complex.

SNA

systems network architecture. The IBM architecture that defines the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. The layered structure of SNA allows the ultimate origins and destinations of information (the users) to be independent of and unaffected by the specific SNA network services and facilities that are used for information exchange.

SNMP (Simple Network Management Protocol)

In the Internet suite of protocols, a network management protocol that is used to monitor routers and attached networks. SNMP is an application layer protocol. Information about devices managed is defined and stored in the application's Management Information Base (MIB).

Sysctl

system control programming manual control (frame). A means of dynamically changing certain Linux kernel parameters during operation.

T

TDEA

See Triple Data Encryption Standard.

TDES

See Triple Data Encryption Standard.

Telnet

A member of the Internet suite of protocols which provides a remote terminal connection service. It allows users of one host to log on to a remote host and interact as if they were using a terminal directly attached to that host.

Terminal

A physical or emulated device, associated with a keyboard and display device, capable of sending and receiving information.

Triple Data Encryption Standard

A block cipher algorithm that can be used to encrypt data transmitted between managed systems and the management server. Triple DES is a security enhancement of DES that employs three successive DES block operations.

U

UNIX

An operating system developed by Bell Laboratories that features multiprogramming in a multiuser environment. The UNIX operating system was originally developed for use on minicomputers but has been adapted for mainframes and microcomputers.

V

V=R

In z/VM, a guest whose real memory (virtual from a z/VM perspective) corresponds to the real memory of z/VM.

V=V

In z/VM, a guest whose real memory (virtual from a z/VM perspective) corresponds to virtual memory of z/VM.

Virtual LAN (VLAN)

A group of devices on one ore more LANs that are configured (using management software) so that they can communicate as if they were attached to the same wire, when in fact they are located on a number of different LAN segments. Because VLANs are based on logical rather than physical connections, they are extremely flexible.

volume

A data carrier that is usually mounted and demounted as a unit, for example a tape cartridge or a disk pack. If a storage unit has no demountable packs the volume is the portion available to a single read/write mechanism.

Z

z114

IBM zEnterprise 114

z13

IBM z13

z14

IBM z14.

z13s

IBM z13s.

z196

IBM zEnterprise 196

zBC12

IBM zEnterprise BC12.

zBX

IBM zEnterprise BladeCenter Extension

zEC12

IBM zEnterprise EC 12.

zEnterprise

IBM zEnterprise System. A heterogeneous hardware infrastructure that can consist of an IBM zEnterprise BC12, a zEnterprise EC12 (zEC12), a zEnterprise 114 (z114) or a zEnterprise 196 (z196) and an attached IBM zEnterprise BladeCenter Extension (zBX), managed as a single logical virtualized system by the Unified Resource Manager.

zSeries

The family of IBM enterprise servers that demonstrate outstanding reliability, availability, scalability, security, and capacity in today's network computing environments.

Bibliography

The publications listed here are considered useful for a more detailed study of the topics contained in this book.

Linux on IBM Z and LinuxONE publications

The Linux on IBM Z and LinuxONE publications can be found on the IBM Documentation website.

You can find the latest version of this publication at https://www.ibm.com/docs/en/linux-on-systems?topic=distributions-red-hat-enterprise-linux.

- *Using the Dump Tools on Red Hat Enterprise Linux 9.1*, SC34-7751

For each of the following publications, the same web page points to the version that most closely reflects Red Hat Enterprise Linux 9.1:

- *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413
- *How to Improve Performance with PAV*, SC33-8414
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596
- *libica Programmer's Reference*, SC34-2602
- *libzpc - A Protected-Key Cryptographic Library*, SC34-7731
- *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294
- *openCryptoki - An Open Source Implementation of PKCS #11*, SC34-7730
- *Troubleshooting*, SC34-2612
- *Configuring Crypto Express Adapters for KVM Guests*, SC34-7717
- *Pervasive Encryption for Data Volumes*, SC34-2782
- *Enterprise Key Management for Pervasive Encryption of Data Volumes*, SC34-7740
- *How to set an AES master key*, SC34-7712
- *KVM Virtual Server Management*, SC34-2752
- *Configuring Crypto Express Adapters for KVM Guests*, SC34-7717
- *Introducing IBM Secure Execution for Linux*, SC34-7721
- *Networking with RoCE Express*, SC34-7745

Red Hat Enterprise Linux 9.1 publications

The documentation for Red Hat Enterprise Linux 9.1 can be found on the Red Hat documentation website.

See https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux

- *Performing a standard RHEL installation* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/performing_a_standard_rhel_installation/index#installing-rhel-on-ibm-z)
- *Configuring basic system settings* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_basic_system_settings/)
- *Configuring and managing networking in Red Hat Enterprise Linux 9.1* (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_and_managing_networking/index)

z/VM publications

The publication numbers listed are for z/VM version 7.

For the complete library including other versions, see

```
www.ibm.com/vm/library
```

- *z/VM: Connectivity*, SC24-6267
- *z/VM: CP Commands and Utilities Reference*, SC24-6268
- *z/VM: CP Planning and Administration*, SC24-6271
- *z/VM: CP Programming Services*, SC24-6272
- *z/VM: Getting Started with Linux on System z*, SC24-6287
- *z/VM: Performance*, SC24-6301
- *z/VM: Saved Segments Planning and Administration*, SC24-6322
- *z/VM: Systems Management Application Programming*, SC24-6327
- *z/VM: TCP/IP Planning and Customization*, SC24-6331
- *z/VM: Virtual Machine Operation*, SC24-6334
- *z/VM: REXX/VM Reference*, SC24-6314
- *z/VM: REXX/VM User's Guide*, SC24-6315

IBM Redbooks publications

You can search for, view, or download Redbooks publications, Redpapers, Hints and Tips, draft publications and additional materials on the Redbooks website.

You can also order hardcopy Redbooks or CD-ROMs. See

```
www.ibm.com/redbooks
```

- *IBM zEnterprise Unified Resource Manager*, SG24-7921
- *Building Linux Systems under IBM VM*, REDP-0120
- *FICON CTC Implementation*, REDP-0158
- *Networking Overview for Linux on zSeries*, REDP-3901
- *Linux on IBM eServer zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP-3596
- *Linux on IBM eServer zSeries and S/390: VSWITCH and VLAN Features of z/VM 4.4*, REDP-3719
- *z/VM: Secure Configuration Guide*, SG24-6323
- *IBM Communication Controller Migration Guide*, SG24-6298
- *Problem Determination for Linux on System z*, SG24-7599
- *Fibre Channel Protocol for Linux and z/VM on IBM System z*, SG24-7266

Other IBM Z publications

General IBM Z publications that might be of interest in the context of Linux on IBM Z.

- *System z Application Programming Interfaces*, SB10-7030
- *IBM DS8000 Series Command-Line Interface User's Guide*, GC27-4212
- *Processor Resource/Systems Manager Planning Guide*, SB10-7041
- *z/Architecture Principles of Operation*, SA22-7832
- *z/Architecture The Load-Program-Parameter and the CPU-Measurement Facilities*, SA23-2260

- *IBM The CPU-Measurement Facility Extended Counters Definition for z10, z196, z114 and zEC12*, SA23-2261
- *IBM The CPU-Measurement Facility Extended Counters Definition for z10, z196/z114, zEC12/zBC12, z13/z13s, and z14*, SA23-2261

Networking publications

- *HiperSockets Implementation Guide*, SG24-6816
- *Open Systems Adapter-Express Customer's Guide and Reference*, SA22-7935
- *OSA-Express Implementation Guide*, SG24-5948

Security related publications

- *zSeries Crypto Guide Update*, SG24-6870

ibm.com resources

On the ibm.com® website you can find information about many aspects of Linux on IBM Z and LinuxONE including z/VM, I/O connectivity, and cryptography.

- For CMS and CP Data Areas, Control Block information, and the layout of the z/VM monitor records see

```
www.ibm.com/vm/pubs/ctlblk.html
```

- For I/O connectivity information, see

```
www.ibm.com/systems/support/storage/ssic/interoperability.wss
```

- For I/O networks to servers and storage devices to deliver high-performing, secure networking and connectivity, see

```
www.ibm.com/it-infrastructure/z/capabilities/networking
```

- For Communications server for Linux information, see

```
www.ibm.com/software/network/commserver/linux
```

- For information about performance monitoring on z/VM, see

```
www.ibm.com/vm/perf
```

- For cryptographic coprocessor information, see

```
www.ibm.com/security/cryptocards
```

- (Requires registration.) For information for planning, installing, and maintaining IBM systems, see

```
www.ibm.com/servers/resourcelink
```

- For information about STP, see

```
www.ibm.com/systems/z/advantages/pso/stp.html
```

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Part Number:

(1P) P/N:

SC34-7750-00