Linux on System z

Device Drivers, Features, and Commands on Red Hat Enterprise Linux 6

SC34-2597-00

Linux on System z

Device Drivers, Features, and Commands on Red Hat Enterprise Linux 6

SC34-2597-00

Note

Before using this document, be sure to read the information in “Notices” on page 481.

This edition applies to Red Hat Enterprise Linux 6 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this document

For the latest version of this document see the Linux on System z® pages on the developerWorks® website at

`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

This document describes the device drivers, features, and commands available to Red Hat Enterprise Linux 6 for the control of IBM® System 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 6 for System z.

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

In this document, System z is taken to include all IBM mainframe systems supported by Red Hat Enterprise Linux 6 for System z.

For what is new, known issues, and frequently asked questions, see the Red Hat Enterprise Linux 6 release notes at

`http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/`

Using sysfs vs. 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 *Red Hat Enterprise Linux 6 Deployment Guide* for details about the configuration files to use.

How this document is organized

The first part of this document contains general and overview information for the System z device drivers for Red Hat Enterprise Linux 6 for System z.

Part two contains chapters specific to individual storage device drivers.

Part three contains chapters specific to individual network device drivers.

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

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 about device drivers and features that support security aspects of Red Hat Enterprise Linux 6 for System z.

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

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

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

Who should read this document

Most of the information in this document is intended for system administrators who want to configure Red Hat Enterprise Linux 6 for System z.

Some sections are of interest primarily to specialists who want to program extensions to the System z device drivers and features for Red Hat Enterprise Linux 6. These sections are marked with the same icon on the left margin as this paragraph.

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, System z terminology.
- You are familiar with Linux device driver software.
- You are familiar with the System z devices attached to your system.

Authority

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

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

Conventions used in this book

This section informs you on the styles, highlighting, and assumptions used throughout the book.

Terminology

In this book, 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, a stand-alone dump utility, or a DCSS. See also “IPL and booting” on page 315.

sysfs and procfs

Throughout the book, the mount point for the virtual Linux file system sysfs is assumed to be /sys. Correspondingly, the mount point for the proc file system is assumed to be /proc.

Documentation directory

This book sometimes refers to files in the Documentation directory in the Linux source tree. On Red Hat Enterprise Linux 6 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>.noarch` RPM.

Hexadecimal numbers

Mainframe books and Linux books 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 book.

Highlighting

This book 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 **bold**.
- Input and output as normally seen on a computer screen is shown

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

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 indicates a variable that you must substitute with specific information. For example:

```
►►── -p <interface>──────────────────────────────────────────────────►◄
```

Here you must code `-p` as shown and supply a value for *<interface>*.
An italicized uppercase word indicates a variable that must appear in uppercase:

```
►►──vmhalt=<COMMAND>──────────────────────────────────────────────────►◄
```

Repetition

An arrow returning to the left means that the item can be repeated.

```
►►──┬──────────┐──────────────────────────────────────────────────────►◄
    ▼ <repeat> │
```

A character within the arrow means you must separate repeated items with that character.

```
►►──┬─,────────┐──────────────────────────────────────────────────────►◄
    ▼ <repeat> │
```

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:

```
►►──┬─A─┬──────────────────────────────────────────────────────────►◄
    ├─B─┤
    └─C─┘
```

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:

```
►►──┬─A─┬──────────────────────────────────────────────────────────►◄
    ├─B─┤
    └─C─┘
```

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:

```
►►──┬─────┬──────────────────────────────────────────────────────────►◄
    ├─A───┤
    ├─B───┤
    └─C───┘
```

Here you may enter either `A` or `B` or `C`, or you may omit the field.

Finding IBM books

The PDF version of this book contains URL links to much of the referenced literature.

For some of the referenced IBM books, links have been omitted to avoid pointing to a particular edition of a book. You can locate the latest versions of the referenced IBM books through the IBM Publications Center at:

`www.ibm.com/shop/publications/order`

Part 1. General concepts

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

Newest version: You can find the newest version of this book at
`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the Red Hat Enterprise Linux 6 release notes at
`http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/`

Chapter 1. How devices are accessed by Linux

User space programs access devices through:

- Device nodes (character and block devices)
- Interfaces (network devices)

Device name, device nodes, and major/minor numbers

The Linux kernel represents the character and block devices it knows as a pair of numbers *<major>*:*<minor>*.

Some major numbers are reserved for particular device drivers, others 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.

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 31). Each device name is associated with a minor number (see Figure 1).

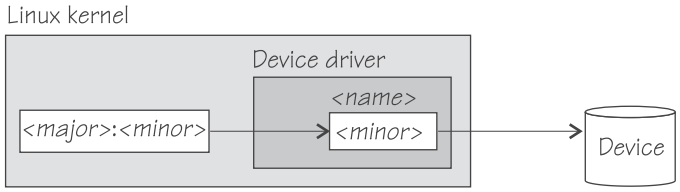


Figure 1. Minor numbers and device names

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).

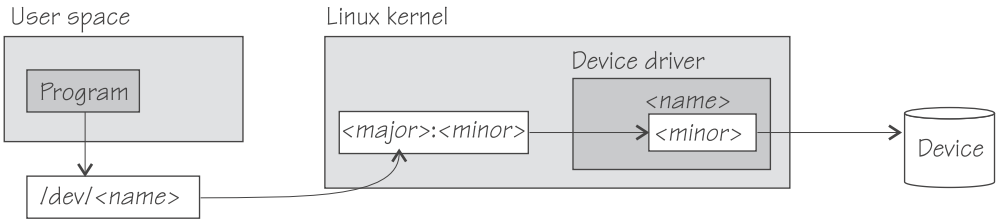


Figure 2. Device nodes

Red Hat Enterprise Linux 6 uses udev to create device nodes for you. There is always a device node that matches the device name 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 (see Figure 3).

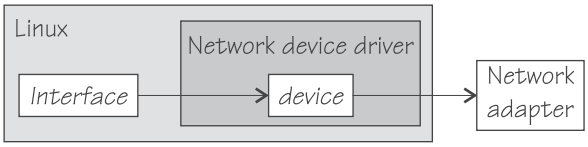


Figure 3. Interfaces

When a network device is defined, it is associated with a real or virtual network adapter. You can configure the adapter properties for a particular network device through the device representation in sysfs (see “Devices and device attributes” on page 8).

You activate or deactivate a connection by addressing the interface with **ifconfig** or an equivalent command. All interfaces that are provided by the network device drivers described in this book are interfaces for the Internet Protocol (IP).

Interface names

The interface names are assigned by the Linux network stack and are of the form *<base_name><n>* where *<base_name>* is a base name used for a particular interface type and *<n>* is an index number that identifies an individual interface of a given type.

Table 1 summarizes the base names used for the network device drivers for interfaces that are associated with real hardware:

Table 1. Interface base names for real devices

Base name	Interface type	Device driver module	Hardware
eth	Ethernet	qeth, lcs	OSA-Express, OSA-Express2, OSA-Express3
osn	ESCON/CDLC bridge	qeth	OSA-Express2, OSA-Express3
ctc	Channel-to-Channel	ctcm	ESCON® channel card, FICON® channel card
mpc	Channel-to-Channel	ctcm	ESCON channel card

Table 2 summarizes the base names used for the network device drivers for interfaces that are associated with virtual hardware:

Table 2. Interface base names for virtual devices

Base name	Interface type	Device driver module	Comment
hsi	HiperSockets™, Guest LAN	qeth	Real HiperSockets or HiperSockets guest LAN
eth	Guest LAN	qeth	QDIO guest LAN
ctc	virtual Channel-to-Channel	ctcm	virtual CTCA
mpc	virtual Channel-to-Channel	ctcm	virtual CTCA

When the first device for a particular interface name is set online, it is assigned the index number 0, the second is assigned 1, the third 2, and so on. For example, the first HiperSockets interface is named hsi0, the second hsi1, the third hsi2, and so on.

When a network device is set offline, it retains its interface name. When a device is removed, it surrenders its interface name and the name can be reassigned as network devices are defined in the future. When an interface is defined, the Linux kernel always assigns the interface name with the lowest free index number for the particular type. For example, if the network device with an associated interface name hsi1 is removed while the devices for hsi0 and hsi2 are retained, the next HiperSockets interface to be defined becomes hsi1.

Matching devices with the corresponding interfaces

If you define multiple interfaces on a Linux instance, you need to keep track of the interface names assigned to your network devices. Red Hat Enterprise Linux 6 uses udev to track the network interface name and preserves the mapping of interface names to network devices across IPLs.

How you can keep track of the mapping yourself differs depending on the network device driver. For qeth, you can use the **lsqeth** command (see “lsqeth - List qeth based network devices” on page 405) to obtain a mapping.

After setting a device online, read `/var/log/messages` or issue **dmesg** 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 9 tells you where you can find the device directories with their attributes in sysfs.

Main steps for setting up a network interface

The following main steps apply to all network device drivers. How to perform a particular step can be different for the different device drivers. The main steps for setting up a network interface are:

- Define a network device.

  The device driver creates directories that represent the device in sysfs.

  Tip: Use the **znetconf** command (except for OSX and OSM devices) to perform this step. See “znetconf - List and configure network devices” on page 455.
- Configure the device through its attributes in sysfs (see “Device views in sysfs” on page 9).

  For some devices, there are attributes that can or need to be set later when the device is online or when the connection is active.
- Set the device online.

  This makes the device known to the Linux network stack and associates the device with an interface name. For devices that are associated with a physical network adapter it also initializes the adapter for the network interface.
- Configure and activate the interface.

  This adds interface properties like IP addresses, MTU, and netmasks to a network interface and makes the network interface available to user space programs.

Chapter 2. Devices in sysfs

Most of the device drivers create structures in sysfs. These structures hold information on individual devices and are also used to configure and control the devices. This section provides an overview of these structures.

Device categories

Figure 4 illustrates a part of sysfs.

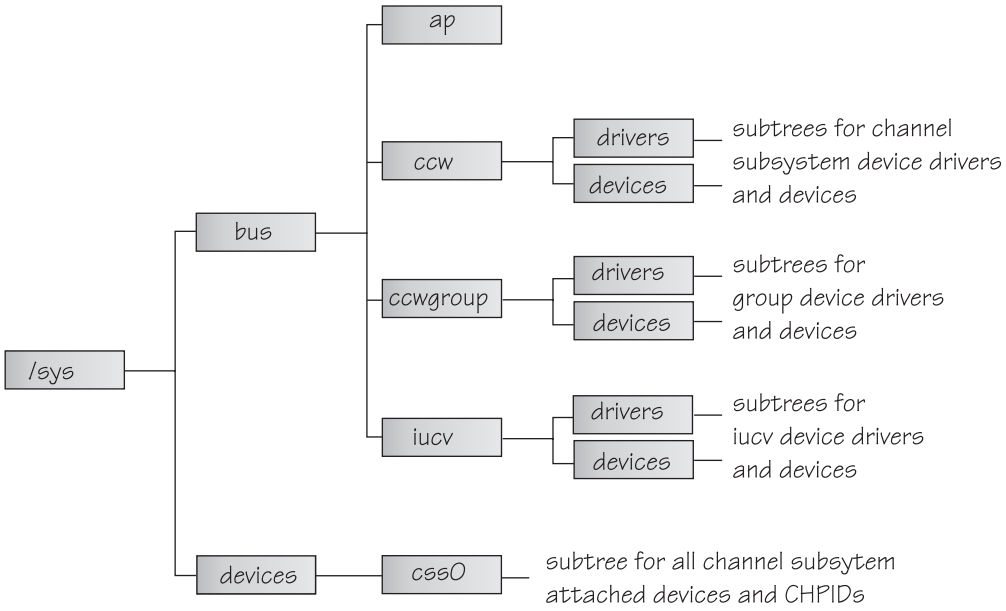


Figure 4. sysfs

`/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. There are several categories of devices. 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.

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.

Note: The NETIUCV device driver is deprecated. For information about IUCV devices, see the first edition of the October 2005 stream version of *Device Drivers, Features, and Commands*, SC33-8289-00.

Table 3 lists the device drivers that have representation in sysfs:

Table 3. 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`
Tape	CCW	`/sys/bus/ccw/drivers/tape_34xx` `/sys/bus/ccw/drivers/tape_3590`
Cryptographic	AP	`/sys/bus/ap/drivers/cex2a` `/sys/bus/ap/drivers/cex2c`
DCSS	n/a	`/sys/devices/dcssblk`
XPRAM	n/a	`/sys/devices/system/xpram`
z/VM recording device driver	IUCV	`/sys/bus/iucv/drivers/vmlogrdr`
OSA-Express, OSA-Express2, OSA-Express3, HiperSockets (qeth)	CCW group	`/sys/bus/ccwgroup/drivers/qeth`
LCS	CCW group	`/sys/bus/ccwgroup/drivers/lcs`
CTCM	CCW group	`/sys/bus/ccwgroup/drivers/ctcm`

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.

Devices and device attributes

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.

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 346 for details.

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

availability
indicates if the device can be used. Possible values are:

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

boxed The device has been locked by another operating system instance and cannot be used until the lock is surrendered or forcibly broken (see “Accessing DASD by force” on page 40).

no device
Applies to disconnected devices only. The device is gone after a machine check and the device driver has requested to keep the (online) device anyway. Changes back to “good” when the device returns after another machine check and the device driver has accepted the device back.

no path
Applies to disconnected devices only. The device has no path left after a machine check or a logical vary off and the device driver has requested to keep the (online) device anyway. Changes back to “good” when the path returns after another machine check or logical vary on and the device driver has accepted 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>
```

“Device views in sysfs” tells you where you can find the device directories with their attributes in sysfs. Red Hat Enterprise Linux 6 uses configuration files to control devices. For example, network devices have interface scripts called `/etc/sysconfig/network-scripts/ifcfg-<interface-name>`. See the *Red Hat Enterprise Linux 6 Deployment Guide* for details about configuration files.

Device views in sysfs

sysfs provides multiple views of device specific data. The most important views are:

- Device driver view
- Device category view
- Device view
- Channel subsystem view

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

```
0.n.dddd
```

where n is the subchannel set ID and dddd is the device ID. For Linux instances that run as z/VM guest operating systems, the subchannel set ID is always 0. Multiple subchannel sets are available on System z9® or later machines.

Device driver view

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 3 on page 8).

<device_bus_id>
identifies an individual device (see “Devices and device attributes” on page 8).

Note: DCSSs and XPRAM 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 CEX2A card):
  ```
  /sys/bus/ap/drivers/cex2a/card3b
  ```

Device category view

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 “Devices and device attributes” on page 8).

Note: DCSSs and XPRAM are not represented in this view.

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

The device 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 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 “Devices and device attributes” on page 8).

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

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

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

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 13

pimpampom
: provides the path installed, path available and path operational masks. Refer to *z/Architecture® Principles of Operation*, SA22-7832 for details on the masks.

Channel path measurement

In sysfs, an attribute is created for 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.

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.

To set a CHPID logically online, set its status attribute to “online” by writing the value “on” to it. To set a CHPID logically offline, set its status attribute to “offline” by writing “off” to it. 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 has been set 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 has been set logically online issue:

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

Configuring a CHPID on LPAR

For Linux on LPAR, directories that represent CHPIDs contain a “configure” attribute that you can use to query and change the configuration state of I/O channel-paths. Supported configuration changes are:

- From standby to configured (“configure”).
- From configured to standby (“deconfigure”).

To configure a CHPID, set its configure attribute by writing the value “1” to it. To deconfigure a CHPID, set its configure attribute by writing “0” to it. 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 364 and “lschp - List channel paths” on page 397.

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
```

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

For information on the device driver modules see `/lib/modules/<kernel_version>/modules.ccwmap`. This file is generated when you install the Linux kernel (version *<kernel_version>*).

Chapter 3. Kernel and module parameters

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 42, “Selected kernel parameters,” on page 459. You can also find descriptions for most of the kernel parameters in `Documentation/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 6 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 6, 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

There are different methods for passing kernel parameters to the Linux kernel.

- 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 6 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

You use the zipl tool to create Linux boot configurations for IBM mainframe systems (see Chapter 33, “Initial program loader for System z - zipl,” on page 289 for details). Which sources of kernel parameters you can use depends on the mode in which you run zipl.

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. You specify details about the boot configuration in the configuration file (see “zipl modes” on page 290).

As shown in Figure 5, there are three sources of kernel parameters for zipl in configuration-file mode.

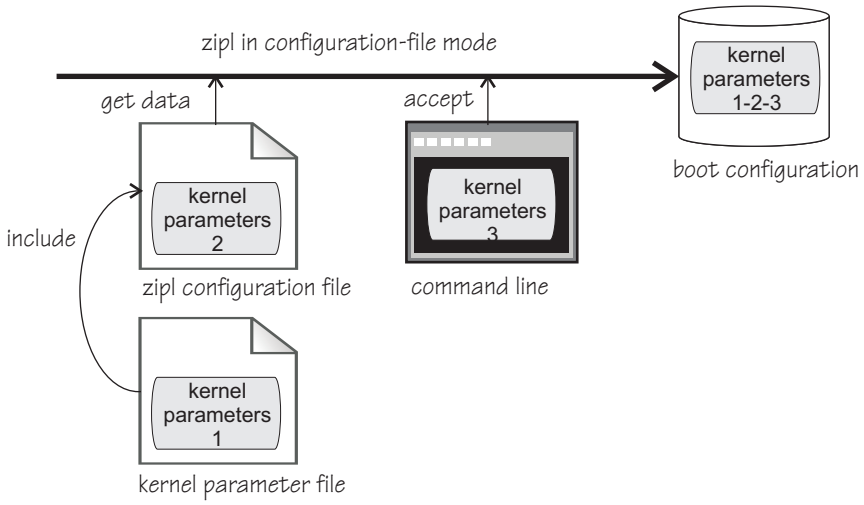


Figure 5. Sources of kernel parameters for zipl in configuration-file mode

In configuration-file mode, zipl concatenates the kernel parameters in the order:

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

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 (see “zipl modes” on page 290).

As shown in Figure 6 on page 19, there are two sources of kernel parameters for zipl in command-line mode.

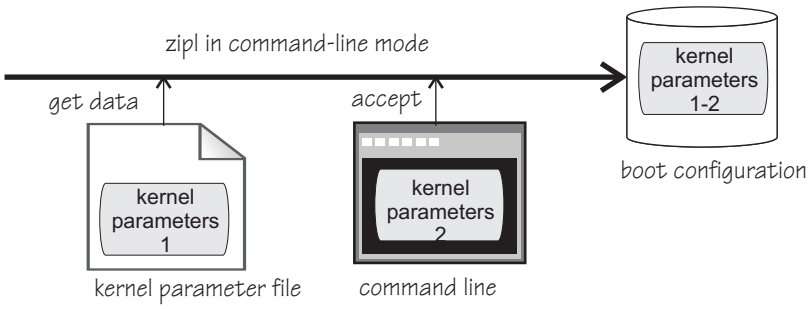


Figure 6. Sources of kernel parameters for zipl in command-line mode

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

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

Conflicting settings and limitations

If the resulting parameter string in the boot configuration contains conflicting settings, the last specification in the string overrides preceding ones.

The kernel parameter file can contain 895 characters of kernel parameters plus an end-of-line character.

In total, the parameter string in the boot configuration is limited to 895 characters. If your specifications exceed this limit, the parameter string in the boot configuration is truncated after the 895th character.

This limitation applies to the parameter string in the boot configuration. You can provide additional parameters when booting Linux. Linux accepts up to 4096 characters of kernel parameters in total. See “Adding kernel parameters to a boot configuration” on page 20.

Using a kernel parameter file

For booting Linux from the z/VM reader, you can directly use a separate kernel parameter file. See “Using the z/VM reader” on page 322 and *Building Linux Systems under IBM VM*, REDP-0120 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 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 319 and “Example for a DASD menu configuration (LPAR)” on page 326 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 a z/VM Linux guest virtual machine” on page 318 that applies to your boot device.

z/VM guest virtual machine with a SCSI boot device
When booting Linux in a z/VM guest virtual machine from a SCSI boot device, you can use the SET LOADDEV command with the SCPDATA option to specify kernel parameters. See “Using a SCSI device” on page 320 for details.

LPAR mode with a SCSI boot device
When booting Linux in LPAR mode from a SCSI boot device, you can specify kernel parameters in the “Operating system specific load parameters” field on the HMC Load panel. See Figure 63 on page 325.

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

Adding kernel parameters to a 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 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 specified with the interactive boot menu
3. Depending on where your are booting Linux:
   - **z/VM:** kernel parameters specified with the PARM parameter for CCW boot devices; kernel parameters specified as SCPDATA for SCSI boot devices
   - **LPAR:** kernel parameters specified on the HMC Load panel for CCW 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.

Examples:

- 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`.
- If the kernel parameters in your boot configuration include `resume=/dev/dasda2` to specify a disk from which to resume the Linux instance when it has been suspended, you can circumvent the resume process by specifying `noresume` when booting.

Replacing all kernel parameters in a boot configuration

Kernel parameters you specify when booting can also 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 (=).

Example:

```
=zfcp.device=0.0.3c3b,0x5005076303048335,0x4050407e00000000 root=/dev/sda1
```

Note: This feature is intended for expert users who want to test a set of parameters. When 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 6 includes in the parameter string for use by the init process.

Read `/proc/cmdline` to find out with which parameters a running Linux instance has been started (see also "Displaying the current kernel parameter line").

Examples for kernel parameters

The following kernel parameters are typically used for booting Red Hat Enterprise Linux 6:

conmode=<*mode*>, **condev=**<*cuu*>, and **console=**<*name*>
to set up the Linux console. See "Console kernel parameter syntax" on page 275 for details.

resume=<*partition*>, **noresume**, **no_console_suspend**
to configure suspend and resume support (see Chapter 35, "Suspending and resuming Linux," on page 333).

See Chapter 42, "Selected kernel parameters," on page 459 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 has been booted.

```
# cat /proc/cmdline
zfcp.device=0.0.3c3b,0x5005076303048335,0x4050407e00000000 root=/dev/sda1
```

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 329 about displaying the parameters that were used to IPL and boot the running Linux instance.

Kernel parameters for rebooting

By default, Linux uses the current kernel parameters for rebooting. See "Rebooting from an alternative source" on page 330 about how to set up Linux to use different kernel parameters for re-IPL and the associated reboot.

Specifying module parameters

You can specify module parameters with modprobe, on the kernel parameter line, or include them 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 6 sets for you.

Module 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 6 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 loading the dasd_mod module.

Including module parameters in a boot configuration

Red Hat Enterprise Linux 6 uses an initial file system (initramfs) when booting. The initramfs does not contain device specifications. Instead it takes module parameters from **dracut** during the boot process. **dracut** obtains the module parameters by parsing the kernel parameter line for parameters with an “rd_” prefix.

Anaconda writes information about devices that need to be accessible during the boot process to zipl.conf for you. This includes the device with the root file system and, if configured, the swap partition that is used to resume a suspended system.

Follow these steps to provide module parameters for modules that are included in an initramfs:

1. With an “rd_” prefix, specify the module parameters in zipl.conf. For example, use `rd_DASD=` instead of `dasd=`.
2. Run **zipl** to include the new parameter line in your boot configuration.

See the dracut man page for more details about parameters with an “rd_” prefix.

Part 2. Storage

This part describes the storage device drivers for Red Hat Enterprise Linux 6 for System z.

Newest version: You can find the newest version of this book at
`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the Red Hat Enterprise Linux 6 release notes at
`http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/`

Chapter 4. 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 a variety of 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 DASD have been largely replaced by emulated DASD, such as 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 attached through a System z FCP adapter are not classified as DASD. They are handled by the zfcp driver (see Chapter 5, “SCSI-over-Fibre Channel device driver,” on page 49).

Features

The DASD device driver supports the following devices and functions:

- The DASD device driver has no dependencies on the adapter that is used to physically connect the DASDs to the System z hardware. You can use any adapter that is supported by the System z hardware.
- The DASD device driver supports ESS virtual ECKD-type disks
- The DASD device driver supports the control unit attached physical devices as summarized in Table 4:

  Table 4. Supported control unit attached DASD

Device format	Control unit type	Device type
ECKD (Extended Count Key Data)	1750	3380 and 3390
	2107	3380 and 3390
	2105	3380 and 3390
	3990	3380 and 3390
	9343	9345
	3880	3390
FBA (Fixed Block Access)	6310	9336
	3880	3370

  All models of the specified control units and device types listed in Table 4 work with the DASD device driver. This includes large devices with more than 65520 cylinders, for example, 3390 Model A. Check the storage support statement for what works with Red Hat Enterprise Linux 6 for System z.
- The DASD device driver is also known to work with these devices:
  - RAMAC
  - RAMAC RVA
- The DASD device driver provides a disk format with up to three partitions per disk. See “System z compatible disk layout” on page 27 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.
- The DASD device driver supports High Performance FICON on storage devices that provide this feature.
- The DASD device driver supports large volumes (devices with more than 65520 cylinders, for example, 3390 Model A), solid state devices, and encrypted devices.

What you should know about DASD

This section describes the available DASD layouts and the naming scheme used for DASD devices.

The IBM label partitioning scheme

The DASD device driver is embedded into the Linux generic support for partitioned disks. This implies that you can have any kind of partition table known to Linux on your DASD.

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 and includes System z IPL, label, and for some layouts VTOC records. Partitioning schemes for platforms other than System z generally do not preserve these mainframe specific records.

Red Hat Enterprise Linux 6 for System z includes the IBM label partitioning scheme that preserves the System z IPL, label, and VTOC records. This partitioning scheme allows Linux to share a disk with other mainframe operating systems. For example, a traditional mainframe operating system could 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:

- “System z compatible disk layout” on page 27
- “Linux disk layout” on page 29
- “CMS disk layout” on page 30

DASD partitions

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. The compatible disk layout allows for up to three partitions on a DASD. The Linux disk layout and the CMS disk layout both permit a single partition only.

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. If you use another partition editor, it is your responsibility to ensure that partitions do not overlap. If they do, data damage will occur.
- Leave no gaps between adjacent partitions to avoid wasting space. Gaps are not reported as errors, and can only be reclaimed by deleting and recreating one or more of the surrounding partitions and rebuilding the file system on them.

A disk need not be partitioned completely. You may begin by creating only one or two partitions at the start of your disk and convert the remaining space to a partition later (perhaps when performance measurements have given you a better value for the block size).

There is no facility for moving, enlarging or reducing partitions, because **fdasd** has no control over the file system on the partition. You only can delete and recreate 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.

System z compatible disk layout

You can only format ECKD-type DASD with the compatible disk layout.

Figure 7 illustrates a DASD with the compatible disk layout.

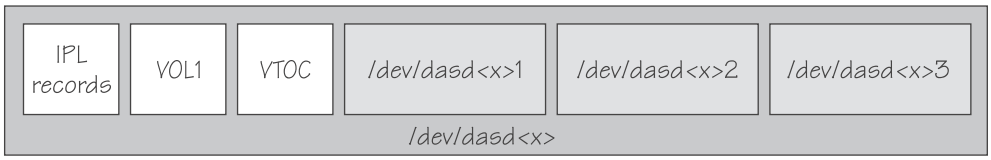


Figure 7. Compatible disk layout

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. Apart from a slight loss in disk capacity this is transparent to the user.

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 31). See “DASD device nodes” on page 32 for alternative addressing possibilities.

Disks with the compatible disk layout can have one to three partitions. Linux can address the partitions as /dev/dasd<*x*>1, /dev/dasd<*x*>2, and /dev/dasd<*x*>3, respectively.

You use the **dasdfmt** command (see “dasdfmt - Format a DASD” on page 375) to format a disk with the compatible disk layout. You use the **fdasd** command (see “fdasd – Partition a DASD” on page 387) to create and modify partitions.

Volume label

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. The contents are:

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 %

  Recommendation: 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 %. You might have to enclose a VOLSER with special characters in apostrophes when specifying 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

Like other System z operating systems, Red Hat Enterprise Linux 6 for System z 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 located in 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 zeroes (format 5).
- For volumes with more than 65534 cylinders, one DSCB (format 7)
- For each partition:
  - On volumes with 65534 or less cylinders, one DSCB (format 1)
  - On volumes with more than 65534 cylinders, one 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 System 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:

```
                            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
```

In Linux, this DASD might appear so:

```
# 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

You can only format ECKD-type DASD with the Linux disk layout. Figure 8 illustrates a disk with the Linux disk layout.

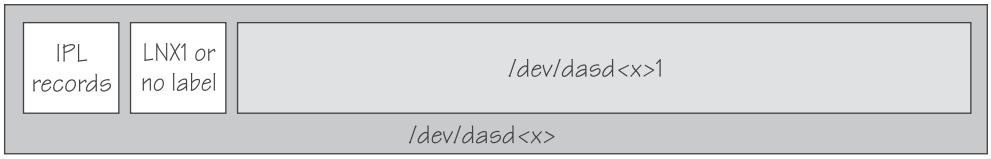


Figure 8. Linux disk layout

DASDs with the Linux disk layout either have an LNX1 label or are not labeled. The IPL records and volume label are not intended for use by Linux applications. Apart from a slight loss in disk capacity this is transparent to the user.

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 31). Linux can access the partition as `/dev/dasd<x>1`.

You use the **dasdfmt** command (see “dasdfmt - Format a DASD” on page 375) to format a disk with the Linux disk layout.

CMS disk layout

The CMS disk layout only applies to Linux as a z/VM guest operating system. The disks are formatted using z/VM tools. Both ECKD- or FBA-type DASD can have the CMS disk layout. Apart from accessing the disks as ECKD or FBA devices, you can also access them using DIAG calls.

Figure 9 illustrates two variants of the CMS disk layout.

Figure 9. CMS disk layout

The variant in the upper part of Figure 9 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 lower part of Figure 9 illustrates a CMS reserved volume. DASD like this have been 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.

Both variants of the CMS disk layout only allow a single Linux partition. The IPL record, volume label and (where applicable) the CMS metadata, are not intended for use by Linux applications. Apart from a slight loss in disk capacity this is transparent to the user.

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 31). Linux can access the partition as `/dev/dasd<x>1`.

“Enabling DIAG calls to access DASDs” on page 41 describes how you can enable DIAG.

Disk layout summary

Table 5 summarizes how the available disk layouts map to device formats, support DIAG calls as an access method, and the maximum number of partitions they support.

Table 5. Disk layout summary

Disk Layout	Device format		DIAG call support (z/VM only)	Maximum number of partitions
	ECKD	FBA		
CDL	Yes	No	No	3
LDL	Yes	No	Yes	1
CMS (z/VM only)	Yes	Yes	Yes	1

DASD naming scheme

The DASD device driver uses the major number 94. For each configured device it uses 4 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 6 shows how the device names map to the available minor numbers.

Table 6. 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 following 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 6 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.

To help you identify a particular disk, udev creates additional devices nodes that are based on the disk's bus ID, the disk label (VOLSER), and information about the file system on the disk. The file system information can be a universally unique identifier (UUID) and, if available, the file system label.

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 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.

When using device nodes based on VOLSER, be sure that the VOLSERs in your environment are unique (see “Volume label” on page 27).

If you assign the same VOLSER to multiple devices, Linux can access all of them through the device nodes that are based on the respective device names. However, only one of them can be accessed through the VOLSER-based device node. This makes the node ambiguous and should be avoided.

Furthermore, if the VOLSER on the device that is addressed by the node is changed, the previously hidden device is not automatically addressed instead. This requires a reboot or the Linux kernel needs to be forced 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 has been assigned, 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.

When using device nodes 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: When using 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 creating a backup disk. 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

Instead of using the standard device nodes, you can use udev-created device nodes to be sure that you access a particular physical disk space, regardless of the device name that is assigned to it.

The example in this section uses device nodes that are based on the bus-ID. You can adapt this example to the other device nodes described in “DASD device nodes” on page 32. The device nodes you can use depend on your setup.

Example

The examples in this section assume that udev provides device nodes as described in “DASD device nodes” on page 32. To assure that you are addressing a device with bus-ID 0.0.b100 you could make substitutions like the following.

Instead of issuing:

```
# fdasd /dev/dasdzzz
```

issue:

```
# fdasd /dev/disk/by-path/ccw-0.0.b100
```

In the file system information in `/etc/fstab` you could replace the following specifications:

```
/dev/dasdzzz1 /temp1 ext2 defaults 0 0
/dev/dasdzzz2 /temp2 ext2 defaults 0 0
```

with these specifications:

```
/dev/disk/by-path/ccw-0.0.b100-part1 /temp1 ext2 defaults 0 0
/dev/disk/by-path/ccw-0.0.b100-part2 /temp2 ext2 defaults 0 0
```

Setting up the DASD device driver

This section describes how to load and configure the DASD device driver modules.

DASD module parameter syntax

```
►►─modprobe──┬─ dasd_mod──┬──────────────────────────────────┬──┬─────────────────────────────┬──────────►◄
             │            │                                  │  │┌─ eer_pages=5 ──────────────┐│
             │            │                                  │  │└─ eer_pages=<pages>─────────┘│
             │            │          ┌─,──────────────────┐  │  │                             │
             │            └─ dasd=──┴┬─ device-spec ─────┬┴──┘  │                             │
             │                       ├─autodetect────────┤      │                             │
             │                       ├─probeonly─────────┤      │                             │
             │                       ├─nopav─────────────┤      │                             │
             │                       └─nofcx─────────────┘      │                             │
             ├─ dasd_eckd_mod──────────────────────────────────┴─────────────────────────────┤
             ├─ dasd_fba_mod───────────────────────────────────────────────────────────────┤
             └─ dasd_diag_mod──────────────────────────────────────────────────────────────┘
```

device-spec:

```
├──┬─<device_bus_id>──────────────────────────┬──┬──────────────────────────────────────────┤
   └─<from_device_bus_id>-<to_device_bus_id>─┘  │  │┌─:──────────────────────────┐│
                                                └─(┴┴─ ro──────────────────────┴┴─)─┘
                                                      ├─diag──────────────────────┤
                                                      ├─erplog────────────────────┤
                                                      └─failfast──────────────────┘
```

Where:

dasd_mod
loads the device driver base module.

When loading the base module you can specify the `dasd=` parameter.

You can use the `eer_pages` parameter to determine the number of pages 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 31 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 you are running as a z/VM guest operating system 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 or not 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 on Linux instances that run as z/VM guest operating systems.

nofcx suppresses accessing the storage server using 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.

(ro)
: specifies that the given device or range is to be accessed in read-only mode.

(diag)
: forces the device driver to access the device (range) using 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)
: returns "failed" for an I/O operation when the last path to a DASD is lost. Use this option with caution (see "Switching immediate failure of I/O requests on or off" on page 44).

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 module parameters in a boot configuration" on page 22.

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 7 shows the resulting allocation of device names:

Table 7. 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 should be suppressed for all DASD:

```
modprobe dasd_mod dasd=nofcx,4711-4713
```

Working with the DASD device driver

This section describes typical tasks that you need to perform when working with DASD devices.

- “Preparing an ECKD-type DASD for use” on page 38
- “Preparing an FBA-type DASD for use” on page 40
- “Accessing DASD by force” on page 40
- “Enabling DIAG calls to access DASDs” on page 41
- “Working with extended error reporting for ECKD” on page 42
- “Switching extended error reporting on and off” on page 43
- “Setting a DASD online or offline” on page 43
- “Enable and disable logging” on page 44

- “Switching immediate failure of I/O requests on or off” on page 44
- “Displaying DASD information” on page 45

Most of these tasks involve writing to and reading from device attributes 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 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 in *Red Hat Enterprise Linux 6 Installation Guide*. For a general discussion of configuration files, see *Red Hat Enterprise Linux 6 Deployment Guide*.

Preparing an ECKD-type DASD for use

This section describes the main steps for enabling an ECKD-type DASD for use by Red Hat Enterprise Linux 6 for System z.

Before you can use an ECKD-type DASD you must format it with a suitable disk layout. If you format the DASD with the compatible disk layout, you need to create one, two, or three partitions. You can then use your partitions as swap areas or to create a Linux file system.

Before you start:

- 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 need to know the device node through which the DASD can be addressed.

Perform these steps to prepare the DASD:

1. Format the device with the **dasdfmt** command (see “dasdfmt - Format a DASD” on page 375 for details). The formatting process can take hours for large DASD.

   Recommendations:

   - Use the default `-d cdl` option. This option formats the DASD with the IBM compatible disk layout that permits you to create partitions on the disk.
   - Use the `-p` option to display a progress bar.

   Example:

   ```
   dasdfmt -b 4096 -d cdl -p  /dev/dasdzzz
   ```

2. Proceed according to your chosen disk layout:

   - If you have formatted your DASD with the Linux disk layout, skip this step and continue with step 3 on page 39. 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 387 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 28).

3. 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 of your choice. 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 lower than that used for formatting the disk with the **dasdfmt** command.

   Recommendation: Use the same block size for the file system that has been used for formatting.

   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).

4. 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
   ```

   If a block device supports barrier requests, journaling file systems like ext3 or reiserfs can make use of this feature to achieve better performance and data integrity. Barrier requests are supported for the DASD device driver and apply to ECKD, FBA, and the DIAG discipline.

   Write barriers are used by file systems and are enabled as a file-system specific option. For example, barrier support can be enabled for an ext3 file system by mounting it with the option `-o barrier=1`:

   ```
   mount -o barrier=1 /dev/dasdzzz1 /mnt
   ```

Preparing an FBA-type DASD for use

This section describes the main steps for enabling an FBA-type DASD for use by Red Hat Enterprise Linux 6 for System z.

Note: To access FBA devices, use the DIAG access method (see "Enabling DIAG calls to access DASDs" on page 41 for more information).

Before you start:

- 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 need to know the device bus-ID or the device node through which the DASD can be addressed.

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 of your choice. For example, use the Linux **mkfs.ext4** command to create an ext4 file system (see the man page for details).

   Example:

   ```
   mkfs.ext4 -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.

   Example: To mount a file system in a partition `/dev/dasdzzy1` issue:

   ```
   # mount /dev/dasdzzy1 /mnt
   ```

Accessing DASD by force

When a Linux instance boots in a mainframe environment, it can encounter DASD 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.

To check if 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 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 9).

Issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/availabilty
```

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/availabilty
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.

CAUTION:
Breaking an external lock can have unpredictable effects on the system that holds the lock.

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
```

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.

Example: To force a DASD with device number 0xb110 online issue:

```
# echo force > /sys/bus/ccw/devices/0.0.b110/online
```

For information on how to break the look of a DASD that has already been analyzed see “tunedasd - Adjust DASD performance” on page 443.

Enabling DIAG calls to access DASDs

Before you start: This section only applies to Linux instances and DASD for which all of the following are true:

- The Linux instance runs as a z/VM guest operating system.
- 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.

You can use DIAG calls to access both ECKD- and FBA-type DASD. You use the device's use_diag sysfs attribute to enable or switch off DIAG calls in a system that is online. Set the use_diag attribute to “1” to enable DIAG calls. Set the use_diag attribute to “0” to switch off DIAG calls (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) using the DIAG access method.

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 DIAG calls are not available and you set the use_diag attribute to “1”, you will not be able to set the device online (see “Setting a DASD online or offline” on page 43).

Note: When switching between enabled and disabled DIAG calls on FBA-type DASD, first re-initialize 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*, SC24-6179.

Example

In this example, DIAG calls are enabled for a DASD with device number 0xb100.

Note: You can only use the use_diag attribute when the device is offline.

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
```

Working with extended error reporting for ECKD

You can perform the following file operations on the device node:

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 as well as non-blocking read is 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 in conjunction with non-blocking read.

Switching extended error reporting on and off

Extended error reporting is turned off by default. To turn extended error reporting on, 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 will generate records and may have further side effects. The records are made available via a character device interface.

To switch off extended error reporting issue a command of this form:

```
# echo 0 > /sys/bus/ccw/devices/<device_bus_id>/eer_enabled
```

Setting a DASD online or offline

When Linux boots, it senses your DASD. Depending on your specification for the “dasd=” parameter, it automatically sets devices online.

Use the **chccwdev** command (“chccwdev - Set a CCW device online” on page 362) to set a DASD online or offline. Alternatively, you can write “1” to the device's online attribute to set it online or “0” to set it offline.

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:

  ```
  # 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:

  ```
  # chccwdev -d 0.0.b100
  ```

  or

  ```
  # echo 0 > /sys/bus/ccw/devices/0.0.b100/online
  ```

Dynamic attach and detach

You can dynamically attach devices to a running Red Hat Enterprise Linux 6 for System 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 15).

Note

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.

Enable and disable logging

You can enable and disable error recovery processing (ERP) logging on a running system. There are two methods for doing this:

- Enable logging during module load using the `dasd=` parameter.

  For example, to define a device range (0.0.7000-0.0.7005) and switch on logging, change the parameter line to contain:

  ```
  dasd=0.0.7000-0.0.7005(erplog)
  ```

- Use the sysfs attribute `erplog` to switch ERP-related logging on or off.

  Logging can be enabled for a specific device by writing "1" to the `erplog` attribute, for example:

  ```
  echo 1 > /sys/bus/ccw/devices/<device_bus_id>/erplog
  ```

  To disable logging, write "0" to the `erplog` attribute, for example:

  ```
  echo 0 > /sys/bus/ccw/devices/<device_bus_id>/erplog
  ```

Switching immediate failure of I/O requests on or off

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.

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.

- You can switch on immediate failure of I/O requests when you load the base module of the DASD device driver:

  For 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 switch immediate failure of I/O requests on or off.

To switch on immediate failure of I/O requests, write "1" to the `failfast` attribute, for example:

```
echo 1 > /sys/bus/ccw/devices/<device_bus_id>/failfast
```

To switch off immediate failure of I/O requests, write "0" to the `failfast` attribute, for example:

```
echo 0 > /sys/bus/ccw/devices/<device_bus_id>/failfast
```

Displaying DASD information

Use **lsdasd** to display DASD information (see “lsdasd - List DASD devices” on page 401.)

Alternatively, you can use sysfs. Each DASD 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. This sysfs directory contains a number of attributes with information on the DASD.

Table 8. DASD device attributes

Attribute	Explanation
alias	“0” if the DASD is a parallel access volume (PAV) base device or “1” if the DASD is an alias device. For an example of how to use PAV see *How to Improve Performance with PAV*, SC33-8414 on developerWorks at `www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html` This attribute is read-only.
discipline	Is the base discipline, ECKD or FBA, that is used to access the DASD. This attribute is read-only. If DIAG is enabled, this attribute might read DIAG instead of the base discipline.
eer_enabled	“1” if the DASD is enabled for extended error reporting or “0” if it is not enabled (see “Switching extended error reporting on and off” on page 43).
failfast	“1” if I/O operations are returned as failed immediately when the last path to the DASD is lost. “0” if I/O operations to the DASD are blocked until a path reappears. (see “Switching immediate failure of I/O requests on or off” on page 44).
online	“1” if the DASD is online or “0” if it is offline (see “Setting a DASD online or offline” on page 43).
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 imposed by the device itself. This attribute is ignored for PAV alias devices.

Table 8. DASD device attributes (continued)

Attribute	Explanation
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 making changes to the disk layout, for example, when formatting the device. **unformatted** The disk layout detection has 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 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 if the Linux instance runs as a z/VM guest operating system and if the z/VM version and service level supports this identifier. This attribute is read-only.
use_diag	"1" if DIAG calls are enabled "0" if DIAG calls are not enabled (see "Enabling DIAG calls to access DASDs" on page 41). Do not enable DIAG calls for PAV alias devices.
vendor	A specification that identifies the manufacturer of the storage system that contains the DASD. This attribute is read-only.

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 8 on page 45.

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/online
1
# cat /sys/bus/ccw/devices/0.0.b100/readonly
1
# 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 5. SCSI-over-Fibre Channel device driver

This chapter describes the SCSI-over-Fibre Channel device driver (zfcp device driver) for the QDIO-based System z SCSI-over-Fibre Channel adapter. The zfcp device driver provides support for Fibre Channel-attached SCSI devices on System z.

Throughout this chapter, the term *FCP channel* refers to a single virtual instance of a QDIO-based System z SCSI-over-Fibre Channel adapter.

Features

The zfcp device driver supports the following devices and functions:

- You can use most SAN-attached SCSI device types, for example, SCSI disks, tapes, CD-ROMs, and DVDs.
- SAN access through the following FCP adapters:
  - FICON Express®
  - FICON Express2
  - FICON Express4
  - FICON Express8 (as of System z10™)
- The zfcp device driver supports switched fabric and point-to-point topologies.

What you should know about zfcp

The zfcp device driver is a low-level or host-bus adapter driver that supplements the Linux SCSI stack. Figure 10 illustrates how the device drivers work together.

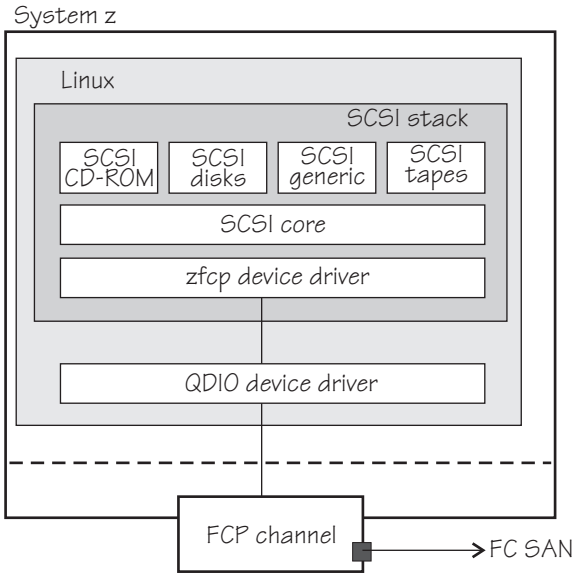


Figure 10. Device drivers supporting the FCP environment

sysfs structures for FCP channels and SCSI devices

FCP channels are CCW devices.

When Linux is booted, it senses the available FCP channels 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 the FCP channel. You use the attributes in this directory to work with the FCP channel.

Example: `/sys/bus/ccw/drivers/zfcp/0.0.3d0c`

The zfcp device driver automatically attaches remote storage ports to the adapter configuration when the adapter is activated and when remote storage ports are added. Each attached remote 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`

You can further extend this structure by adding logical units (usually SCSI devices) to the ports (see “Configuring SCSI devices” on page 65). For each unit you add you get a directory of the form:

`/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<fcp_lun>`

where *<fcp_lun>* is the logical unit number (LUN) of the SCSI device. You use the attributes in this directory to work with an individual SCSI device.

Example: `/sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000`

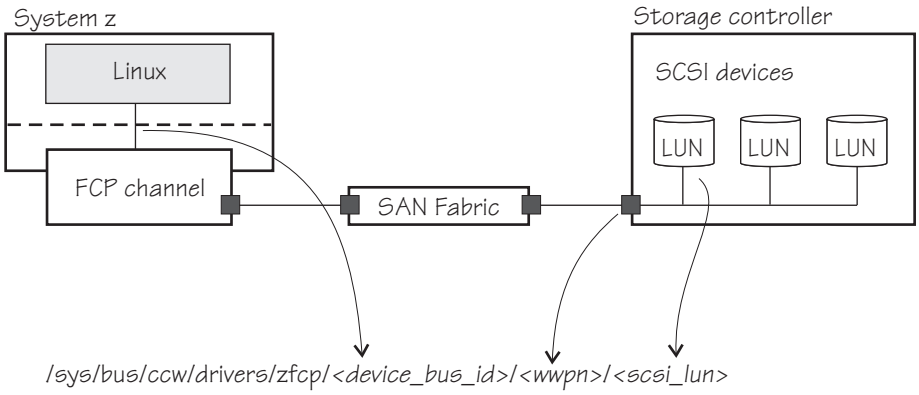


Figure 11. SCSI device in sysfs

Figure 11 illustrates how the path to the sysfs representation of a SCSI device is derived from properties of various components in an IBM mainframe FCP environment.

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 adapters, ports, units, and their associated SCSI stack objects, a command called **lszfcp** is provided with s390-tools. See “lszfcp - List zfcp devices” on page 415 for more details about the command.

See also “Mapping the representations of a SCSI device in sysfs” on page 66.

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 6 provides udev to create device nodes for you that allow you to identify the corresponding actual device.

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 examples in this chapter 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. Refer to `Documentation/devices.txt` in the Linux source tree for more information on the SCSI device naming scheme.

To help you identify a particular device, udev creates additional 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 based on bus IDs
udev creates device nodes of the form

```
/dev/disk/by-path/ccw-<device_bus_id>-zfcp-<wwpn>:<lun>
```

for whole SCSI device and

```
/dev/disk/by-path/ccw-<device_bus_id>-zfcp-<wwpn>:<lun>-part<n>
```

for the *<n>*th partition, where WWPN is the world wide port number of the target port and LUN is the logical unit number representing the target SCSI device.

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 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 respectively, 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-zfcp-0x500507630300c562:0x401040ea00000000`
- `/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-label/boot`
- `/dev/disk/by-path/ccw-0.0.3c1b-zfcp-0x500507630300c562:0x401040ea00000000-part1`
- `/dev/disk/by-id/scsi-36005076303ffc56200000000000010ea-part1`
- `/dev/disk/by-uuid/7eaf9c95-55ac-4e5e-8f18-065b313e63ca`
- `/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-label/SWAP-sda2`
- `/dev/disk/by-path/ccw-0.0.3c1b-zfcp-0x500507630300c562:0x401040ea00000000-part2`
- `/dev/disk/by-id/scsi-36005076303ffc56200000000000010ea-part2`
- `/dev/disk/by-uuid/b4a818c8-747c-40a2-bfa2-acaa3ef70ead`
- `/dev/disk/by-id/wwn-0x6005076303ffc56200000000000010ea-part2`

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. 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 SAN management clients that run on System z.

As shown in Figure 12 on page 53, the zfcp HBA API support includes a user space library.

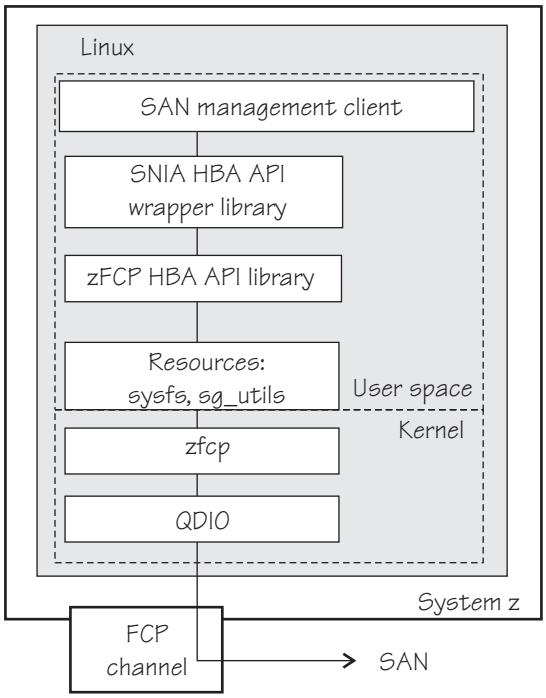


Figure 12. zfcp HBA API support modules

The SNIA (Storage Networking Industry Association) library can interface with the zFCP HBA API. The SNIA library is not part Red Hat Enterprise Linux 6. It is available as hbaapi_src_<*x.x*>.tgz, and can be found at

`hbaapi.sourceforge.net`

The SNIA HBA API library offers a common entry point for applications that manage HBAs. Using the library, an application can talk to any HBA independently of vendor.

The default method in Red Hat Enterprise Linux 6 is for applications to use the zFCP HBA API library directly.

For information on setting up the HBA API support, see “Installing the zfcp HBA API library” on page 55.

FCP LUN access control

As of IBM System z10

FCP LUN access control is not supported.

Access to devices can be restricted by access control software on the FCP channel. For more information on FCP LUN Access Control, visit The IBM Resource Link® website at:

`https://www.ibm.com/servers/resourcelink/`

The Resource Link page requires registration. If you are not a registered user of Resource Link, you will need to register and then log in. On the left navigation bar, click **Tools**, then in the Servers column on the ACT page, click the link **Configuration Utility for FCP LUN Access Control**.

N_Port ID Virtualization for FCP channels

N_Port ID Virtualization (NPIV) allows a single FCP port to appear as multiple, distinct ports that provide separate port identification. NPIV support can be configured on the SE per CHPID and LPAR for an FCP adapter. The zfcp device driver supports NPIV error messages and adapter attributes. See “Displaying adapter information” on page 57 for the adapter attributes.

For more details, refer to the connectivity page at

`www.ibm.com/systems/z/connectivity/fcp.html`

N_Port ID Virtualization is available as of IBM System z9.

Further information

FC/FCP/SCSI-3 specifications
Describes SCSI-3, the Fibre Channel Protocol, and fiber channel related information.

`www.t10.org` and `www.t11.org`

Getting Started with zSeries® Fibre Channel Protocol
Introduces the concepts of Fibre Channel Protocol support, and shows how various SCSI devices can be configured to build an IBM mainframe FCP environment. The information is written for Linux 2.4, but much of it is of a general nature and also applies to Linux 2.6:

`www.ibm.com/redbooks/redpapers/pdfs/redp0205.pdf`

Fibre Channel Protocol for Linux and z/VM on IBM System z
Includes an explanation of how FCP is configured using Red Hat Enterprise Linux 5.

`www.ibm.com/redbooks/pdfs/sg247266.pdf`

Linux for IBM System z9 and IBM zSeries
Includes a chapter about FCP-attached SCSI disks.

`www.ibm.com/redbooks/abstracts/sg246694.html?Open`

Supported FCP connectivity options
Lists supported SCSI devices and provides links to further documentation on FCP and SCSI.

`www.ibm.com/systems/z/connectivity/`

How to use FC-attached SCSI devices with Linux on System z, SC33-8413
See www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Setting up the zfcp device driver

This section provides information on how you can specify a SCSI boot device.

zfcp module parameters

This section describes how to load and configure the zfcp device driver.

zfcp module parameter syntax

```
►►─modprobe─zfcp──device=<device_bus_id>,<wwpn>,<fcp_lun>──────────►
                 ┌─ dbfsize=4──────┐   ┌─ queue_depth=32──────┐
►─────────────────┼─────────────────┼───┼──────────────────────┼──────────►◄
                 └─ dbfsize=<pages>─┘   └─ queue_depth=<depth>─┘
```

where:

<device_bus_id>
: specifies the device bus-ID of the FCP channel through which the SCSI device is attached.

<wwpn>
: specifies the target port through which the SCSI device is accessed.

<fcp_lun>
: specifies the LUN of the SCSI device.

<pages>
: specifies the number of pages which should be used for the debug feature.

  The debug feature is available for each adapter and the following areas:

  hba Host bus adapter
  san Storage Area Network
  rec Error Recovery Process
  scsi SCSI

  The value given is used for all areas. The default is 4, that is, four pages are used for each area and adapter. In the following example the dbsfsize is increased to 6 pages:

  ```
  zfcp.dbfsize=6
  ```

  This results in six pages being used for each area and adapter.

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 will be used as the default queue depth for new SCSI devices. You can change the queue depth for each SCSI device using the queue_depth sysfs attribute, see “Setting the queue depth” on page 69.

Installing the zfcp HBA API library

Before you begin: To use the HBA API support you need the following packages:

- The zfcp HBA API library RPM, libzfcphbaapi0.
- Optionally, the SNIA library, hbaapi_src_*<x.x>*.tgz

You can install the libzfcphbaapi0 RPM using **yum**.

Red Hat Enterprise Linux 6 does not provide the SNIA library. If you want to run applications compiled against it or if you want to compile applications against it, you need to download and install it yourself.

The SNIA library expects a configuration file called `/etc/hba.conf` that contains the path to the vendor-specific library libzfcphbaapi.so. A client application needs to issue the **HBA_LoadLibrary()** call as the first call to load the vendor-specific library. The vendor-specific library, in turn, supplies the function **HBA_RegisterLibrary** that returns all function pointers to the common library and thus makes them available to the application.

Working with the zfcp device driver

This section describes typical tasks that you need to perform when working with FCP channels, target ports, and SCSI devices. Set an FCP channel online before you attempt to perform any other tasks.

- Working with FCP channels
  - “Setting an FCP channel online or offline”
  - “Displaying adapter information” on page 57
  - “Recovering a failed FCP channel” on page 60
  - “Starting and stopping collection of QDIO performance statistics” on page 61
  - “Finding out if NPIV is in use” on page 61
- Working with target ports
  - “Scanning for ports” on page 62
  - “Displaying port information” on page 63
  - “Recovering a failed port” on page 64
  - “Removing ports” on page 64
- Working with SCSI devices
  - “Configuring SCSI devices” on page 65
  - “Mapping the representations of a SCSI device in sysfs” on page 66
  - “Displaying information about SCSI devices” on page 67
  - “Setting the queue depth” on page 69
  - “Recovering a failed SCSI device” on page 70
  - “Updating the information about SCSI devices” on page 71
  - “Setting the SCSI command timeout” on page 71
  - “Controlling the SCSI device state” on page 72
  - “Removing SCSI devices” on page 72

Most of these tasks involve writing to and reading from device attributes 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 configuration file /etc/zipl.conf for FCP devices that are part of the root file system and /etc/zfcp.conf for data disks. An example of how to define an FCP device persistently is in *Red Hat Enterprise Linux 6 Installation Guide*. For a general discussion of configuration files, see *Red Hat Enterprise Linux 6 Deployment Guide*.

For debugging, traces are available. For information about traces and how to use them, see the chapter on debugging using zfcp traces in *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413.

Setting an FCP channel online or offline

By default, FCP channels are offline. Set an FCP channel online before you perform any other tasks.

Use the **chccwdev** command (“chccwdev - Set a CCW device online” on page 362) to set an FCP channel online or offline. Alternatively, you can write “1” to an FCP channel's online attribute to set it online, or “0” to set it offline.

Setting an FCP channel 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 channel online was successful you can use a script that first sets the FCP channel device online and after this operation completes checks if the WWPN of a remote storage port has appeared in the sysfs.

When you set an FCP channel offline, the port and LUN subdirectories are preserved. Setting an FCP channel offline in sysfs interrupts the communication between Linux and the FCP channel hardware. After a timeout has expired, the port and LUN attributes indicate that the ports and LUNs are no longer accessible. The transition of the CCW 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 channel 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 channel was set offline.

Examples

- To set an FCP channel with device bus-ID 0.0.3d0c online issue:

```
# chccwdev -e 0.0.3d0c
```

or

```
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/online
```

- To set an FCP channel with device bus-ID 0.0.3d0c offline issue:

```
# chccwdev -d 0.0.3d0c
```

or

```
# echo 0 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/online
```

Displaying adapter information

Before you start: The FCP channel must be online for the adapter information to be valid.

For each online FCP channel, there is a number of read-only attributes in sysfs that provide information on the corresponding adapter card. Table 9 summarizes the relevant attributes.

Table 9. Attributes with adapter information

Attribute	Explanation
hardware_version	Hardware version

Table 9. Attributes with adapter information (continued)

Attribute	Explanation
card_version	Adapter version
lic_version	Hardware microcode level
in_recovery	Shows if adapter is in recovery (0 or 1)
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

For the attributes availability, cmb_enable, and cutype, see “Devices and device attributes” on page 8. The status attribute is reserved.

Table 10. Relevant transport class attributes, fc_host attributes

Attribute	Explanation
maxframe_size	Maximum frame size of adapter
node_name	Worldwide node name (WWNN) of adapter
permanent_port_name	WWPN associated with the physical port of the FCP channel
port_id	Destination ID of the adapter port.
port_name	WWPN. If N_Port ID Virtualization is not available, this shows the same value as permanent_port_name.
port_type	Port type indicating topology of port.
serial_number	Serial number of adapter.
speed	Speed of FC link.
supported_classes	Supported FC service class.
supported_speeds	Supported speeds.
tgid_bind_type	Target binding type.

Table 11. 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 received in error.
dumped_frames	Number of frames lost due to 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.

Table 11. Relevant transport class attributes, fc_host statistics (continued)

Attribute	Explanation
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.

Issue a command of this form to read an attribute:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<attribute>
```

where:

<device_bus_id>
: is the device bus-ID that corresponds to the FCP channel.

<attribute>
: is one of the attributes in Table 9 on page 57.

To read attributes of the associated fc_host use:

```
# cat /sys/class/fc_host/<host_name>/<attribute>
```

where:

<host_name>
: is the ID of the host.

<attribute>
: is one of the attributes in Table 10 on page 58.

Examples

- In this example, information is displayed on an adapter card for an FCP channel that corresponds to a device 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
0x00009111
```

- Alternatively you can use **lszfcp** (see “lszfcp - List zfcp devices” on page 415) to display all attributes of an adapter:

```
# lszfcp -b 0.0.3d0c -a
0.0.3d0c host0
Bus = "ccw"
    availability        = "good"
    card_version        = "0x0003"
    cmb_enable          = "0"
    cutype              = "1731/03"
    devtype             = "1732/03"
    failed              = "0"
    hardware_version    = "0x00000000"
    in_recovery         = "0"
    lic_version         = "0x00000600"
    modalias            = "ccw:t1731m03dt1732dm03"
    online              = "1"
    peer_d_id           = "0x000000"
    peer_wwnn           = "0x0000000000000000"
    peer_wwpn           = "0x0000000000000000"
    status              = "0x5400082e"
Class = "fc_host"
    maxframe_size       = "2112 bytes"
    node_name           = "0x5005076400cd6aad"
    permanent_port_name = "0x5005076401c08f98"
    port_id             = "0x650f13"
    port_name           = "0x5005076401c08f98"
    port_type           = "NPort (fabric via point-to-point)"
    serial_number       = "IBM020000000D6AAD"
    speed               = "2 Gbit"
    supported_classes   = "Class 2, Class 3"
    supported_speeds    = "1 Gbit, 2 Gbit"
    tgtid_bind_type     = "wwpn (World Wide Port Name)"
Class = "scsi_host"
    cmd_per_lun         = "1"
    host_busy           = "0"
    proc_name           = "zfcp"
    sg_tablesize        = "538"
    state               = "running"
    unchecked_isa_dma   = "0"
    unique_id           = "0"
```

Recovering a failed FCP channel

Before you start: The FCP channel must be online.

Failed FCP channels are automatically recovered by the zfcp device driver. You can read the in_recovery attribute to check if recovery is under way. 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 channel, recovery might have failed or the device driver might have failed to detect that the FCP channel is malfunctioning.

To find out if recovery has 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 has failed and “0” otherwise.

You can start or restart the recovery process for the FCP channel 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 channel 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 channel:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/in_recovery
0
# echo 0 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/failed
```

Starting and stopping collection of QDIO performance statistics

QDIO serves as base support for the qeth device driver (QETH subchannel triplets are CCW devices) and for the zfcp device driver (FCP channels are CCW devices) that supports SCSI devices.

For QDIO performance statistics in general there is a device group attribute called `/sys/bus/ccw/qdio_performance_stats`.

This attribute is initially set to 0, that is, QDIO performance data is not collected. To start collection for QDIO, write 1 to the attribute, for example:

```
echo 1 > /sys/bus/ccw/qdio_performance_stats
```

To stop collection write 0 to the attribute, for example:

```
echo 0 > /sys/bus/ccw/qdio_performance_stats
```

Stopping QDIO performance data collection resets the current statistic values to zero.

To display QDIO performance statistics issue:

```
cat /proc/qdio_perf
```

Finding out if NPIV is in use

If the adapter attributes permanent_port_name and port_name are not NULL and are different from each other, the subchannel is operating in NPIV mode.

Example

You can examine whether the adapter attributes port_name and permanent_port_name are the same:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.1940/host0/fc_host/host0/port_name
0xc05076ffef805388

# cat /sys/bus/ccw/drivers/zfcp/0.0.1940/host0/fc_host/host0/permanent_port_name
0x50050764016219a0
```

Alternatively you can use **lszfcp** (see “lszfcp - List zfcp devices” on page 415) to display the above attributes:

```
# lszfcp -b 0.0.1940 -a
0.0.3d0c host0
Bus = "ccw"
    availability        = "good"
    ...
Class = "fc_host"
    maxframe_size       = "2112 bytes"
    node_name           = "0x5005076400c1ebae"
    permanent_port_name = "0x50050764016219a0"
    port_id             = "0x65ee01"
    port_name           = "0xc05076ffef805388"
    port_state = "Online"
    port_type           = "NPort (fabric via point-to-point)"
    serial_number       = "IBM0200000001EBAE"
    ...
```

The example shows that permanent_port_name is different from the port_name, and the subchannel operates in NPIV mode.

Scanning for ports

Before you start: The FCP channel must be online.

The zfcp device driver automatically attaches remote storage ports to the adapter configuration at adapter activation as well as when remote storage ports are added. 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 are attached.

Issue a command of this form:

```
# echo 1 > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/port_rescan
```

where:

<device_bus_id>
: is the device bus-ID that corresponds to the FCP channel.

List the contents of `/sys/bus/ccw/drivers/zfcp/<device_bus_id>` to find out which ports are currently configured for the FCP channel.

Example

In this example, a port with WWPN 0x500507630303c562 has already been configured for an FCP Channel with device 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
```

Displaying port information

For each target port, there is a number of read-only attributes in sysfs that provide port information. Table 12 summarizes the relevant attributes.

Table 12. Attributes with port information

Attribute	Explanation
access_denied	Flag that indicates if the port access is restricted by access control software on the FCP channel (see “FCP LUN access control” on page 53). The value is “1” if access is denied and “0” if access is permitted.
in_recovery	Shows if port is in recovery (0 or 1)

Table 13. Transport class attributes with port information

Attribute	Explanation
node_name	WWNN of the remote 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.

Issue a command of this form to read an attribute:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<attribute>
```

where:

<device_bus_id>
: is the device bus-ID that corresponds to the FCP channel.

<wwpn>
: is the WWPN of the target port.

<attribute>
: is one of the attributes in Table 12.

To read attributes of the associated fc_host use a command of this form:

```
# cat /sys/class/fc_remote_port/<rport_name>/<attribute>
```

where:

<rport_name>
: is the name of the remote port.

<attribute>
: is one of the attributes in Table 13.

Examples

- In this example, information is displayed for a target port 0x500507630300c562 that is attached through an FCP channel that corresponds to a device bus-ID 0.0.3d0c:

  ```
  # cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/access_denied
  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"
    node_name           = "0x5005076303ffc562"
    port_id             = "0x652113"
    port_name           = "0x500507630300c562"
    port_state          = "Online"
    roles               = "FCP Target"
    scsi_target_id      = "0"
```

Recovering a failed port

Before you start: The FCP channel must be online.

Failed target ports are automatically recovered by the zfcp device driver. You can read the in_recovery attribute to check if recovery is under way. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/in_recovery
```

where the variables are the same as in “Configuring SCSI devices” on page 65.

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.

To find out if recovery has 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.

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 connected through an FCP channel 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 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

Before you start: The FCP channel must be online.

List the contents of `/sys/bus/ccw/drivers/zfcp/<device_bus_id>` to find out which ports are currently configured for the FCP channel.

To remove a port from an FCP channel write the port's WWPN to the FCP channel's port_remove attribute. Issue a command of this form:

```
# echo <wwpn> > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/port_remove
```

where:

<device_bus_id>
: is the device bus-ID that corresponds to the FCP channel.

<wwpn>
: is the WWPN of the port to be removed.

You cannot remove a port while SCSI devices are configured for it (see “Configuring SCSI devices”) or if the port is in use, for example, by error recovery. Note that the next port scan will attach a removed port again if the port is available. If you do not want this, consider zoning.

Example

In this example, two ports with WWPN 0x500507630303c562 and 0x500507630300c562 have been configured for an FCP Channel with device bus-ID 0.0.3d0c. The port with WWPN 0x500507630303c562 is 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
```

Configuring SCSI devices

To configure a SCSI device for a target port 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
```

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>
: is the device bus-ID that corresponds to the FCP channel.

<wwpn>
: is the WWPN of the target port.

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.
2. It initiates the registration of the SCSI device with the Linux SCSI stack. The FCP channel device must be online for this step.
3. It waits until the Linux SCSI stack registration has completed successfully or returned 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 66).

To check if 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 channel device is offline in Linux.

To find out which SCSI devices are currently configured for the port, list the contents of `/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>`.

Example

In this example, a target port with WWPN 0x500507630300c562 is connected through an FCP channel with device 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
```

Mapping the representations of a SCSI device in sysfs

Each SCSI device that is configured is represented by multiple directories in sysfs. In particular:

- A directory in the zfcp branch (see “Configuring SCSI devices” on page 65)
- A directory in the SCSI branch

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>
: This is the scsi_host_number for the corresponding FCP channel.

<scsi_id>
: This is the scsi_id for the target port.

<scsi_lun>
: This is the scsi_lun for the SCSI device.

The values for scsi_id and scsi_lun depend on the storage device. Often, they are single-digit numbers but for some storage devices they have numerous digits.

Figure 13 shows how the directory name is composed of attributes of consecutive directories in the sysfs zfcp branch. You can find the name of the directory in the sysfs SCSI branch by reading the corresponding attributes in the zfcp branch.

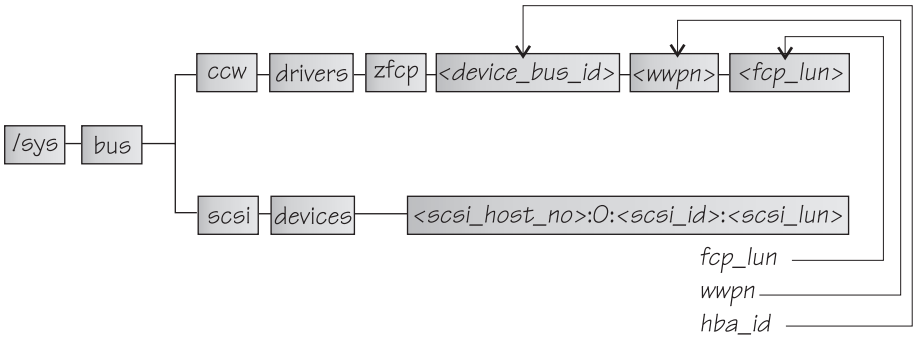


Figure 13. SCSI devices in sysfs

To find the SCSI device for a zfcp unit you must compare the SCSI device attributes `hba_id`, `wwpn`, and `fcp_lun` of all available SCSI devices with the triple consisting of *<device_bus_id>*, *<wwpn>* and *<fcp_lun>* of your zfcp unit.

To simplify this task, you can use **lszfcp** (see “lszfcp - List zfcp devices” on page 415).

Example

This example shows how you can use **lszfcp** to display the name of the SCSI device that corresponds to a zfcp unit, for example:

```
# lszfcp -l 0x4010403200000000
0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:0
```

In the example, the output informs you that the unit with the LUN 0x4010403200000000, which is configured on a port with the WWPN 0x500507630300c562 on an adapter with the device_bus_id 0.0.3d0c, maps to SCSI device "0:0:0:0".

To confirm that the SCSI device belongs to the zfcp unit:

```
# cat /sys/bus/scsi/devices/0:0:0:0/hba_id
0.0.3d0c
# cat /sys/bus/scsi/devices/0:0:0:0/wwpn
0x500507630300c562
# cat /sys/bus/scsi/devices/0:0:0:0/fcp_lun
0x4010403200000000
```

Displaying information about SCSI devices

For each SCSI device, there is a number of read-only attributes in sysfs that provide access information for the device. These attributes indicate if the device access is restricted by access control software on the FCP channel. Table 14 summarizes the relevant attributes.

Table 14. Attributes with device access information

Attribute	Explanation
access_denied	Flag that indicates if access to the device is restricted by access control software on the FCP channel. The value is “1” if access is denied and “0” if access is permitted. (See “FCP LUN access control” on page 53).
access_shared	Flag that indicates if access to the device is shared or exclusive. The value is “1” if access is shared and “0” if access is exclusive. (See “FCP LUN access control” on page 53).
access_readonly	Flag that indicates if write access to the device is permitted or if access is restricted to read-only. The value is “1” if access is restricted read-only and “0” if write access is permitted. (See “FCP LUN access control” on page 53).
in_recovery	Shows if unit is in recovery (0 or 1)

For each SCSI device, there are also read-only attributes with information about the device.

Table 15. SCSI device class attributes

Attribute	Explanation
device_blocked	Flag that indicates if device is in blocked state (0 or 1).

Table 15. SCSI device class attributes (continued)

Attribute	Explanation
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.
queue_type	The type of queue for the SCSI device. The value can be one of the following: • none • simple • ordered
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.
fcp_lun	The LUN of the SCSI device in 64-bit format.
hba_id	The bus ID of the SCSI device.
wwpn	The WWPN of the remote port.

Issue a command of this form to read an attribute:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<fcp_lun>/<attribute>
```

where:

<device_bus_id> is the device bus-ID that corresponds to the FCP channel.
<wwpn> is the WWPN of the target port.
<fcp_lun> is the FCP LUN of the SCSI device.
<attribute> is one of the attributes in Table 14 on page 67.

Use the **lszfcp** command (see “lszfcp - List zfcp devices” on page 415) 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>/<attribute>
```

where:

<device_name> is the name of the associated SCSI device.
<attribute> is one of the attributes in Table 15 on page 67.

Tip: For SCSI tape devices you can display a summary of this information by using the **lstape** command (see “lstape - List tape devices” on page 409).

Examples

- In this example, information is displayed for a SCSI device with LUN 0x4010403200000000 that is accessed through a target port with WWPN

0x500507630300c562 and is connected through an FCP channel with device bus-ID 0.0.3d0c. For the device, shared read-only access is permitted.

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000/access_denied
0
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000/access_shared
1
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000/access_readonly
1
```

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 63).

- You can use **lszfcp** to display attributes of a SCSI device:

```
# lszfcp -l 0x4010403200000000 -a
0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:0
Class = "scsi_device"
      device_blocked      = "0"
      fcp_lun             = "0x4010403200000000"
      hba_id              = "0.0.3d0c"
      iocounterbits       = "32"
      iodone_cnt          = "0x111"
      ioerr_cnt           = "0x1"
      iorequest_cnt       = "0x111"
      model               = "2107900         "
      queue_depth         = "32"
      queue_type          = "simple"
      rev                 = ".203"
      scsi_level          = "6"
      state               = "running"
      timeout             = "30"
      type                = "0"
      vendor              = "IBM     "
      wwpn                = "0x500507630300c562"
```

Setting the queue depth

Changing the queue depth is usually a storage server requirement. 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 value of the queue_depth kernel parameter (see “zfcp module parameters” on page 54) 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 kernel or module parameter.
- 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 number defined by 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 a failed SCSI device

Before you start: The FCP channel must be online.

Failed SCSI devices are automatically recovered by the zfcp device driver. You can read the in_recovery attribute to check if recovery is under way. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<scsi_lun>/in_recovery
```

where the variables have the same meaning as in “Configuring SCSI devices” on page 65.

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 or the device driver might have failed to detect that the SCSI device is malfunctioning.

To find out if recovery has failed read the failed attribute. Issue a command of this form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<scsi_lun>/failed
```

The value is “1” if recovery has failed and “0” otherwise.

You can start or restart the recovery process for the SCSI device by writing “0” to the failed attribute. Issue a command of this form:

```
# echo 0 > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<scsi_lun>/failed
```

Example

In the following example, SCSI device with LUN 0x4010403200000000 is malfunctioning, The SCSI device is accessed through a target port with WWPN 0x500507630300c562 that is connected through an FCP channel with a device bus

ID 0.0.3d0c. The first command reveals that recovery is not already under way. The second command manually starts recovery for the SCSI device:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000/in_recovery
0
# echo 0 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x4010403200000000/failed
```

Updating the information about SCSI devices

Before you start: The FCP channel must be online.

Information about the available SCSI devices is discovered automatically by the zfcp device driver when the adapter is activated. You can use the `rescan` attribute of the SCSI device to detect any subsequent changes that are made to a storage device on the storage server.

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 66.

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

Before you start: The FCP channel must be online.

There is a timeout for SCSI commands. If the timeout expires before a SCSI command has completed, error recovery starts. The default timeout is 30 seconds. You can change the timeout if the default is not suitable for your storage system.

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 66.

The attribute value specifies the timeout in seconds.

To set a different timeout, enter a command of this form:

```
# echo <timeout> > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<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

Before you start: The FCP channel must be online.

If the connection to a storage system is working but the storage system has a problem, the error recovery can stop with taking the SCSI device offline. This condition is indicated by a message like “Device offlined - not ready after error recovery”. You can use the `state` attribute of the SCSI device to set the device back online.

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 66. 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.

To set an offline device online again, write `running` to the `state` attribute. 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 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

To remove a SCSI device you need to remove it from the target port.

Remove the device from the port by writing the device's LUN to the port's unit_remove attribute. 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 65. Removing a LUN with unit_remove automatically unregisters the SCSI device first.

Should you wish to unregister the device manually, you can do so by writing “1” to the delete attribute of the directory that represents the device in the sysfs SCSI branch. See “Mapping the representations of a SCSI device in sysfs” on page 66 for information on how to find this directory. Issue a command of this form:

```
# echo 1 > /sys/bus/scsi/devices/<device>/delete
```

Example

The following example removes a SCSI device with LUN 0x4010403200000000, accessed through a target port with WWPN 0x500507630300c562 and an FCP channel with a device bus-ID 0.0.3d0c. The corresponding directory in the sysfs SCSI branch is assumed to be `/sys/bus/scsi/devices/0:0:1:1`.

1. Optionally, delete the device:

   ```
   # echo 1 > /sys/bus/scsi/devices/0:0:1:1/delete
   ```

2. Remove the device:

   ```
   # echo 0x4010403200000000 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/unit_remove
   ```

Scenario

The following scenario describes the steps from setting an FC adapter online to listing the available LUNs.

1. Check for available FC adapters 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 yes  80  80  ff   36000000 00000000
   ```

   Another possible type would be, for example, 1732/04.

2. Set the adapter online:

   ```
   # chccwdev 0.0.3c02 --online
   ```

   A port scan is performed automatically after the adapter is set online.

3. Optional: Confirm that the adapter is available and online:

   ```
   # lszfcp
   0.0.3c02 host0
   ```

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 adapter 0.0.3c02, port `0x50050763030bc562`:

```
# lsluns -p 0x50050763030bc562
Scanning for LUNs on adapter 0.0.3c02
        at port 0x50050763030bc562:
                0x4010400000000000
                0x4010400100000000
                0x4010400200000000
                0x4010400300000000
                0x4010400400000000
                0x4010400500000000
                0x4010400600000000
                ...
```

API provided by the zfcp HBA API support

This section provides information for those who want to program SAN management clients that run on Red Hat Enterprise Linux 6 for System z.

Functions provided

The zfcp HBA API (see “zfcp HBA API (FC-HBA) support” on page 52) is defined in the Fibre Channel - HBA API (FC-HBA) specification (see www.t11.org).

The zfcp HBA API implements the following FC-HBA functions:

- 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()

All other FC-HBA functions return status code HBA_STATUS_ERROR_NOT_SUPPORTED where possible.

Note: ZFCP HBA API for Linux 2.6 can access only adapters, ports and units that are configured in the operating system.

Environment variables

The zfcp HBA API support uses the following environment variables for logging errors in the zfcp HBA API library:

LIB_ZFCP_HBAAPI_LOG_LEVEL
to specify 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.

Chapter 6. Channel-attached tape device driver

The tape device driver supports channel-attached tape devices on Red Hat Enterprise Linux 6 for System z.

SCSI tape devices attached through a System z FCP adapter are handled by the zfcp device driver (see Chapter 5, “SCSI-over-Fibre Channel device driver,” on page 49).

Features

The tape device driver supports the following devices and functions:

- 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.
- Character and block devices (see “Tape device modes and logical devices”).
- Control operations through **mt** (see “Using the mt command” on page 80).
- Message display support (see “tape390_display - display messages on tape devices and load tapes” on page 441).
- Encryption support (see “tape390_crypt - manage tape encryption” on page 437).
- Up to 128 physical tape devices.

What you should know about channel-attached tape devices

This section provides information about the available operation modes, about devices names, and about device nodes for your channel-attached tape devices.

Tape device modes and logical devices

The tape device driver supports up to 128 physical tape devices. Each physical tape device can be used in three different modes. The tape device driver treats each mode as a separate logical device:

Non-rewinding character device
: Provides sequential (traditional) tape access without any caching done in the kernel.

  You can use the character device in the same way as any other Linux tape device. You can write to it and read from it using normal 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**. Most Linux tape software should work with the character device.

  When the device is closed, the tape is left at the current position.

Rewinding character device
: Provides tape access like the non-rewinding device, except that the tape is rewound when the device is closed.

Block device
: Provides a read-only tape block device.

  This device could be used for the installation of software in the same way as tapes are used under other operating systems on the System z

platforms. (This is similar to the way most Linux software distributions are shipped on CD using the ISO9660 file system.)

It is advisable to use only the ISO9660 file system on System z tapes, because this file system is optimized for CD-ROM devices, which – just like 3480, 3490, or 3590 tape devices – cannot perform fast searches.

The ISO9660 file system image file need not be the first file on the tape but can start at any position. The tape must be positioned at the start of the image file before the mount command is issued to the tape block device.

The file system image must reside on a single tape. Tape block devices cannot span multiple tape volumes.

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 16:

Table 16. 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
Block devices	btibm*<n>*	2×*<n>*

where *<n>* is the index number 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 409) 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 and a major number to channel-attached block tape devices. The major numbers can but need not be the same. 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 on the major/minor assignments. The directories have the form:

- `/sys/class/tape390/ntibm<n>`
- `/sys/class/tape390/rtibm<n>`
- `/sys/block/btibm<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 for the character or block tape devices 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 has been allocated to the tape device in the first row. This means that 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
```

The resulting names and minor numbers for the online devices are:

Bus ID	Index (TapeNo)	Device	Device name	Minor number
0.0.01ac	not assigned	not assigned		not assigned
0.0.01a1	0	non-rewind	ntibm0	0
		rewind	rtibm0	1
		block	btibm0	0
0.0.01a0	1	non-rewind	ntibm1	2
		rewind	rtibm1	3
		block	btibm1	2
0.0.0172	2	non-rewind	ntibm2	4
		rewind	rtibm2	5
		block	btibm2	4

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 used for character devices is 254 the minor numbers are as expected for the respective device names.

Similarly, the major/minor assignments for the online block devices can be read from their respective representations in `/sys/block`:

```
# cat /sys/block/btibm0/dev
254:0
# cat /sys/block/btibm1/dev
254:2
# cat /sys/block/btibm2/dev
254:4
```

The minor numbers are as expected for the respective device names. In the example, the major number used for block devices is also 254.

Tape device nodes

User space programs access tape devices by *device nodes*. Red Hat Enterprise Linux 6 uses udev to create three 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 78.

For example, if you have two tape devices, udev will create the device nodes shown in Table 17:

Table 17. Tape device nodes

Node for	non-rewind device	rewind device	block device
First tape device	`/dev/ntibm0`	`/dev/rtibm0`	`/dev/btibm0`
Second tape device	`/dev/ntibm1`	`/dev/rtibm1`	`/dev/btibm1`

Using the mt command

Basic Linux tape control is handled by the **mt** utility. Refer to the man page for general information on **mt**.

Be aware that for channel-attached tape hardware there are some differences in the MTIO interface with corresponding differences for some operations of the **mt** command:

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 85 for details).

Setting up 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

```
►►─modprobe──┬─ tape_34xx ─┬──────────────────────────────────────►◄
             └─ tape_3590 ─┘
```

See the **modprobe** man page for details on **modprobe**.

Working with the tape device driver

This section describes typical tasks that you need to perform when working with tape devices:

- Setting a tape device online or offline
- Displaying tape information
- Enabling compression
- Loading and unloading tapes

For information on working with the channel measurement facility, see Chapter 37, “Channel measurement facility,” on page 345.

For information on how to display messages on a tape device's display unit, see “tape390_display - display messages on tape devices and load tapes” on page 441.

Setting a tape device online or offline

Setting a physical tape device online makes all corresponding logical devices accessible:

- The non-rewind character device
- The rewind character device
- The block device (if supported)

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.

Use the **chccwdev** command (see “chccwdev - Set a CCW device online” on page 362) 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.

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 80) to identify the corresponding logical devices. The index number is in the range 0 to 127. A maximum of 128 physical tape devices can be online concurrently.

If you are using the standard device nodes, you need to 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 has been 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.

Examples

- To set a physical tape device with device bus-ID 0.0.015f online, issue:

  ```
  # chccwdev -e 0.0.015f
  ```

  or

  ```
  # echo 1 > /sys/bus/ccw/devices/0.0.015f/online
  ```

  To find the index number the tape device driver has assigned, 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
  - `/dev/btibm2` for the block device
- To set a physical tape device with device bus-ID 0.0.015f offline, issue:

  ```
  # chccwdev -d 0.0.015f
  ```

  or

  ```
  # echo 0 > /sys/bus/ccw/devices/0.0.015f/online
  ```

Displaying tape information

Use the **lstape** command (see “lstape - List tape devices” on page 409) to display summary information about your tape devices.

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 18. 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 81)
cmb_enable	"1" if channel measurement block is enabled for the physical device or "0" if it is not enabled (see Chapter 37, "Channel measurement facility," on page 345)
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 currently 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 **BLKUSE** Device is being used as a block device by a process on this Linux image **OFFLINE** The device is offline. **NOT_OP** Device is not operational
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

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.

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

To control Improved Data Recording Capability (IDRC) compression, use the **mt** command provided by the RPM mt-st.

Compression is off after the tape device driver has loaded. To switch compression on, 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 switch compression off, issue:

```
# mt -f <tape> compression 0
```

Any other numeric value has no effect, and any other argument switches compression off.

Example

To switch on compression for a tape device with a device node `/dev/ntibm0` issue:

```
# mt -f /dev/ntibm0 compression 1
```

Loading and unloading tapes

You can 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 441) 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 441). 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.

Scenario: Using a tape block device

In this scenario, an ISO9660 file system is to be created as the second file on a tape. The scenario uses the **mt** and **mkisofs** commands. Refer to the respective man pages for details.

Assumptions: The following assumptions are made:

- The required tape device driver modules have either been compiled into the kernel or have already been loaded.
- The ISO9660 file system support has been compiled into the kernel.
- A tape device is attached through a device bus-ID 0.0.015f.

1. Create a Linux directory, `somedir`, and fill it with the contents of the file system:

```
# mkdir somedir
# cp <contents> somedir
```

2. Set the tape online:

```
# chccwdev -e 0.0.015f
```

3. If you are using standard device nodes, find out which index number the tape device driver has assigned to it. You can skip this step if you are using udev-created device nodes that distinguish devices by device bus-ID rather than the index number.

```
# lstape 0.0.015f --ccw-only
TapeNo  BusID      CuType/Model DevType/Model   BlkSize State   Op      MedState
1       0.0.015f   3480/01      3480/04         auto    UNUSED  ---     LOADED
```

   The index number is shown in the TapeNo column of the command output, “1” in the example. The standard device nodes are therefore `/dev/ntibm1`, `/dev/rtibm1`, and `/dev/btibm1`.

4. Insert a tape.
5. Ensure the tape is positioned at the correct position on the tape. For example, to set it to the beginning of the second file, issue:

```
# mt -f /dev/ntibm1 rewind
# mt -f /dev/ntibm1 fsf 1
```

   `fsf` skips a specified number of files, one in the example.

6. Set the block size of the character driver. (The block size 2048 bytes is commonly used on ISO9660 CD-ROMs.)

```
# mt -f /dev/ntibm1 setblk 2048
```

7. Write the file system to the character device driver:

```
# mkisofs -l -f -o file.iso somedir
# dd if=file.iso of=/dev/ntibm1 bs=2048
```

8. Set the tape to the beginning of the file:

```
# mt -f /dev/ntibm1 rewind
# mt -f /dev/ntibm1 fsf 1
```

9. Now you can mount your new file system as a block device:

```
# mount -t iso9660 -o ro,block=2048 /dev/btibm1 /mnt
```

Chapter 7. XPRAM device driver

With the XPRAM block device driver Red Hat Enterprise Linux 6 for System z can access expanded storage. Thus XPRAM can be used as a basis for fast swap devices and/or fast file systems. Expanded storage range can be swapped in or out of the main storage in 4 KB blocks. All XPRAM devices do always provide a block size of 4096 bytes.

XPRAM features

The XPRAM device driver provides the following features:

- Automatic detection of expanded storage.

  If expanded storage is not available, XPRAM fails gracefully with a log message reporting the absence of expanded storage.
- The expanded storage can be divided into up to 32 partitions.

What you should know about XPRAM

This section provides information on XPRAM partitions and the device nodes that make them accessible.

XPRAM partitions and device nodes

The XPRAM device driver uses major number 35. The standard device names are of the form slram*<n>*, where *<n>* is the corresponding minor number.

You can use the entire available expanded storage as a single XPRAM device or divide it into up to 32 partitions. Each partition is treated as a separate XPRAM device.

If the entire expanded storage is used a single device, the device name is slram0. For partitioned expanded storage, the *<n>* in the device name denotes the (n+1)th partition. For example, the first partition is called slram0, the second slram1, and the 32nd partition is called slram31.

Table 19. XPRAM device names, minor numbers, and partitions

Minor	Name	To access
0	slram0	the first partition or the entire expanded storage if there are no partitions
1	slram1	the second partition
2	slram2	the third partition
... <n> ...	... slram<n> ...	... the (<n>+1)th partition ...
31	slram31	the 32nd partition

The device nodes that you need to access these partitions are created by udev when you load the XPRAM device driver module. The nodes are of the form `/dev/slram`*<n>*, where *<n>* is the index number of the partition. In addition, to the device nodes udev creates a symbolic link of the form `/dev/xpram`*<n>* that points to the respective device node.

XPRAM use for diagnosis

Issuing an IPL command to reboot Linux does not reset expanded storage, so it is persistent across IPLs and could be used, for example, to store diagnostic information. The expanded storage is reset when logging off the z/VM guest virtual machine or when deactivating the LPAR.

Reusing XPRAM partitions

You might be able to reuse existing file systems or swap devices on an XPRAM device or partition after reloading the XPRAM device driver (for example, after rebooting Linux). For file systems or swap devices to be reusable, the XPRAM kernel or module parameters for the new device or partition must match the parameters of the previous use of XPRAM.

If you change the XPRAM parameters, you must create a new file system (for example with **mke2fs**) or a new swap device for each partition that has changed. A device or partition is considered changed if its size has changed. All partitions following a changed partition are also considered changed even if their sizes are unchanged.

Setting up the XPRAM device driver

This section describes how to load the XPRAM device driver and how to split the available expanded storage into partitions.

You can optionally partition the available expanded storage by using the devs and sizes module parameters when you load the xpram module.

XPRAM module parameter syntax

```
►►─modprobe─ xpram──┬──────────────────────────────────────────────────────────┬──►◄
                    └─ devs=<number_of_partitions>──┬──────────────────────────┬─┘
                                                    │          ┌──────────┐    │
                                                    │          ↓          │    │
                                                    └─ sizes=──────<partition_size>──┘
```

where:

<number_of_partitions>
: is an integer in the range 1 to 32 that defines how many partitions the expanded storage is split into.

<partition_size>
: specifies the size of a partition. The i-th value defines the size of the i-th partition.

  Each size is a non-negative integer that defines the size of the partition in KB or a blank. Only decimal values are allowed and no magnitudes are accepted.

  You can specify up to *<number_of_partitions>* values. If you specify less values than *<number_of_partitions>*, the missing values are interpreted as blanks. Blanks are treated like zeros.

Any partition defined with a non-zero size is allocated the amount of memory specified by its size parameter.

Any remaining memory is divided as equally as possible among any partitions with a zero or blank size parameter, subject to the two constraints that blocks must be allocated in multiples of 4K and addressing constraints may leave un-allocated areas of memory between partitions.

Examples

- The following specification allocates the extended storage into four partitions. Partition 1 has 2 GB (2097152 KB), partition 4 has 4 GB (4194304 KB), and partitions 2 and 3 use equal parts of the remaining storage. If the total amount of extended storage was 16 GB, then partitions 3 and 4 would each have approximately 5 GB.

```
# modprobe xpram devs=4 sizes=2097152,0,0,4194304
```

- The following specification allocates the extended storage into three partitions. The partition 2 has 512 KB and the partitions 1 and 3 use equal parts of the remaining extended storage.

```
# modprobe xpram devs=3 sizes=,512
```

- The following specification allocates the extended storage into two partitions of equal size.

```
# modprobe xpram devs=2
```

Part 3. Networking

This part describes the network device drivers for Red Hat Enterprise Linux 6 for System z.

Newest version: You can find the newest version of this book at
`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the Red Hat Enterprise Linux 6 release notes at
`http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/`

An example network setup that uses some available network setup types is shown in Figure 14 on page 92.

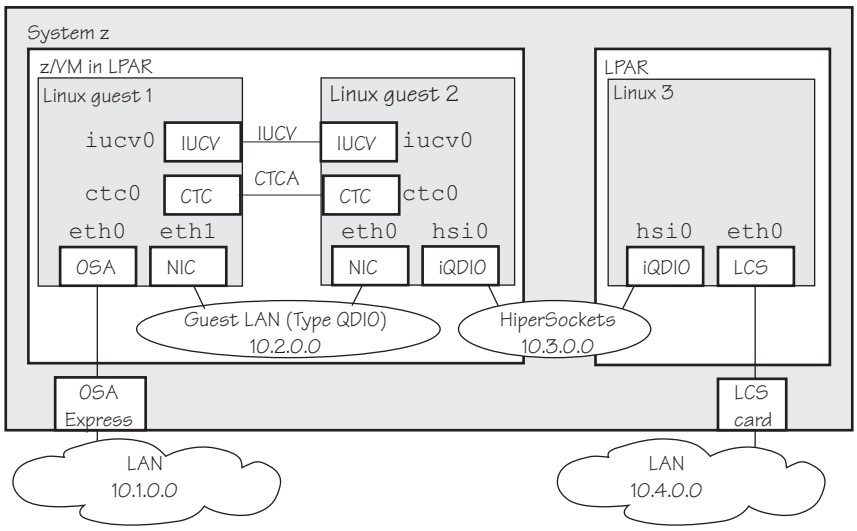


Figure 14. Networking example

In the example there are three Linux instances; two of them run as z/VM guest operating systems in one LPAR and a third Linux instance runs in another LPAR. Within z/VM, Linux instances can be connected directly by virtual-CTC, or 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 System z mainframe to an external network.

Chapter 8. qeth device driver for OSA-Express (QDIO) and HiperSockets

The qeth device driver supports a number of networking possibilities, among them:

Real connections using OSA-Express

A System z mainframe offers OSA-Express adapters, which are real LAN-adapter hardware, see Figure 15. 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 OSN in the hardware configuration). OSD-devices are the standard System z LAN-adapters, while OSN-devices serve as NCP-adapters. For details on OSA-Express in QDIO mode, see *OSA-Express Customer's Guide and Reference*, SA22-7935.

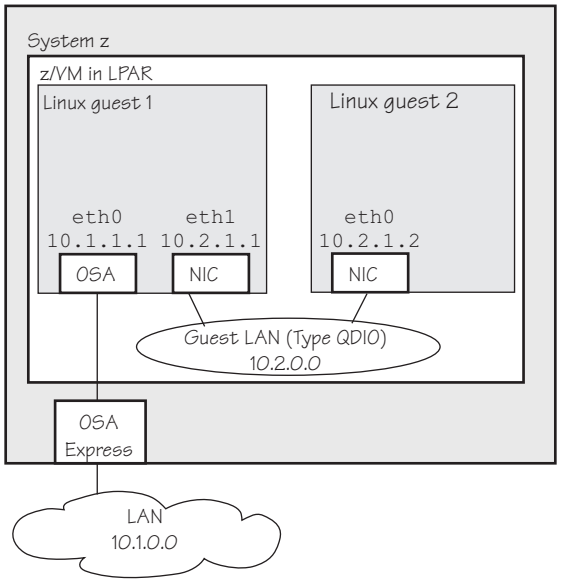


Figure 15. OSA-Express adapters are real LAN-adapter hardware

The OSA-Express LAN adapter may serve as a Network Control Program (NCP) adapter for an internal ESCON/CDLC interface to another mainframe operating system. This feature is exploited by the IBM Communication Controller for Linux (CCL) introduced with System z9. Note that the OSA CHPID type does not support any additional network functions and its only purpose is to provide a bridge between the CDLC and QDIO interfaces to connect to the Linux NCP. For more details see the *IBM Communication Controller Migration Guide*, SG24-6298.

As of zEnterprise, the qeth device driver supports CHPIDs of type OSM and OSX. CHPID OSM (OSA-Express for Unified Resource Manager) provides connectivity to the intranode management network (INMN) from Unified Resource Manager functions to zCPC. CHPID OSX (OSA-Express for zBX) provides connectivity to and access control for the intraensemble data network (IEDN), which is managed by Unified Resource Manager functions and connects zCPCs and zBXs within an ensemble.

HiperSockets

A System z mainframe offers internal connections called *HiperSockets*.

These simulate QDIO network adapters and provide high-speed TCP/IP communication for operating system instances within and across LPARs. For details on HiperSockets, see *HiperSockets Implementation Guide*, SG24-6816.

Virtual connections when running Linux as a z/VM guest operating system

z/VM offers virtualized LAN-adapters that enable connections between z/VM guests 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 guests, 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).

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 16.

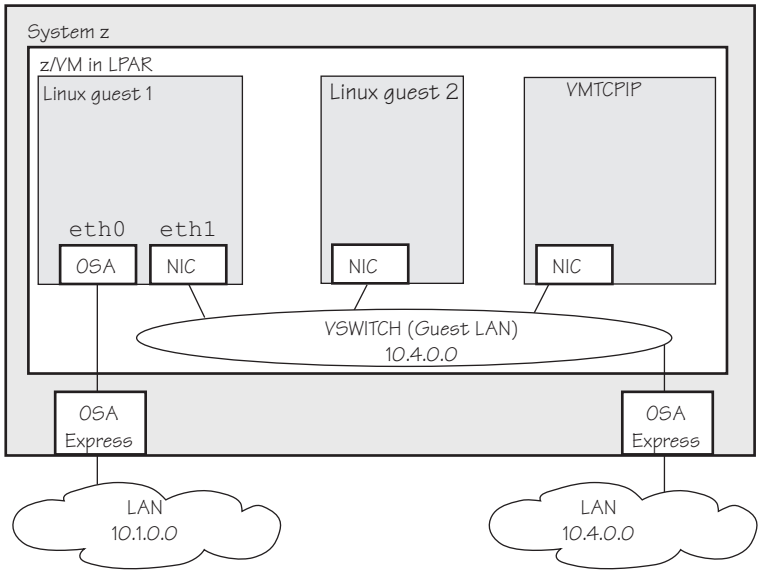


Figure 16. Virtual switch

A dedicated OSA adapter can be an option, but is not required for a VSWITCH.

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, see *z/VM Connectivity*, SC24-6174.

The qeth network device driver supports System z OSA-Express3, OSA-Express2, and OSA-Express features in QDIO mode, HiperSockets, z/VM guest LAN, and z/VM VSWITCH as follows:

Table 20. The qeth device driver supported OSA-Express features

Feature	System z196	System z10	System z9
Hipersockets	Yes	Yes	Yes (layer 3 only)
OSA-Express3			
Gigabit Ethernet	Yes	Yes	No
10 Gigabit Ethernet	Yes	Yes	No
1000Base-T Ethernet	Yes	Yes	No
OSA-Express2			
Gigabit Ethernet	Yes	Yes	Yes
10 Gigabit Ethernet	No	Yes	Yes
1000Base-T Ethernet	Yes	Yes	Yes
OSA-Express			
Fast Ethernet	No	No	Yes
1000Base-T Ethernet	No	No	Yes
Gigabit Ethernet	No	No	Yes

Note: Unless otherwise indicated, OSA-Express refers to OSA-Express, OSA-Express2, and OSA-Express3.

Device driver functions

The qeth device driver supports functions listed in Table 21 and Table 22 on page 96.

Table 21. Real connections

Function	OSA Layer 2	OSA Layer 3	HiperSockets Layer 2 Ethernet	HiperSockets Layer 3 Ethernet
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/sw/No
Linux ARP	Yes	No (hw ARP)	Yes	No
Linux neighbor solicitation	Yes	Yes	Yes	No
Unique MAC address	Yes (random)	No	Yes	Yes
Change MAC address	Yes	No	Yes	No

Table 21. Real connections (continued)

Function	OSA Layer 2	OSA Layer 3	HiperSockets Layer 2 Ethernet	HiperSockets Layer 3 Ethernet
Promiscuous mode	No	No	No	• 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	Yes	Yes
Secondary unicast MAC address	Yes	No	Yes	No
Offload features				
TCP segmentation offload (TSO)	No	Yes	No	No
rx HW checksum	No	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
Legend: **No** Function not supported or not required **hw** Function performed by hardware **sw** Function performed by software **Yes** Function supported				

Table 22. Guest LAN connections

Function	OSA Layer 2	OSA Layer 3	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

Table 22. Guest LAN connections (continued)

Function	OSA Layer 2	OSA Layer 3	HiperSockets (Layer 3)
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	No	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	Yes	Yes	Yes
Secondary unicast MAC address	Yes	No	No
Offload features			
TSO	No	No	No
rx HW checksum	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
Direct connectivity to z/OS	No	Yes	Yes
SNMP support	No	No	No
Multiport support	No	No	No
Data connection isolation	No	No	No
Legend: **No** Function not supported or not required **hw** Function performed by hardware **sw** Function performed by software **Yes** Function supported			

What you should know about the qeth device driver

This section describes qeth group devices in relation to subchannels and their corresponding device numbers and device bus-IDs. It also describes the interface names that are assigned to qeth group devices and how an OSA-Express adapter handles IPv4 and IPv6 packets.

Layer 2 and layer 3

The qeth device driver consists of a common core and two device disciplines:

The layer 2 discipline (qeth_l2)

The layer 2 discipline supports:

- OSA and OSA guest LAN devices
- OSA for NCP devices
- HiperSockets devices (as of System z10)
- OSM devices for Unified Resource Manager
- OSX for OSA-Express devices for zBX

The layer 2 discipline is the default setup for OSA. On HiperSockets the default continues to be layer 3. OSA guest LANs are layer 2 by default, while HiperSockets guest LANs are always layer 3. See “Setting the layer2 attribute” on page 108 for details.

The layer 3 discipline (qeth_l3)

The layer 3 discipline supports:

- OSA and OSA guest LAN devices running in layer 3 mode (with faked link layer headers)
- HiperSockets and HiperSockets guest LAN devices running in layer 3 mode (with faked link layer headers)
- OSX for OSA-Express devices for zBX

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 27 on page 105. Some performance-critical applications might benefit from being layer 3.

Keep layer 2 and layer 3 guest LANs separate and keep layer 2 and layer 3 HiperSockets LANs separate. Layer 2 and layer 3 interfaces cannot communicate within a HiperSockets LAN or guest LAN.

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.

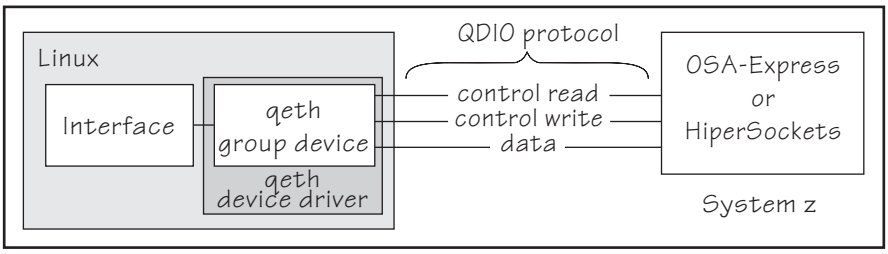


Figure 17. I/O subchannel interface

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

Before you start: 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.

You need to perform several steps before user-space applications on your Linux instance can 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.

These tasks and the configuration options are described in detail in “Working with the qeth device driver” on page 104.

qeth interface names and device directories

The qeth device driver automatically assigns interface names to the qeth group devices and creates the corresponding sysfs structures. According to the type of CHPID and feature used, the naming scheme uses the following base names:

eth<*n*> for Ethernet features.

hsi<*n*> for HiperSockets devices.

osn<*n*> for ESCON/CDLC bridge (OSA NCP).

where <*n*> is an integer that uniquely identifies the device. When the first device for a base name is set online it is assigned 0, the second is assigned 1, the third 2, and so on. Each base name is counted separately.

For example, the interface name of the first Ethernet feature that is set online is “eth0”, the second “eth1”, and so on. When the first HiperSockets device is set online, it is assigned the interface name “hsi0”.

While an interface is online, it is represented in sysfs as:

`/sys/class/net/<interface>`

The qeth device driver shares the name space for Ethernet interfaces with the LCS device driver. Each driver uses the name with the lowest free identifier <*n*>, regardless of which device driver occupies the other names. For example, if the first qeth Ethernet feature is set online and there is already one LCS Ethernet feature online, the LCS feature is named “eth0” and the qeth feature is named “eth1”. See also “LCS interface names” on page 153.

The mapping between interface names and the device bus-ID that represents the qeth group device in sysfs is preserved when a device is set offline and back

online. However, it can change when rebooting, when devices are ungrouped, or when devices appear or disappear with a machine check.

“Finding out the interface name of a qeth group device” on page 113 and “Finding out the bus ID of a qeth interface” on page 113 provide information on how to map device bus-IDs and interface names.

Support for IP Version 6 (IPv6)

IPv6 is supported on:

- Ethernet interfaces of the OSA-Express adapter running 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 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.

Using IPv6 is largely transparent to users. You must be aware of the IP version when specifying IP addresses and when using commands that return IP version specific output (for example, 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 (see Figure 18).

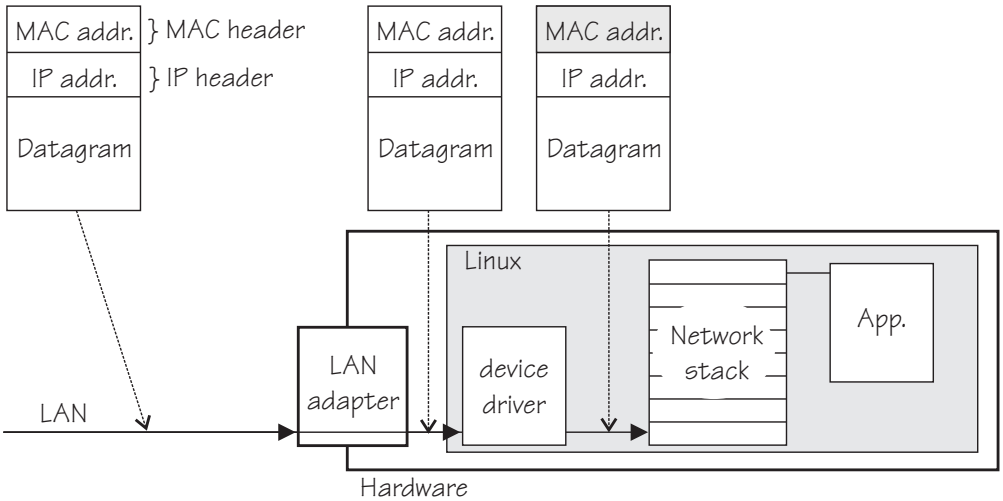


Figure 18. Standard IPv4 processing

MAC address handling as shown in Figure 18) 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 18 on page 100. 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. Users working with a directly attached OSA-card should assign a unique MAC-address themselves (see below).

For HiperSockets connections, a MAC address is generated.

For connections within a QDIO based z/VM guest LAN environment, z/VM assigns the necessary MAC addresses to its guests.

For Linux instances that are directly attached to an OSA-Express adapter in QDIO mode, you should assign the MAC addresses yourself. You can add a line `LLADDR='<MAC address>'` to the configuration file `/etc/sysconfig/network-scripts/ifcfg-<if-name>` Alternatively, you can change the MAC address by issuing the command:

```
ifconfig <interface> hw ether <MAC address>
```

Note: Be sure not to assign the MAC address of the OSA-Express adapter to your Linux instance.

For OSX and OSM CHPIDs you cannot set your own MAC addresses. Linux uses the MAC addresses defined by the Unified Resource Manager.

MAC headers in layer 3 mode

Since a qeth layer 3 mode device driver is an Ethernet offload engine for IPv4 and a partial Ethernet offload engine for IPv6 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 and uses the registered IP addresses to forward a packet to the recipient TCP/IP stack. Thus the OSA-Express adapter is able to deliver IPv4 packets to the correct Linux images. Apart from broadcast packets, a Linux image can only get packets for IP addresses it has configured in the stack and registered with the OSA-Express adapter.

Because the OSA-Express QDIO microcode builds MAC headers for outgoing IPv4 packets and removes them from incoming IPv4 packets, the operating systems' network stacks only send and receive IPv4 packets without MAC headers.

This 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 140.

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 23. 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 24. Ethernet addresses of incoming frames

Frame	Destination address	Source address
IPv4	Real device address	FAKELL
IPv6	Real device address	FAKELL
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 25.

Table 25. 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. Since 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.

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 to automatically load all required additional modules in the correct order:

qeth module syntax

```
►►─modprobe──┬─ qeth ──┬──────────────────────────────────────────►◄
             ├─ qeth_l2─┤
             └─ qeth_l3─┘
```

where:

qeth is the core module that contains common functions 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 device must be offline. If the new discipline is accepted by the device driver the old network interface will be 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 the qeth device driver

This section provides an overview of the typical tasks that you need to perform when working with qeth group devices.

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 *Red Hat Enterprise Linux 6 Installation Guide*. For a general discussion of network configuration files, see *Red Hat Enterprise Linux 6 Deployment Guide*.

Table 26 on page 105 and Table 27 on page 105 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 **znetconf** command to configure devices (except for OSX and OSM devices) instead of using the attributes directly (see “znetconf - List and configure network devices” on page 455).

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. Refer to the respective task descriptions to see the applicability of each attribute.

OSA for NCP handles NCP-related packets. Most of the attributes do not apply to OSA for NCP devices. The attributes that apply are:

- if_name
- card_type
- buffer_count
- recover

Table 26. qeth tasks and attributes common to layer2 and layer3.

Task	Corresponding attributes	Possible attribute values
“Creating a qeth group device” on page 107	group	n/a, see “Devices and device attributes” on page 8
“Removing a qeth group device” on page 108	ungroup	0 or 1
“Setting the layer2 attribute” on page 108	layer2	0 or 1, see “Layer 2 and layer 3” on page 97[1]
	portname	any valid port name
“Providing Large Send - TCP segmentation offload” on page 109	large_send	no TSO
“Using priority queueing” on page 109	priority_queueing	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 110	buffer_count	integer in the range 8 to 128, the default is 16
“Specifying the relative port number” on page 111	portno	integer, either 0 or 1, the default is 0
“Finding out the type of your network adapter” on page 112	card_type	n/a, read-only
“Setting a device online or offline” on page 112	online	0 or 1
“Finding out the interface name of a qeth group device” on page 113	if_name	n/a, read-only
“Finding out the bus ID of a qeth interface” on page 113	none	n/a
“Activating an interface” on page 113	none	n/a
“Deactivating an interface” on page 115	none	n/a
“Recovering a device” on page 116	recover	1
“Isolating data connections” on page 116	isolation	none, drop, forward
“Starting and stopping collection of QETH performance statistics” on page 117	performance_stats	0 or 1

[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 27. qeth tasks and attributes in layer 3 mode.

Task	Corresponding attributes	Possible attribute values
“Setting up a Linux router” on page 118	route4 route6	primary_router secondary_router primary_connector secondary_connector multicast_router no_router
“Setting the checksumming method” on page 121	checksumming	hw_checksumming sw_checksumming no_checksumming
“Faking broadcast capability” on page 121	fake_broadcast [1]	0 or 1

Table 27. qeth tasks and attributes in layer 3 mode (continued).

Task	Corresponding attributes	Possible attribute values
"Taking over IP addresses" on page 122	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 125	rxip/add4 rxip/add6 rxip/del4 rxip/del6	IPv4 or IPv6 IP address
"Configuring a device for virtual IP address (VIPA)" on page 126	vipa/add4 vipa/add6 vipa/del4 vipa/del6	IPv4 or IPv6 IP address
"Setting up a HiperSockets network traffic analyzer" on page 141	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 431).

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 eth0:

```
/sys/bus/ccwgroup/drivers/qeth/0.0.a100
/sys/bus/ccwgroup/devices/0.0.a100
/sys/devices/qeth/0.0.a100
/sys/class/net/eth0/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
eth0
# cat /sys/bus/ccwgroup/devices/0.0.a100/if_name
eth0
# cat /sys/devices/qeth/0.0.a100/if_name
eth0
# cat /sys/class/net/eth0/device/if_name
eth0
```

However, the path through the `/sys/class/net` branch is available only while the device is online. Furthermore, it might lead to a different device if the assignment of interface names changes 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 **znetconf** to configure network devices (see “znetconf - List and configure network devices” on page 455). Alternatively, yo can use sysfs as described in this section.

Before you start: You need to 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.

To define a qeth group device, write the device numbers of the subchannel triplet to `/sys/bus/ccwgroup/drivers/qeth/group`. 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
```

Result: 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, a single OSA-Express CHPID in QDIO mode is used to connect a Linux instance to a network.

Mainframe configuration:

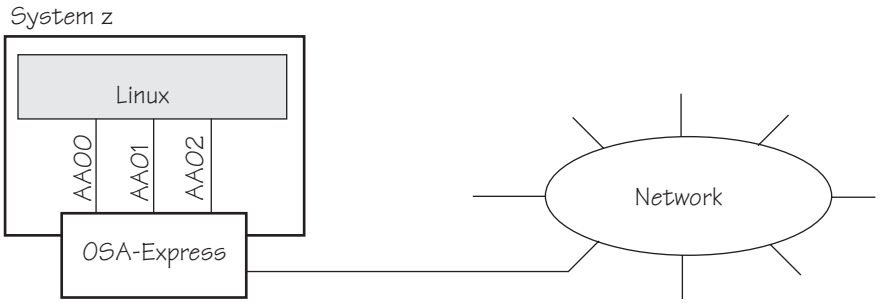


Figure 19. Mainframe configuration

Linux configuration:

Assuming that 0xaa00 is the device number 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

Before you start: The device must be set offline before you can remove it.

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 is known to be exclusively run in a discipline (for example, OSN needs the layer 2 discipline) the corresponding discipline module is automatically requested.

The qeth device driver attempts to load the layer3 discipline for HiperSockets devices and layer2 for non-HiperSockets devices.

You can make use of the layer2-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
: HiperSockets on layer 2 are supported as of System z10. 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
: Linux has to configure the same mode as the underlying z/VM virtual LAN definition. The z/VM definition "Ethernet mode" is available for VSWITCHes and for guest LANs of type QDIO.

Before you start: 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.

The qeth device driver separates the configuration options in sysfs regarding to the device discipline. Hence the first configuration action after grouping the device must be the configuration of the discipline. To set the discipline, issue a command of the form:

```
echo <integer> > /sys/devices/qeth/<device_bus_id>/layer2
```

where *<integer>* is

- 0 to turn the layer2 attribute off; this results in the layer 3 discipline.

- 1 to turn the layer2 attribute on; this results in the layer 2 discipline (default).

If the layer2 attribute has a value of -1 the layer has not been set and the default layer setting is used when the device is set online.

If you configured the discipline successfully, additional configuration attributes are displayed (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 it is a layer 2 device and tries to load the layer 2 discipline.

For information on layer2, refer to:

- *OSA-Express Customer's Guide and Reference*, SA22-7935
- *OSA-Express Implementation Guide*, SG25-5848
- *Networking Overview for Linux on zSeries*, REDP-3901
- *z/VM Connectivity*, SC24-6174

Providing Large Send - TCP segmentation offload

Before you start: Large Send is available only for real OSA in layer 3 mode.

Large Send enables you to offload the TCP segmentation operation from the Linux network stack to the OSA-Express2 or OSA-Express3 features. Large Send can lead to enhanced performance and latency for interfaces with predominately large outgoing packets.

To set Large Send, issue a command of the form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/large_send
```

where *<value>* can be any one of:

no No Large Send is provided. The Linux network stack performs the segmentation. This is the default.

TSO
The network adapter provides hardware Large Send. You can use hardware Large Send for an OSA-Express2 or OSA-Express3 that connects to an interface through a real LAN.

The qeth device driver does not check if the destination IP address is able to receive TCP segmentation offloaded packets. Thus it will send out the packet, which, if systems share an OSA-Express2 or OSA-Express3 CHPID, will lead to unpredictable results for the receiving system.

Examples

- To enable hardware Large Send for a device 0x1a10 issue:

```
# echo TSO > /sys/bus/ccwgroup/drivers/qeth/0.0.1a10/large_send
```

Using priority queueing

Before you start:

- This section applies to OSA-Express CHPIDs in QDIO mode only.
- The device must be offline while you set the queueing options.

An OSA-Express CHPID in QDIO mode has four output queues (queues 0 to 3) in central storage. The priority queueing feature gives these queues different priorities (queue 0 having the highest priority). 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.

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:

```
# echo <method> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/priority_queueing
```

where *<method>* can be any of these values:

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
: to select a queue according to the IP type of service that is assigned to packets by some applications. The service type is a field in the IP datagram header that can be set with a **setsockopt** call. Table 28 shows how the qeth device driver maps service types to the available queues:

Table 28. IP service types and queue assignment for type of service queueing

Service type	Queue
Low latency	0
High throughput	1
High reliability	2
Not important	3

no_prio_queueing
: causes the qeth device driver to use queue 2 for all packets. This 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 is equivalent to the default.

no_prio_queueing:3
: causes the qeth device driver to use queue 3 for all packets.

Example

To make a device 0xa110 use queueing by type of service issue:

```
# echo prio_queueing_tos > /sys/bus/ccwgroup/drivers/qeth/a110/priority_queueing
```

Specifying the number of inbound buffers

Before you start: The device must be offline while you specify the number of inbound buffers.

By default, the qeth device driver assigns 16 buffers for inbound traffic to each qeth group device. Depending on the amount of available storage and the amount of traffic, you can assign from 8 to 128 buffers.

Note: For Linux 2.4, this parameter was fixed at 128 buffers. With Linux 2.6, you only get 128 buffers if you set the buffer_count attribute to 128.

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 is:

- For an OSA-Express CHPID in QDIO mode or an OSA-Express CHPID in OSN mode: 64 KB
- For HiperSockets: depending on the HiperSockets CHPID definition, 16 KB, 24 KB, 40 KB, or 64 KB

Set the buffer_count attribute to the number of inbound buffers you want to assign. Issue a command of the form:

```
# echo <number> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/buffer_count
```

Example

In this example, 64 inbound buffers are assigned to device 0.0.a000.

```
# echo 64 > /sys/bus/ccwgroup/drivers/qeth/0.0.a000/buffer_count
```

Specifying the relative port number

Before you start:

- This section applies to adapters that show more than one port to Linux, physical or logical. These adapters are:
  - OSA-Express3 Gigabit Ethernet as of z10 and later systems.
  - OSA-Express3 1000Base-T Ethernet as of z10 and later systems.

  In all other cases only a single port is available.
- The device must be offline while you specify the relative port number.

The OSA-Express3 Gigabit Ethernet adapter and 1000Base-T Ethernet adapter introduced with z10 both provide two physical ports for a single CHPID. By default, the qeth group device uses port 0. To use a different port, issue a command of the form:

```
# 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.

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a000/portno
```

Finding out the type of your network adapter

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. Issue a command of the form:

```
# cat /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/card_type
```

The card_type attribute gives information about both the type of network adaptor and also about the type of network link (if applicable) available at the card's ports. See Table 29 for details.

Table 29. Possible values of card_type and what they mean

Value of card_type	Adapter type	Link type
OSD_10GIG	OSA card in OSD mode	10 Gigabit Ethernet
OSD_1000		Gigabit Ethernet
OSD_100		Fast Ethernet
OSD_GbE_LANE		Gigabit Ethernet, LAN Emulation
OSD_FE_LANE		Fast Ethernet, LAN Emulation
OSD_Express		Unknown
OSN	OSA for NCP	ESCON/CDLC bridge or N/A
OSM	OSA-Express for Unified Resource Manager	1000BASE-T
OSX	OSA-Express for zBX	10 Gigabit Ethernet
HiperSockets	HiperSockets, CHPID type IQD	N/A
GuestLAN QDIO	Guest LAN based on OSA	N/A
GuestLAN Hiper	Guest LAN based on HiperSockets	N/A
Unknown	Other	

Example

To find the card_type of a device 0.0.a100 issue:

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a100/card_type
OSD_100
```

Setting a device online or offline

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:

```
# 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 113).

Setting a device offline closes this network device. If IPv6 is active, you will loose any IPv6 addresses set for this device. After setting the device online, you can restore lost IPv6 addresses only by issuing the "ifconfig" or "ip" commands again.

Example

To set a qeth device with bus ID 0.0.a100 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a100/online
```

To set the same device offline issue:

```
# 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. Use the **lsqeth -p** command (see “lsqeth - List qeth based network devices” on page 405) to obtain a mapping for all qeth interfaces and devices.

Alternatively, you can use sysfs. To find out the interface name of a qeth group device for which you know the device bus-ID read 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

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a100/if_name
eth0
```

Finding out the bus ID of a qeth interface

Use the **lsqeth -p** command (see “lsqeth - List qeth based network devices” on page 405) to obtain a mapping for all qeth interfaces and devices.

Alternatively, you can use sysfs. For each network interface, there is a directory in sysfs under `/sys/class/net/`, for example, `/sys/class/net/eth0` for interface eth0. 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 eth0 issue, for example:

```
# readlink /sys/class/net/eth0/device
../../../devices/qeth/0.0.a100
```

In this example, eth0 corresponds to the device bus-ID 0.0.a100.

Activating an interface

Before you start:

- You need to know the interface name of the qeth group device (see “Finding out the interface name of a qeth group device”).

- You need to know the IP address you want to assign to the device.

The MTU size defaults to the correct settings for HiperSockets and to 1492 for OSA-Express CHPIDs in QDIO mode.

In most cases 1492 is 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 8992. Exceeding 1492 for regular frames or 8992 for jumbo frames might cause performance degradation. See *OSA-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 MTU sizes of 8 KB, 16 KB, 32 KB or 56 KB, respectively.

The MTU size defaults to the correct settings for both HiperSockets and OSA-Express CHPIDs in QDIO mode. As a result, you need not specify the MTU size when activating the interface.

Note that, on heavily loaded systems, MTU sizes exceeding 8 KB can lead to memory allocation failures for packets due to memory fragmentation. A symptom of this problem are messages of the form "order-N allocation failed" in the system log; in addition, network connections will drop packets, in extreme cases to the extent that the network is no longer usable.

As a workaround, use MTU sizes at most of 8 KB (minus header size), even if the network hardware allows larger sizes (for example, HiperSockets or 10 Gigabit Ethernet).

You activate or deactivate network devices with **ifconfig** or an equivalent command. For details of the **ifconfig** command refer to the **ifconfig** man page.

Examples

- This example activates a HiperSockets CHPID:

```
# ifconfig hsi0 192.168.100.10 netmask 255.255.255.0
```

- This example activates an OSA-Express CHPID in QDIO mode:

```
# ifconfig eth0 192.168.100.11 netmask 255.255.255.0 broadcast 192.168.100.255
```

  Or, using the default netmask and its corresponding broadcast address:

```
# ifconfig eth0 192.168.100.11
```

- This example reactivates an interface that had already been activated and subsequently deactivated:

```
# ifconfig eth0 up
```

- This example activates an OSA-Express2 CHPID defined as an OSN type CHPID for OSA NCP:

```
# ifconfig osn0 up
```

Confirming that an IP address has been set under layer 3

The Linux network stack design does not allow feedback about IP address changes. If **ifconfig** or an equivalent command fails to set an IP address on an OSA-Express network CHPID, a query with **ifconfig** 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 display these messages with **dmesg**. In Red Hat Enterprise Linux 6 you can also find the messages in `/var/log/messages`.

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.

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.

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 122 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 125 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 126 for details.

You can use the **qethconf** command (see “qethconf - Configure qeth devices” on page 431) 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 **ifconfig** or an equivalent command or by setting the network device offline. While setting a device offline involves actions on the attached device, deactivating only stops the interface logically within Linux.

To deactivate an interface with **ifconfig**, Issue a command of the form:

```
# ifconfig <interface_name> down
```

Example

To deactivate eth0 issue:

```
# ifconfig eth0 down
```

Recovering a device

You can use the recover attribute of a qeth group device to recover it in case of failure. For example, error messages in `/var/log/messages` might inform you of a malfunctioning device. Issue a command of the form:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/recover
```

Example

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a100/recover
```

Isolating data connections

You can restrict communications between operating system instances that share the same OSA port on an OSA adapter.

A Linux instance can configure the OSA adapter to prevent any direct package exchange between itself and other operating system instances that share the same OSA adapter. This ensures a higher degree of isolation than VLANs.

For example, if 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. The two other instances could still communicate directly through the OSA adapter without the external switch in the network path (see Figure 20).

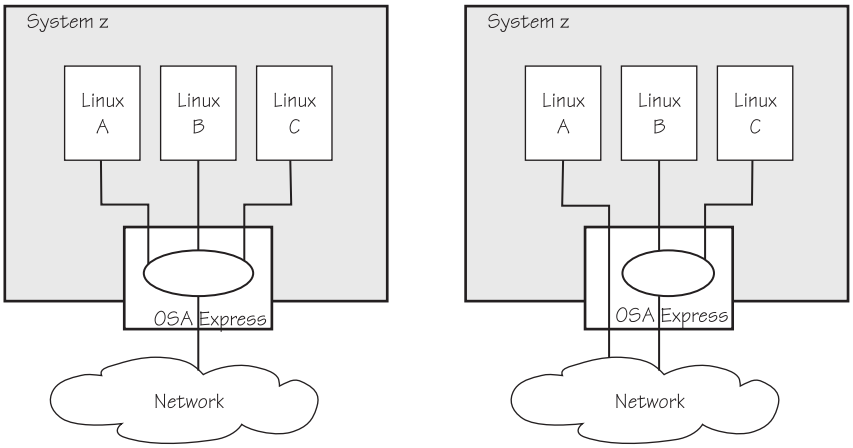


Figure 20. Linux instances sharing a port on an OSA adapter (left), Linux A is isolated (right).

Before you begin:

- Data isolation is available with the following OSA cards with the respective microcode level (MCL):
  - OSA-Express2 on z9, MCL G40946.008
  - OSA-Express2 on z10, MCL N10953.002
  - OSA-Express3 on z10, MCL N10959.004 and N10967.055

  One such adapter is required and must be configured as an OSA adapter for the operating system instance.

QDIO data connection isolation is configured as a policy. The policy can take the following values:

1. none: No isolation. This is the default.
2. ISOLATION_DROP: 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.
3. ISOLATION_FORWARD: This policy results in a similar behavior as ISOLATION_DROP. The only difference is that packets are forwarded to the connected switch instead of being dropped. At the time of this writing, none of the available switches implements support for this policy.

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.

The policy is implemented as a sysfs attribute called isolation. Note that the attribute appears in sysfs regardless of whether the hardware supports the feature.

Examples:

- To check the current isolation policy:

```
# cat /sys/devices/qeth/0.0.f5f0/isolation
```

- To set the isolation policy to ISOLATION_DROP:

```
# echo "drop" > /sys/devices/qeth/0.0.f5f0/isolation
```

- To set the isolation policy to ISOLATION_FORWARD:

```
# echo "forward" > /sys/devices/qeth/0.0.f5f0/isolation
```

- To set the isolation policy to none:

```
# echo "none" > /sys/devices/qeth/0.0.f5f0/isolation
```

Refer to *z/VM Connectivity*, SC24-6174 for information about how to set up data connection isolation on a VSWITCH.

Starting and stopping collection of QETH performance statistics

For QETH performance statistics there is a device group attribute called `/sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/performance_stats`.

This attribute is initially set to 0, that is QETH performance data is not collected. To start collection for a specific QETH device, write 1 to the attribute, for example:

```
echo 1 > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/performance_stats
```

To stop collection write 0 to the attribute, for example:

```
echo 0 > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/performance_stats
```

Stopping QETH performance data collection for a specific QETH device is accompanied by a reset of current statistic values to zero.

To display QETH performance statistics, use the **ethtool** command. Refer to the **ethtool** man page for details. The following example shows statistic and device driver information:

```
 # ethtool -S eth0
NIC statistics:
     rx skbs: 86
     rx buffers: 85
     tx skbs: 86
     tx buffers: 86
     tx skbs no packing: 86
     tx buffers no packing: 86
     tx skbs packing: 0
     tx buffers packing: 0
     tx sg skbs: 0
     tx sg frags: 0
     rx sg skbs: 0
     rx sg frags: 0
     rx sg page allocs: 0
     tx large kbytes: 0
     tx large count: 0
     tx pk state ch n->p: 0
     tx pk state ch p->n: 0
     tx pk watermark low: 2
     tx pk watermark high: 5
     queue 0 buffer usage: 0
     queue 1 buffer usage: 0
     queue 2 buffer usage: 0
     queue 3 buffer usage: 0
     rx handler time: 856
     rx handler count: 84
     rx do_QDIO time: 16
     rx do_QDIO count: 11
     tx handler time: 330
     tx handler count: 87
     tx time: 1236
     tx count: 86
     tx do_QDIO time: 997
     tx do_QDIO count: 86

# ethtool -i eth0
driver: qeth_l3
version: 1.0
firmware-version: 087a
bus-info: 0.0.f5f0/0.0.f5f1/0.0.f5f2
```

To control QDIO performance statistics as well, see “Starting and stopping collection of QDIO performance statistics” on page 61.

Working with the qeth device driver in layer 3 mode

Setting up a Linux router

Before you start:

- A suitable hardware setup is in place that permits your Linux instance to act as a router.
- The Linux instance is set up as a router.

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. 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 30. 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

The values are explained in detail below.

An OSA-Express CHPID in QDIO mode honors 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 honors the following values, provided 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 135).

secondary_connector
to make your Linux instance a backup connection between a HiperSockets network and an external network (see “HiperSockets Network Concentrator” on page 135).

Both types of CHPIDs honor:

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 135.

no_router
is the default. You can use this value to reset a router setting to the default.

Note: 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, this can be done by issuing:

```
# sysctl -w net.ipv4.conf.all.forwarding=1
```

For IPv6, this can be done by issuing:

```
# sysctl -w net.ipv6.conf.all.forwarding=1
```

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 needs to be enabled for Linux in an LPAR or as a z/VM guest operating system to act as a router. In Red Hat Enterprise Linux 6 you can set IP forwarding permanently in `/etc/sysctl.conf` or dynamically with the sysctl command.

Mainframe configuration:

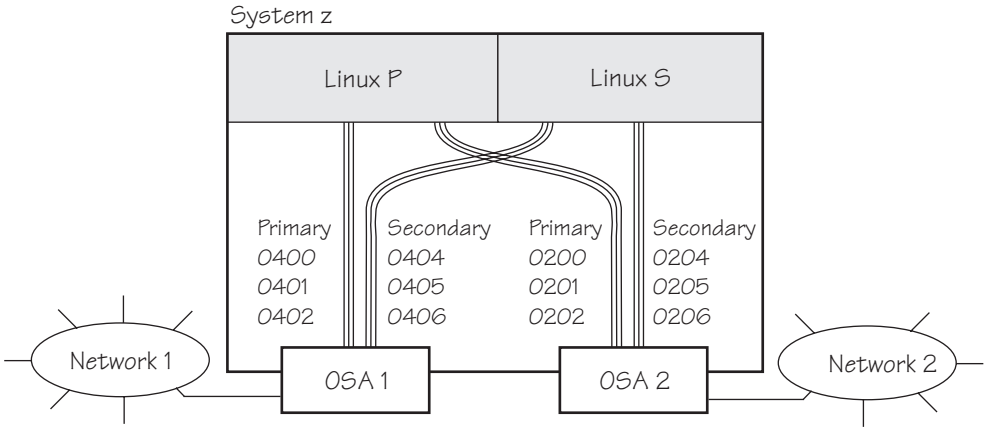


Figure 21. Mainframe configuration

It is assumed that both Linux instances are configured as routers in their respective LPARs or in z z/VM.

Linux P configuration:

To create the qeth group devices:

```
# 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:

```
# 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:

```
# 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:

```
# 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:

```
/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 135 for further examples.

Setting the checksumming method

A checksum is a form of redundancy check to protect the integrity of data. In general, checksumming is used for network data.

Before you start: The device must be offline while you set the checksumming method.

You can determine how checksumming is performed for incoming IP packages by setting a value for the checksumming attribute of your qeth device. Issue a command of the form:

```
# echo <method> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/checksumming
```

where *<method>* can be any of these values:

hw_checksumming
performs the checksumming in hardware if the CHPID is an OSA-Express CHPID in QDIO mode and your OSA adapter hardware supports checksumming.

If you set “hw_checksumming” for an adapter that does not support it or for a HiperSockets CHPID, the TCP/IP stack performs the checksumming instead of the adapter.

sw_checksumming
performs the checksumming in the TCP/IP stack. This is the default.

no_checksumming
suppresses checksumming.

Attention: Suppressing checksumming might jeopardize data integrity.

Examples

- To find out the checksumming setting for a device 0x1a10 read the checksumming attribute:

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.1a10/checksumming
sw_checksumming
```

- To enable hardware checksumming for a device 0x1a10 issue:

```
# echo hw_checksumming > /sys/bus/ccwgroup/drivers/qeth/0.0.1a10/checksumming
```

Faking broadcast capability

Before you start:

- This section applies to devices that do not support broadcast only.
- The device must be offline while you enable faking broadcasts.

For devices that support broadcast, the broadcast capability is enabled automatically.

To find out if a device supports broadcasting, use ifconfig. If the resulting list shows the BROADCAST flag the device supports broadcast. This example shows that the device eth0 supports broadcast:

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:09:6B:1A:9A:B7
          inet addr:9.152.25.187  Bcast:9.152.27.255  Mask:255.255.252.0
          inet6 addr: fe80::9:6b00:af1a:9ab7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1492  Metric:1
          RX packets:107792 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12176 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:29753474 (28.3 MiB)  TX bytes:1979603 (1.8 MiB)
```

Some processes, for example, the *gated* routing daemon, require the devices' broadcast capable flag to be set in the Linux network stack. To set this 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:

```
# 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 has broadcast capability.

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a100/fake_broadcast
```

Taking over IP addresses

This section describes how to configure for IP takeover if the layer2 option (see “MAC headers in layer 2 mode” on page 100) is not enabled. If you have enabled the layer2 option, you can configure for IP takeover as you would in a distributed server environment.

Taking over an IP address overrides any previous allocation of this address to another LPAR. If another LPAR on the same CHPID has already registered for that IP address, this association is removed.

An OSA-Express CHPID in QDIO mode can take over IP addresses from any System 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. 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. 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 431 for more details on 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 device hsi0.

```
# qethconf ipa list
ipa add 192.168.10.0/24 hsi0
```

The following command adds a range of IP addresses that can be taken over by device eth0.

```
# qethconf ipa add 192.168.11.0/24 eth0
qethconf: Added 192.168.11.0/24 to /sys/class/net/eth0/device/ipa_takeover/add4.
qethconf: Use "qethconf ipa list" to check for the result
```

Listing the activated IP addresses now shows both ranges of addresses.

```
# qethconf ipa list
ipa add 192.168.10.0/24 hsi0
ipa add 192.168.11.0/24 eth0
```

The following command deletes the range of IP addresses that can be taken over by device eth0.

```
# qethconf ipa del 192.168.11.0/24 eth0
qethconf: Deleted 192.168.11.0/24 from /sys/class/net/eth0/device/ipa_takeover/del4.
qethconf: Use "qethconf ipa list" to check for the result
```

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 eth2:

```
qethconf ipa add fec0::/64 eth2
qethconf: Added fec0:0000:0000:0000:0000:0000:0000:0000/64 to
          sysfs entry /sys/class/net/eth2/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 eth2
```

The following command deletes the IPv6 address range that can be taken over by eth2:

```
qethconf ipa del fec0:0000:0000:0000:0000:0000:0000:0000/64 eth2:
qethconf: Deleted fec0:0000:0000:0000:0000:0000:0000:0000/64 from
          sysfs entry /sys/class/net/eth2/device/ipa_takeover/del6.
qethconf: For verification please use "qethconf ipa list"
```

Stage 3: Issuing a command to take over the address

Before you start:

- 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 123).
- 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 123).

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.

IPv4 example: To make a device hsi0 take over IP address 192.168.10.22 issue:

```
# ip addr add 192.168.10.22 dev hsi0
```

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 device hsi0 take over IP address 192.168.10.22 if hsi0 is already configured to have IP address 192.168.10.22 issue:

```
# ip addr del 192.168.10.22 dev hsi0
# ip addr add 192.168.10.22 dev hsi0
```

IPv6 example: To make a device eth2 take over fec0::111:25ff:febd:d9da/64 issue:

```
ip addr add fec0::111:25ff:febd:d9da/64 nodad dev eth2
```

For IPv6, setting the **nodad** (no duplicate address detection) option ensures that the eth2 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 eth2 take over IP address fec0::111:25ff:febd:d9da/64 when eth2 is already configured to have that particular IP address issue:

```
ip addr del fec0::111:25ff:febd:d9da/64 nodad dev eth2
ip addr add fec0::111:25ff:febd:d9da/64 nodad dev eth2
```

Be aware of the information in “Confirming that an IP address has been set under layer 3” on page 115 when using IP takeover.

Configuring a device for proxy ARP

This section describes how to configure for proxy ARP if the layer2 option (see “MAC headers in layer 2 mode” on page 100) is not enabled. If you have enabled the layer2 option, you can configure for proxy ARP as you would in a distributed server environment.

Before you start: This section applies to qeth group devices that have been set up as routers only.

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 on proxy ARP, see

`www.sjdjweis.com/linux/proxyarp/`

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 431).

Be aware of the information in “Confirming that an IP address has been set under layer 3” on page 115 when working with proxy ARP.

Example

Figure 22 shows an environment where proxy ARP is used.

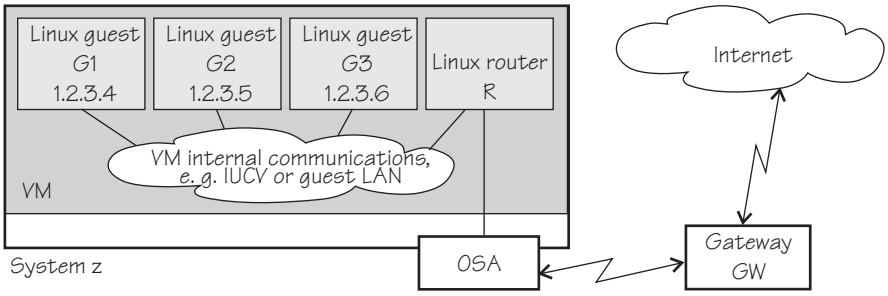


Figure 22. Example of proxy ARP usage

G1, G2, and G3 are Linux guests (connected, for example, through a guest LAN to a Linux router R), reached from GW (or the outside world) via R. R is the ARP proxy for G1, G2, and G3. That is, R agrees to take care of packets 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 eth0
qethconf: Added 1.2.3.4 to /sys/class/net/eth0/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 eth0
parp add 1.2.3.5 eth0
parp add 1.2.3.6 eth0
```

Configuring a device for virtual IP address (VIPA)

This section describes how to configure for VIPA if the layer2 option (see “MAC headers in layer 2 mode” on page 100) is not enabled. If you have enabled the layer2 option, you can configure for VIPA as you would in a distributed server environment.

Before you start:

- This section does not apply to HiperSockets.

System z use 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 431).

For an example of how to use VIPA, see “Scenario: VIPA – minimize outage due to adapter failure.”

Be aware of “Confirming that an IP address has been set under layer 3” on page 115 when working with VIPAs.

Scenario: VIPA – minimize outage due to adapter failure

This chapter describes how to use

- Standard VIPA
- Source VIPA (version 2.0.0 and later)

Using VIPA you can assign IP addresses that are not associated with a particular adapter. This minimizes outage caused by adapter failure. Standard VIPA is usually sufficient for applications, such as web servers, that do *not* open connections to other nodes. Source VIPA is used for applications that open connections to other nodes. Source VIPA Extensions enable you to work with multiple VIPAs per destination in order to achieve multipath load balancing.

Notes:

1. See the information in “Confirming that an IP address has been set under layer 3” on page 115 concerning possible failure when setting IP addresses for OSA-Express features in QDIO mode (qeth driver).
2. The configuration file layout for Source VIPA has changed since the 1.x versions. In the 2.0.0 version a *policy* is included. For details see the README and the man pages provided with the package.

Standard VIPA

Purpose

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.

Usage

These are the main steps you must follow 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 above.
3. Set up *routes* to the virtual IP address, on clients or gateways. To do so, you can use either:
   - Static routing (shown in the example of Figure 23 on page 128).
   - Dynamic routing. For details of how to configure routes, you must refer to the documentation delivered with your *routing daemon* (for example, `zebra` or `gated`).

If outage of an adapter occurs, you must *switch adapters*.

- To do so under static routing, you should:
  1. Delete the route that was set previously.
  2. Create an alternative route to the virtual IP address.
- To do so under dynamic routing, you should refer to the documentation delivered with your *routing daemon* for details.

Example

This example assumes static routing is being used, and shows you how to:

1. Configure VIPA under static routing.
2. Switch adapters when an adapter outage occurs.

Figure 23 shows the network adapter configuration used in the example.

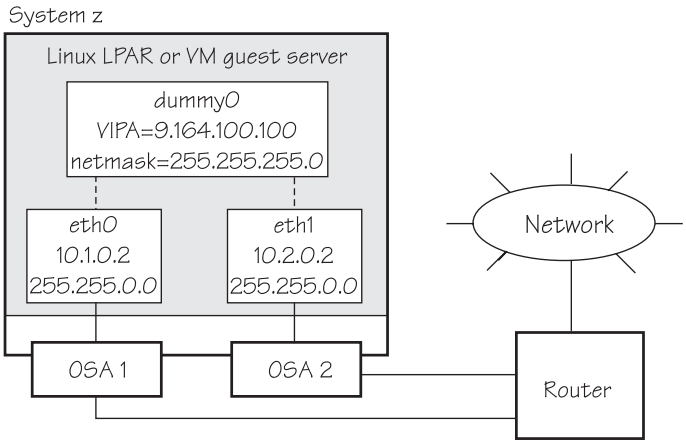


Figure 23. Example of using Virtual IP Address (VIPA)

1. Define the real interfaces

```
[server]# ifconfig eth0 10.1.0.2 netmask 255.255.0.0
[server]# ifconfig eth1 10.2.0.2 netmask 255.255.0.0
```

2. Ensure that the dummy module has been loaded. If necessary, load it by issuing:

```
[server]# modprobe dummy
```

3. Create a dummy interface with a virtual IP address 9.164.100.100 and a netmask 255.255.255.0:

```
 [server]# ifconfig dummy0 9.164.100.100 netmask 255.255.255.0
```

4. Enable the network devices for this VIPA so that it accepts packets for this IP address.

```
 [server]# qethconf vipa add 9.164.100.100 eth0
qethconf: Added 9.164.100.100 to /sys/class/net/eth0/device/vipa/add4.
qethconf: Use "qethconf vipa list" to check for the result
 [server]# qethconf vipa add 9.164.100.100 eth1
qethconf: Added 9.164.100.100 to /sys/class/net/eth1/device/vipa/add4.
qethconf: Use "qethconf vipa list" to check for the result
```

For IPv6, the address is specified in IPv6 format:

```
[server]# qethconf vipa add 2002000000000000000000000012345678 eth0
qethconf: Added 2002000000000000000000000012355678 to /sys/class/net/eth0/device/vipa/add6.
qethconf: Use "qethconf vipa list" to check for the result
[server]# qethconf vipa add 2002000000000000000000000012355678 eth1
qethconf: Added 2002000000000000000000000012355678 to /sys/class/net/eth1/device/vipa/add6.
qethconf: Use "qethconf vipa list" to check for the result
```

5. Ensure that the addresses have been set:

```
[server]# qethconf vipa list
vipa add 9.164.100.100 eth0
vipa add 9.164.100.100 eth1
```

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 via the gateway with address `10.1.0.2`.

```
[router]# route add -host 9.164.100.100 gw 10.1.0.2
```

Now we assume an *adapter outage* occurs. We must therefore:

1. Delete the previously-created route.

```
[router]# route delete -host 9.164.100.100
```

2. Create the alternative route to the virtual IP address.

```
[router]# route add -host 9.164.100.100 gw 10.2.0.2
```

Source VIPA

Purpose

Source VIPA is particularly suitable for high-performance environments. It selects one source address out of a range of source addresses when it replaces the source address of a socket. The reason for using several source addresses lies in the inability of some operating system kernels to do load balancing among several connections with the same source and destination address over several interfaces.

To achieve load balancing, a *policy* has to be selected in the *policy* section of the configuration file of Source VIPA (/etc/src_vipa.conf). This policy section also allows to specify several source addresses used for one destination. Source VIPA then applies the source address selection according to the rules of the policy selected in the configuration file.

This Source VIPA solution does not affect kernel stability. Source VIPA is controlled by a configuration file containing flexible rules for when to use Source VIPA based on destination IP address ranges.

Note: This implementation of Source VIPA applies to IPv4 only.

Usage

Installation: Source VIPA is delivered as part of the s390utils package. Install the package as usual.

Configuration: With Source VIPA version 2.0.0 the configuration file has changed: the policy section was added. The default configuration file is /etc/src_vipa.conf.

/etc/src_vipa.conf or the file pointed to by the environment variable SRC_VIPA_CONFIG_FILE, contains lines such as the following:

```
# comment
D1.D2.D3.D4/MASK POLICY S1.S2.S3.S4 [T1.T2.T3.T4 [...]]
.INADDR_ANY P1-P2 POLICY S1.S2.S3.S4 [T1.T2.T3.T4 [...]]
.INADDR_ANY P POLICY S1.S2.S3.S4 [T1.T2.T3.T4 [...]]
```

`D1.D2.D3.D4/MASK` specifies a range of destination addresses and the number of bits set in the subnet mask (MASK). As soon as a socket is opened and connected to these destination addresses and the application does not do an explicit bind to a source address, Source VIPA does a bind to one of the source addresses specified (S, T, [...]) using the policy selected in the configuration file to distribute the source addresses. See the policy section below for available load distribution policies. Instead of IP addresses in dotted notation, hostnames can also be used and will be resolved using DNS.

`.INADDR_ANY P1-P2 POLICY S1.S2.S3.S4` or `.INADDR_ANY P POLICY S1.S2.S3.S4` causes bind calls with `.INADDR_ANY` as a local address to be intercepted if the port the socket is bound to is between P1 and P2 (inclusive). In this case, `.INADDR_ANY` will be replaced by one of the source addresses specified (S, T, [...]), which can be 0.0.0.0.

All `.INADDR_ANY` statements will be read and evaluated in order of appearance. This means that multiple `.INADDR_ANY` statements can be used to have bind calls intercepted for every port outside a certain range. This is useful, for example, for `rlogin`, which uses the `bind` command to bind to a local port but with `.INADDR_ANY` as a source address to use automatic source address selection. See “Policies” below for available load distribution policies.

The default behavior for all ports is that the kind of bind calls will not be modified.

Policies: With Source VIPA Extensions you provide a range of dummy source addresses for replacing the source addresses of a socket. The policy selected determines which method is used for selecting the source addresses from the range of dummy addresses..

onevipa
: Only the first address of all source addresses specified is used as source address.

random
: The source address used is selected randomly from all the specified source addresses.

llr (local round robin)
: The source address used is selected in a round robin manner from all the specified source addresses. The round robin takes place on a per-invocation base: each process is assigned the source addresses round robin independently from other processes.

rr:ABC
: Stands for round robin and implements a global round robin over all Source VIPA instances sharing the same configuration file. All processes using Source VIPA access an IPC shared memory segment to fulfil a global round robin algorithm. This shared memory segment is destroyed when the last running Source VIPA ends. However, if this process does not end gracefully (for example, is ended by a `kill` command), the shared memory segment (size: 4 bytes) can stay in the memory until it is removed by `ipcrm`. The tool `ipcs` can be used to display all IPC resources and to get the key or id used for `ipcrm`. ABC are UNIX permissions in octal writing (for example, 700) that are used to create the shared memory segment. This permission mask

should be as restrictive as possible. A process having access to this mask can cause an imbalance of the round robin distribution in the worst case.

lc Attempts to balance the number of connections per source address. This policy always associates the socket with the VIPA that is least in use. If the policy cannot be parsed correctly, the policy is set to round robin per default.

Enabling an application: The command:

```
src_vipa.sh <application and parameters>
```

enables the Source VIPA functionality for the application. The configuration file is read once the application is started. It is also possible to change the starter script and run multiple applications using different Source VIPA settings in separate files. For this, a SRC_VIPA_CONFIG_FILE environment variable pointing to the separate files has to be defined and exported prior to invoking the respective application.

Notes:

1. LD_PRELOAD security prevents `setuid` executables to be run under Source VIPA; programs of this kind can only be run when the real UID is 0. The ping utility is usually installed with `setuid` permissions.
2. The maximum number of VIPAs per destination is currently defined as 8.

Example

Figure 24 shows a configuration where two applications with VIPA `9.164.100.100` and `9.164.100.200` are to be set up for Source VIPA with a local round robin policy.

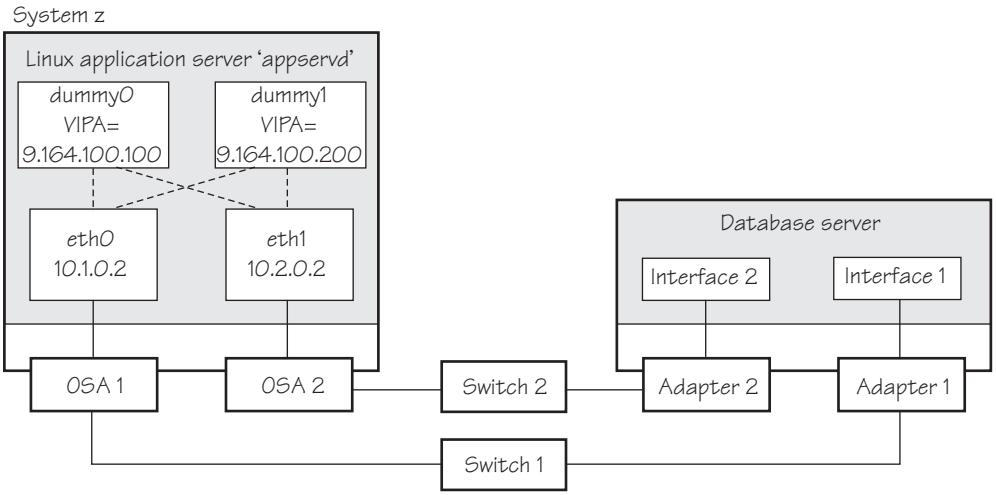


Figure 24. Example of using source VIPA

The required entry in the Source VIPA configuration file is:

```
9.0.0.0/8 lrr 9.164.100.100 9.164.100.200
```

Scenario: Virtual LAN (VLAN) support

VLAN technology works according to IEEE Standard 802.1Q by logically segmenting the network into different broadcast domains so that packets are switched only between ports designated for the same VLAN. By containing traffic originating on a particular LAN to other LANs within the same VLAN, switched

virtual networks avoid wasting bandwidth, a drawback inherent in traditional bridged/switched networks where packets are often forwarded to LANs that do not require them.

Introduction to VLANs

VLANs increase traffic flow and reduce overhead by allowing you to 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 using routers. Although routers add latency by delaying transmission of data while using more of the data packet to determine destinations, they are preferable to building a single broadcast domain, which could easily be flooded with traffic.

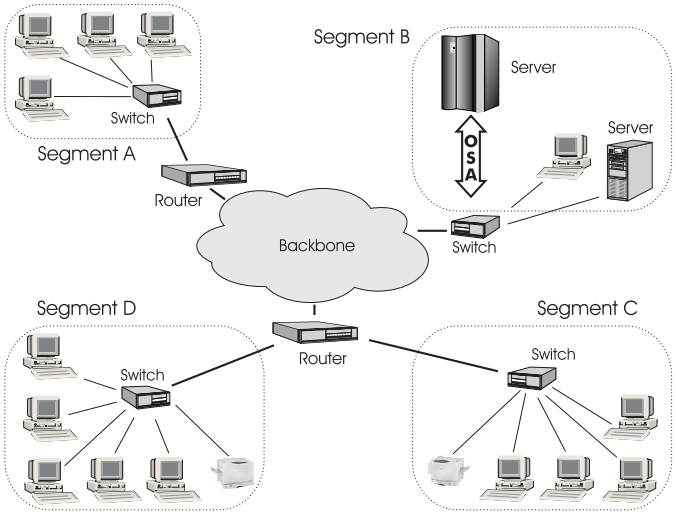


Figure 25. Conventional routed network

By organizing the network into VLANs through the use of Ethernet switches, distinct broadcast domains can be maintained without the latency 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.

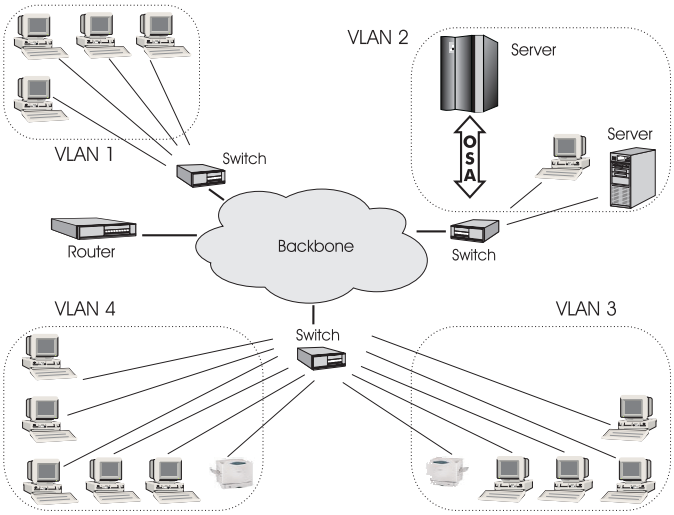


Figure 26. Switched VLAN network

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 reduces traffic both inside the domain and outside, on the rest of the network.

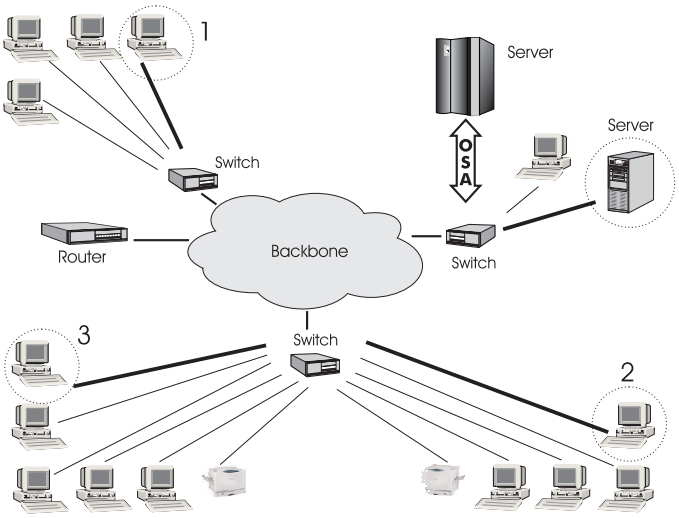


Figure 27. VLAN network organized for traffic flow

Configuring VLAN devices

VLANs are configured using the **vconfig** command. Refer to the **vconfig** man page for details.

Information on the current VLAN configuration is available by listing the files in

`/proc/net/vlan/*`

with `cat` or `more`. For example:

```
bash-2.04# cat /proc/net/vlan/config
VLAN Dev name    | VLAN ID
Name-Type: VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD  bad_proto_recvd: 0
eth2.100       | 100  | eth2
eth2.200       | 200  | eth2
eth2.300       | 300  | eth2
bash-2.04# cat /proc/net/vlan/eth2.300
eth2.300  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: eth2
INGRESS priority mappings: 0:0  1:0  2:0  3:0  4:0  5:0  6:0 7:0
EGRESS  priority Mappings:
bash-2.04#
```

Examples

VLANs are allocated in an existing interface representing a physical Ethernet LAN. The following example creates two VLANs, one with ID 3 and one with ID 5.

```
ifconfig eth1 9.164.160.23 netmask 255.255.224.0 up
vconfig add eth1 3
vconfig add eth1 5
```

The vconfig commands have added interfaces "eth1.3" and "eth1.5", which you can then configure:

```
ifconfig eth1.3 1.2.3.4 netmask 255.255.255.0 up
ifconfig eth1.5 10.100.2.3 netmask 255.255.0.0 up
```

The traffic that flows out of eth1.3 will be in the VLAN with ID=3 (and will not be received by other stacks that listen to VLANs with ID=4).

The internal routing table will ensure that every packet to 1.2.3.x goes out via eth1.3 and everything to 10.100.x.x via eth1.5. Traffic to 9.164.1xx.x will flow through eth1 (without a VLAN tag).

To remove one of the VLAN interfaces:

```
ifconfig eth1.3 down
vconfig rem eth1.3
```

The following example illustrates the definition and connectivity test for a VLAN comprising five different Linux systems (two LPARs, two z/VM guest virtual machines, and one x86 system), each connected to a physical Ethernet LAN through eth1:

```
(LINUX1: LPAR 64bit)
     vconfig add eth1 5
     ifconfig eth1.5 10.100.100.1 broadcast 10.100.100.255 netmask 255.255.255.0 up


(LINUX2: LPAR 31bit)
     vconfig add eth1 5
     ifconfig eth1.5 10.100.100.2 broadcast 10.100.100.255 netmask 255.255.255.0 up


(LINUX3: VM Guest 64bit)
     vconfig add eth1 5
     ifconfig eth1.5 10.100.100.3 broadcast 10.100.100.255 netmask 255.255.255.0 up
```

```
(LINUX4: VM Guest 31bit)
     vconfig add eth1 5
     ifconfig eth1.5 10.100.100.4 broadcast 10.100.100.255 netmask 255.255.255.0 up


(LINUX5: x86)
     vconfig add eth1 5
     ifconfig eth1.5 10.100.100.5 broadcast 10.100.100.255 netmask 255.255.255.0 up

Test the connections:

     ping 10.100.100.[1 - 5]          // Unicast-PING
     ping -I eth1.5 224.0.0.1         // Multicast-PING
     ping -b 10.100.100.255           // Broadcast-PING
```

HiperSockets Network Concentrator

This section describes how to configure a HiperSockets Network Concentrator on a QETH device in layer 3 mode.

Before you start: This section applies 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 appear as if they were directly connected to the LAN. This helps to reduce the complexity of network topologies resulting from server consolidation. HiperSockets Network Concentrator allows to migrate systems from the LAN into a System z Server environment, or systems connected by a different HiperSockets Network Concentrator into a System z Server environment, without changing the network setup. 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. This 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 sets them as Proxy ARP entries (see “Configuring a device for proxy ARP” on page 125) 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” on page 118).
- All HiperSockets interfaces 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. This can be achieved 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 has to be established via the OSA interface. To achieve this, the IP address 0.0.0.0 can be assigned to the HiperSockets interface while an address used in the HiperSockets Network Concentrator IP subnet is to be assigned to the OSA interface. This sets the network routes up 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. This 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 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 executed 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. In order to do this it needs to 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 setup as

secondary_connector. A secondary_connector takes over the connecting functionality, 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 not fitting in the link's MTU are discarded as there is no IP fragmentation for multicast bridging. Warnings are printed to `/var/log/messages` or 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 6 this creates entries in `/var/log/messages`.
- Registering all internal addresses with the OSA adapter can take several seconds for each address.
- To shut down the HiperSockets Network Concentrator functionality, simply issue `killall ip_watcher.pl`. This removes all routing table and Proxy ARP entries added while using HiperSockets Network Concentrator.

Notes

- With the current OSA and HiperSockets hardware design, broadcast packets that are sent out of an interface are echoed back by the hardware of the originating system. This makes it impossible to bridge broadcast traffic without causing bridging loops. Therefore, broadcast bridging is currently disabled.
- 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

Figure 28 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.

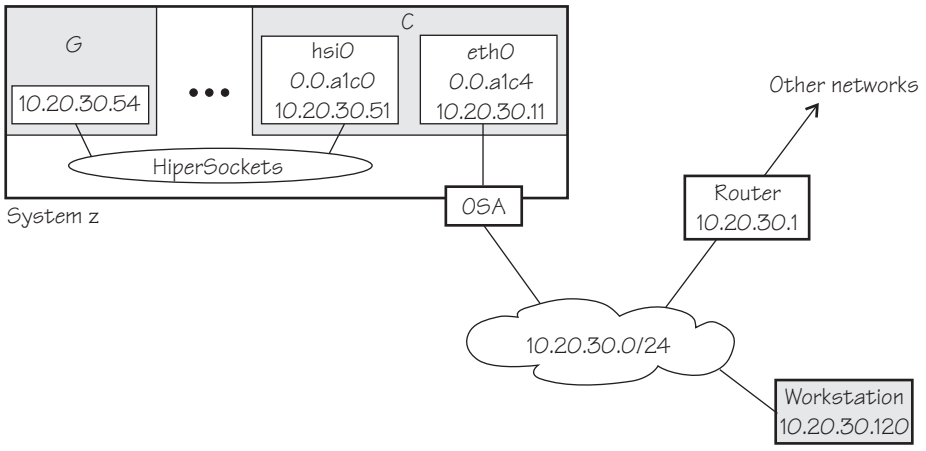


Figure 28. HiperSockets network concentrator setup

Setup for the network concentrator C:

The HiperSockets interface hsi0 (device bus-ID 0.0.a1c0) has IP address 10.20.30.51, and the netmask is 255.255.255.0. 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 eth0 (with device bus-ID 0.0.a1c4) has IP address 10.20.30.11, and the netmask is 255.255.255.0. 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:

```
# route del -net 10.20.30.0 netmask 255.255.255.0 dev hsi0
```

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, and the netmask is 255.255.255.0. 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, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

Figure 29 shows the example of Figure 28 on page 137 with an additional mainframe. On the second mainframe a Linux instance D acts as a HiperSockets network concentrator.

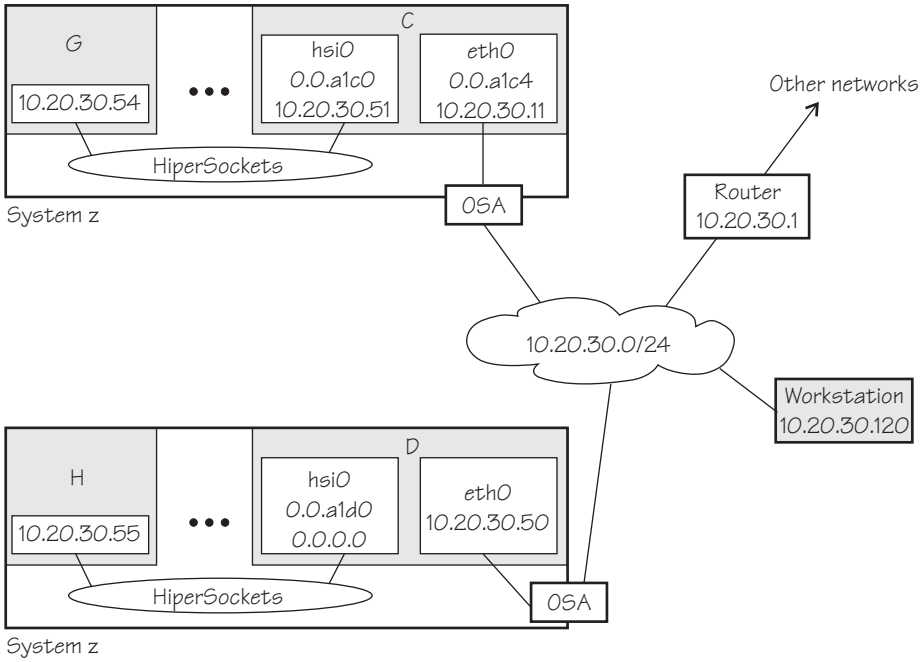


Figure 29. Expanded HiperSockets network concentrator setup

The configuration of C, G, and the workstation remain the same as for Figure 28 on page 137.

Setup for the network concentrator D:

The HiperSockets interface hsi0 has IP address 0.0.0.0.

Assuming that the device bus-ID of the HiperSockets interface is 0.0.a1d0, issue:

```
# echo primary_connector > /sys/bus/ccwgroup/drivers/qeth/0.0.a1d0/route4
```

The OSA-Express CHPID in QDIO mode interface eth0 has IP address 10.20.30.50, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

D is not configured as a multicast router, it therefor only forwards unicast packets.

To enable IP forwarding issue:

```
# sysctl -w net.ipv4.ip_forward=1
```

Tip: See *Red Hat Enterprise Linux 6 Installation Guide* for information about how to use configuration files to automatically enable IP forwarding when booting.

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, and the netmask is 255.255.255.0. 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 (see "MAC headers in layer 3 mode" on page 101). 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 have to be taken into account:

- DHCP using OSA-Express adapters in QDIO mode
- DHCP in a z/VM guest LAN

For information on setting up DHCP for a Linux instance in a z/VM guest LAN environment, refer to Redpaper *Linux on IBM eServer™ zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP-3596 at

`www.ibm.com/redbooks/`

This book discusses *dhclient* and *dhcp* as 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 System 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 same 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 ETHEREAL. 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. 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 normal network interface.

Before you start:

- You need SE authorization for the analyzing partition and the partitions to be analyzed.

  Tip: SE authorization changes for the HiperSockets network traffic analyzer require recreating the device by un-grouping and regrouping (see “Removing a qeth group device” on page 108 and “Creating a qeth group device” on page 107). Do any authorization changes before configuring the NTA device.
- You need a traffic dumping tool such as tcpdump.
- You need a mainframe system that supports HiperSockets network traffic analyzer. HiperSockets network traffic analyzer became available for z10 in March 2010.

Linux setup:

Ensure that the qeth device driver module has been loaded.

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 hsi0 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 455. 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 hsi0 up
   ```

3. Switch the interface into promiscuous mode:

```
# tcpdump -i hsi0
```

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:

```
# ifconfig hsi0 | grep "RX packets"
RX packets:6789 errors:0 dropped:5 overruns:0 frame:0

# tcpdump -i hsi0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on hsi1, 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 for guest LANs that are defined through a z/VM virtual switch and for other types of z/VM guest LANs. 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 general information on VLAN and z/VM virtual switches, see *Linux on IBM eServer zSeries and S/390: VSWITCH and VLAN Features of z/VM 4.4*, REDP-3719 at

`www.ibm.com/redbooks/`

Before you start:

- 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 operating system of the same z/VM instance as the guest LAN you want to investigate.

Linux setup:

Ensure that the qeth device driver has been loaded.

z/VM setup:

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 guest LAN is defined through a z/VM virtual switch, perform the following steps from a CMS session on your z/VM system:

1. Check if the z/VM guest virtual machine already has the required 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 required.

2. If the output from step 1 on page 142 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.

3. Authorize the Linux guest 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 used in this section 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*, SC24-6175.

Chapter 9. OSA-Express SNMP subagent support

The OSA-Express Simple Network Management Protocol (SNMP) subagent (osasnmpd) supports management information bases (MIBs) for the following OSA-Express features in QDIO mode only:

- OSA-Express
  - Fast Ethernet
  - 1000Base-T Ethernet
  - Gigabit Ethernet
- OSA-Express2
  - Gigabit Ethernet
  - 10 Gigabit Ethernet
  - 1000Base-T Ethernet (as of System z9)
- OSA-Express3 (as of System z10)
  - Gigabit Ethernet
  - 10 Gigabit Ethernet
  - 1000Base-T Ethernet

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) and HiperSockets
- The osasnmpd subagent from s390-tools
- The net-snmp package delivered with Red Hat Enterprise Linux 6

What you need to know about osasnmpd

The osasnmpd subagent requires a master agent to be installed on a Linux system. You get the master agent from either the net-snmp or the ucd-snmp package. The subagent uses the Agent eXtensibility (AgentX) protocol to communicate with the master agent.

net-snmp/ucd-snmp is an Open Source project that is owned by the Open Source Development Network, Inc. (OSDN). For more information on net-snmp/ucd-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 on 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 need to download the appropriate MIB file for the LIC level (see "Downloading the IBM OSA-Express MIB" on page 147), but you do not need to update the subagent. Place the updated MIB file in a directory that is searched by the master agent.

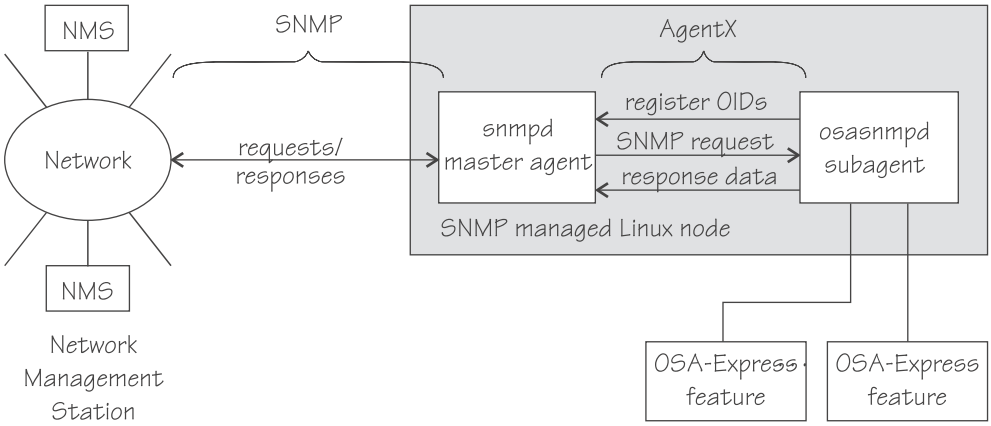


Figure 30. OSA-Express SNMP agent flow

Figure 30 illustrates the interaction between the snmpd master agent and the osasnmpd subagent.

Example: This example shows the processes running after the snmpd master agent and the osasnmpd subagent have been started. When you start osasnmpd, a daemon called osasnmpd-2.6 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-2.6
```

When the master agent receives an SNMP request for an OID that has been 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

This section describes the following setup tasks you need to perform if you want to use the osasnmpd subagent:

- Downloading the IBM OSA-Express MIB
- Configuring access control

Downloading the IBM OSA-Express MIB

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.

1. Go to `www.ibm.com/servers/resourcelink/`

   A user ID and password are required. You can apply for a user ID if you do not yet have one.
2. Sign in.
3. Select “Library” from the left-hand 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 149).

Result: You can now make the OID information from the MIB file available to the master agent. This allows you to use textual OIDs instead of numeric OIDs when using 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

During subagent startup or when network interfaces are added or removed, the subagent has to query OIDs from the interfaces group of the standard MIB-II. To start successfully, the subagent requires at least read access to the standard MIB-II on the local node.

This section gives an example of how you can use the snmpd.conf and snmp.conf configuration files to assign access rights using 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. Refer to the snmpd man page for a more information on how you can assign access rights to snmpd.

1. Refer to the Red Hat Enterprise Linux 6 documentation to find out where you need to 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.

   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 148.

   <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
   ```

   By default, AgentX support is compiled into the net-snmp master agent 5.1.x and, as of version 4.2.2, also into the ucd-snmp master agent.

8. Save and close snmpd.conf.
9. Open snmp.conf with your preferred text editor.
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 147.

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 148.

Example:

```
defVersion   2c
defCommunity osacom
```

These default specifications simplify issuing master agent commands.

13. Save and close snmp.conf.

Working with the osasnmpd subagent

This section describes the following tasks:

- Starting the osasnmpd subagent
- Checking the log file
- Issuing queries
- Stopping osasnmpd

Starting the osasnmpd subagent

After downloading osasnmpd package and setting up the osasnmpd subagent, start the subagent using the command:

```
# osasnmpd
```

The osasnmpd subagent, in turn, starts a daemon called osasnmpd-2.6.

For command options see the **osasnmpd** command manpage.

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 eth0
Initialization of OSA-E subagent successful...
```

Issuing queries

This section provides some examples of what SNMP queries might look like. For more comprehensive information on the master agent commands refer to 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 on which information is requested.

The query examples in this section gather information on an interface, eth0, for which the **lsqeth** (see “lsqeth - List qeth based network devices” on page 405) output looks like this:

```
# lsqeth eth0
Device name                     : eth0
---------------------------------------------
        card_type               : OSD_100
        cdev0                   : 0.0.f200
        cdev1                   : 0.0.f201
        cdev2                   : 0.0.f202
        chpid                   : 6B
        online                  : 1
        portname                : OSAPORT
        portno                  : 0
        route4                  : no
        route6                  : no
        checksumming            : sw checksumming
        state                   : UP (LAN ONLINE)
        priority_queueing       : always queue 0
        detach_state            : 0
        fake_ll                 : 0
        fake_broadcast          : 0
        buffer_count            : 16
        add_hhlen               : 0
        layer2                  : 0
```

The CHPID for the eth0 of our example is 0x6B.

- 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 = eth0
  ```

  Using this GET request you can see that eth0 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 our eth0 example. The example assumes that the community osacom has been authorized as described in “Configuring access control” on page 147.

If you have provided defaults for the SNMP version and the community (see step 12 on page 150), 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 somewhat 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. 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 have 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 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
# kill 729
```

Chapter 10. LAN channel station device driver

The LAN channel station device driver (LCS device driver) supports these Open Systems Adapters (OSA) features in non-QDIO mode:

Table 31. The LCS device driver supported OSA features

Feature	System z196 and z10	System z9
OSA-Express3		
1000Base-T Ethernet	Yes	No
OSA-Express2		
1000Base-T Ethernet	Yes	Yes
OSA-Express		
Fast Ethernet	No	Yes
1000Base-T Ethernet	No	Yes

Features

The LCS device driver supports the following devices and functions:

- Automatically detects an Ethernet connection
- Internet Protocol, version 4 (IPv4) only

What you should know about LCS

This section provides information about LCS group devices and interfaces.

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.

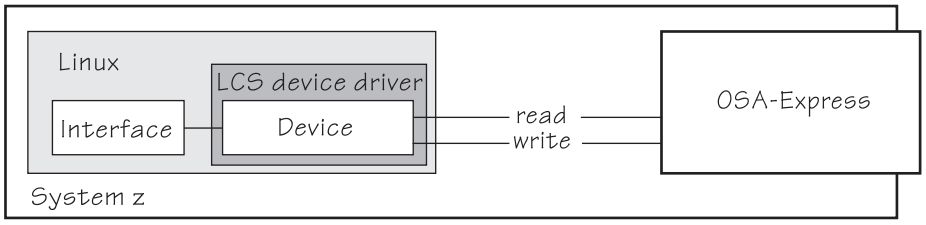


Figure 31. I/O subchannel interface

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.

LCS interface names

When an LCS group device is set online, the LCS device driver automatically assigns an interface name to it. The naming scheme uses the base name:

eth<*n*> for Ethernet features

where *<n>* is an integer that uniquely identifies the device. When the first device for a base name is set online it is assigned 0, the second is assigned 1, the third 2, and so on. Each base name is counted separately.

For example, the interface name of the first Ethernet feature that is set online is “eth0”, the second “eth1”, and so on.

The LCS device driver shares the name space for Ethernet interfaces with the qeth device driver. Each driver uses the name with the lowest free identifier *<n>*, regardless of which device driver occupies the other names. For example, if at the time the first LCS Ethernet feature is set online, there is already one qeth Ethernet feature online, the qeth feature is named “eth0” and the LCS feature is named “eth1”. See also “qeth interface names and device directories” on page 99.

Setting up the LCS device driver

There are no module parameters for the LCS device driver.

You need to 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 the LCS device driver

This section describes typical tasks that you need to perform when working with LCS devices.

- Creating an LCS group device
- Removing an LCS group device
- Specifying a timeout for LCS LAN commands
- Setting a device online or offline
- Activating and deactivating an interface
- Recovering a device

Most of these tasks involve writing to and reading from device attributes 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 an LCS device persistently is in *Red Hat Enterprise Linux 6 Installation Guide*. For a general discussion of network configuration files, see *Red Hat Enterprise Linux 6 Deployment Guide*.

Creating an LCS group device

Before you start: You need to know the device bus-IDs that correspond to the read and write subchannel of your OSA card as defined in the IOCDS of your mainframe.

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
```

Result: 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/cu3088/0.0.d000`

Removing an LCS group device

Before you start: The device must be set offline before you can remove it.

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

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 may take a longer time. The default is 5 s.

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 from 1 to 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 a device online or offline

To set an LCS group device online, set the online device group attribute to “1”. To set a 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
```

Setting a device online associates it with an interface name. Setting the device offline preserves the interface name.

Read `/var/log/messages` or issue **dmesg** to find out which interface name has been assigned. You will need to know the interface name to activate the network interface.

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 have found the correct interface name by reading the link.

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
# dmesg
...
 lcs: LCS device eth0 without IPv6 support
 lcs: LCS device eth0 with Multicast support
...
```

The interface name that has been assigned to the LCS group device in the example is eth0. To confirm that this is the correct name for our group device issue:

```
# readlink /sys/class/net/eth0/device
../../../devices/lcs/0.0.d000
```

To set the device offline issue:

```
# echo 0 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/online
```

Activating and deactivating an interface

Before you can activate an interface you need to have set the group device online and found out the interface name assigned by the LCS device driver (see “Setting a device online or offline”).

You activate or deactivate network devices with **ifconfig** or an equivalent command. For details of the **ifconfig** command refer to the **ifconfig** man page.

Examples

- This example activates an Ethernet interface:

  ```
  # ifconfig eth0 192.168.100.10 netmask 255.255.255.0
  ```

- This example deactivates the Ethernet interface:

  ```
  # ifconfig eth0 down
  ```

- This example reactivates an interface that had already been activated and subsequently deactivated:

```
# ifconfig eth0 up
```

Recovering a device

You can use the recover attribute of an LCS group device to recover it in case of failure. For example, error messages in `/var/log/messages` might inform you of a malfunctioning device. 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 11. CTCM device driver

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.

CTC connections are high-speed point-to-point connections between two operating system instances on System z.

Communications Server for Linux uses MPC connections to connect Red Hat Enterprise Linux 6 to VTAM® on traditional mainframe operating systems.

Deprecated connection type
CTC connections are deprecated. Do not use for new network setups.

Features

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.
- 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

This section provides information about CTCM group devices and the network interfaces that are created by the CTCM device driver.

CTCM group devices

The CTCM device driver requires two I/O subchannels for each interface, a read subchannel and a write subchannel (see Figure 32). The device bus-IDs that correspond to the two subchannels must be configured for control unit type 3088.

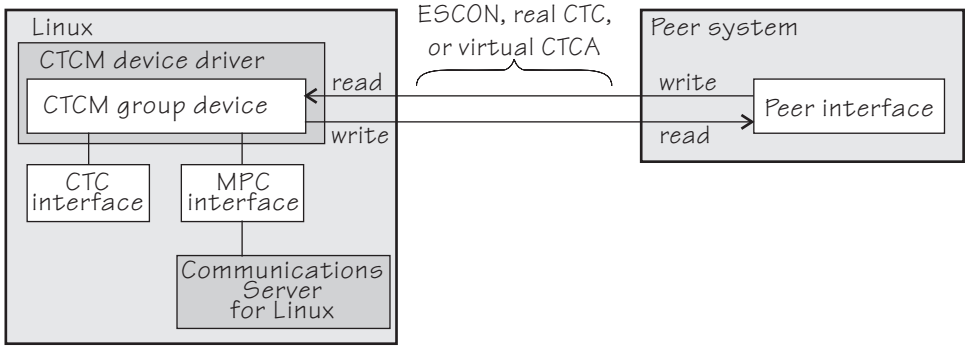


Figure 32. I/O subchannel interface

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.

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

This section applies to CTC interfaces only.

If your CTC connection is to a router or z/VM TCP/IP service machine, you can connect to an external network, see Figure 33.

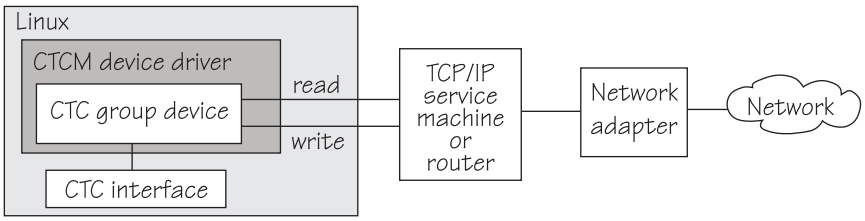


Figure 33. Network connection

Further information

For more information about Communications Server for Linux and on using MPC connections, go to `ibm.com/software/network/commserver/linux/`.

For more information about FICON, see Redpaper *FICON CTC Implementation*, REDP-0158.

Setting up the CTCM device driver

There are no module parameters for the CTCM device driver. You need to load the ctcm module before using it. Load it with the **modprobe** command to ensure that any other required modules are loaded:

```
# modprobe ctcm
```

Working with the CTCM device driver

This section describes typical tasks that you need to perform when working with CTCM devices.

- Creating a CTCM group device
- Removing a CTCM group device
- Displaying the channel type
- Setting the protocol
- Setting a device online or offline
- Setting the maximum buffer size (CTC only)
- Activating and deactivating a CTC interface (CTC only)
- Recovering a lost CTC connection (CTC only)

See the Communications Server for Linux documentation for information on how to configure and activate MPC interfaces.

Creating a CTCM group device

Before you start: You need to know the device bus-IDs that correspond to the local read and write subchannel of your CTCM connection as defined in your IOCDS.

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
```

Result: 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/cu3088/0.0.2000`

Removing a CTCM group device

Before you start: The device must be set offline before you can remove it.

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

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

Before you start: The device must be offline while you set the protocol.

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 OS/390®, or z/OS, for example, a z/VM TCP service machine. This is the default.

1 This protocol provides enhanced package checking for Linux peers.

3 This protocol provides for compatibility with OS/390 or z/OS peers.

4 This protocol provides for MPC connections to VTAM on traditional mainframe operating systems.

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

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
```

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 setting the group device back online, the interface name can change as described in “Interface names assigned by the CTCM device driver” on page 160.

Read `/var/log/messages` or issue **dmesg** to find out which interface name has been assigned to the group device. You will need to know the interface name to access the CTCM group device.

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 have found the correct interface name by reading the link.

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
# dmesg | fgrep "ch-0.0.2000"
mpc0: read: ch-0.0.2000, write: ch-0.0.2001, proto: 4
```

The interface name that has been assigned to the CTCM group device in the example is mpc0. To confirm that this is the correct name for our group device issue:

```
# readlink /sys/class/net/mpc0/device
../../../devices/cu3088/0.0.2000
```

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

Before you start:

- This section applies to CTC interfaces only. MPC interfaces automatically use the highest possible maximum buffer size.
- The device must be online when setting the buffer size.

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>*.

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

Before you start activating a CTC interface:

- This section applies to CTC interfaces only. For information about how to activate MPC interfaces see the Communications Server for Linux documentation.
- You need to know the interface name (see “Setting a device online or offline” on page 163).

Use **ifconfig** or an equivalent command to activate the interface:

Syntax for activating a CTC interface with the ifconfig command

```
►►─ifconfig─ <interface>─ <ip_address>─ pointopoint─ <peer_ip_address>──────►
                                                                              
►─┬─ mtu 32760 ──────────┬─┬─ up ─┬──────────────────────────────────────────►◄
  └─ mtu <max_transfer_unit>─┘   └──────┘                                      
```

Where:

<interface>
: is the interface name that was assigned when the CTCM group device was set online.

<ip_address>
: is the IP address 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 which may 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).

To deactivate an interface issue a command of this form:

```
# ifconfig <interface> down
```

Examples

- This example activates a CTC interface ctc0 with an IP address 10.0.51.3 for a peer with address 10.0.50.1 and an MTU of 32760.

```
# ifconfig ctc0 10.0.51.3 pointopoint 10.0.50.1 mtu 32760
```

- This example deactivates ctc0:

```
# ifconfig ctc0 down
```

Recovering a lost CTC connection

This section applies to CTC interfaces only.

If one side of a CTC connection crashes, you cannot simply reconnect after a reboot. You also need to deactivate the interface on the crashed side's peer. Proceed like this:

1. Reboot the crashed side.
2. Deactivate the interface on the peer (see “Activating and deactivating a CTC interface” on page 164).
3. Activate the interface on the crashed side and on the peer (see “Activating and deactivating a CTC interface” on page 164).

   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.

If the CTC connection is uncoupled, you must couple it again and re-configure the interface of both peers using **ifconfig** (see “Activating and deactivating a CTC interface” on page 164).

Scenarios

This section provides some typical scenarios for CTC connections:

- Connecting to a peer in a different LPAR
- Connecting a Linux instance to a peer instance in the same z/VM system

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 (see Figure 34 on page 166).

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.

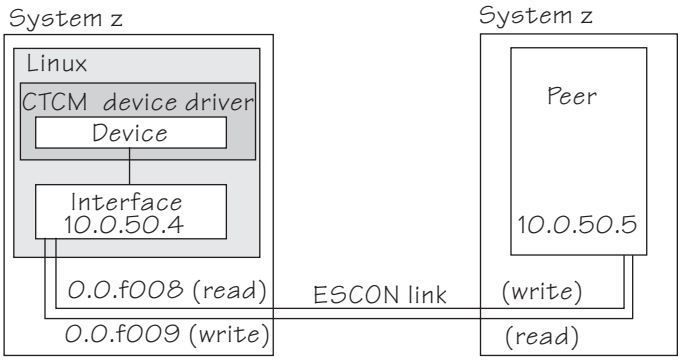


Figure 34. CTC scenario with peer in a different LPAR

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
# dmesg | fgrep "ch-0.0.f008"
ctc0: read: ch-0.0.f008, write: ch-0.0.f009, proto: 1
```

   In the example, the interface name is ctc0.

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:

```
# ifconfig ctc0 10.0.50.4 pointopoint 10.0.50.5
```

Connecting a Linux instance to a peer instance in the same z/VM system

Linux is running as a z/VM guest operating system. The Linux instance is to be connected to another guest operating system of the same z/VM using a virtual CTCA connection (see Figure 35).

Assumptions:

- The guest ID of the peer is “guestp”.
- A separate subnet has been obtained from the TCP/IP network administrator. IP addresses 10.0.100.100 and 10.0.100.101 are to be used by the Linux guest and the peer, respectively.

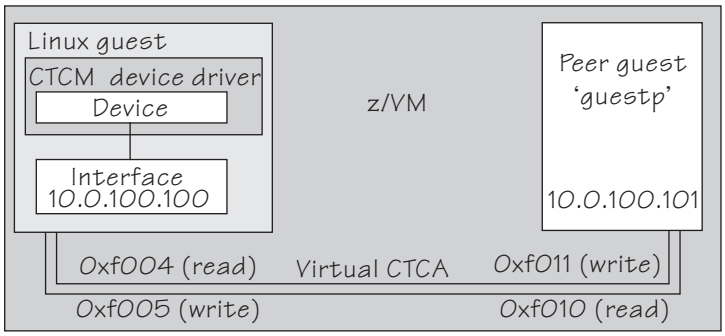


Figure 35. CTC scenario with peer in the same z/VM

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
   ```

   from the console of the running CMS machine (preceded by `#CP` if necessary), or from an `EXEC` file (such as `PROFILE EXEC A`).
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
# dmesg | fgrep "ch-0.0.f004"
ctc1: read: ch-0.0.f004, write: ch-0.0.f005, proto: 1
```

In the example, the interface name is ctc1.

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:

```
# ifconfig ctc1 10.0.100.100 pointopoint 10.0.100.101
```

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 164).

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 163 if the peer is not Linux, refer to the respective operating system documentation.

Part 4. z/VM virtual server integration

This part describes device drivers and features that help 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
`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the Red Hat Enterprise Linux 6 release notes at
`http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/`

Chapter 12. z/VM concepts

This chapter contains information that is not strictly needed to run the functionality in question, however, it might help you understand some of the background.

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 need to read performance data. In the case of monitoring Linux guests, this data is APPLDATA monitor records. Linux instances need to write these records for the tool to read, as shown in Figure 36.

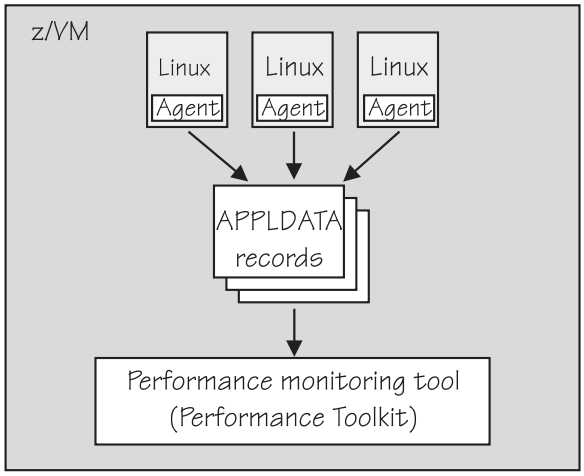


Figure 36. Linux instances write APPLDATA records for performance monitoring tools

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, a user space tool that uses the monwriter device driver to write file system size information as defined records.

For process data there is a command, mon_procd, a user space tool that uses the monwriter device driver to write system information as defined records.

In summary, Red Hat Enterprise Linux 6 for System z supports writing and collecting performance data as follows:

- The Linux kernel can write z/VM monitor data for Linux instances, see Chapter 13, “Writing kernel APPLDATA records,” on page 175.

- Applications running on Linux guests can write z/VM monitor data, see Chapter 14, “Writing application APPLDATA records,” on page 181.
- You can collect monitor file system size information, see “mon_fsstatd – Monitor z/VM guest file system size” on page 417.
- You can collect system information on up to 100 concurrent running processes. see “mon_procd – Monitor Linux guest” on page 422.

Monitoring on Linux

For performance monitoring on Linux, you can use a tool such as Tivoli® OMEGAMON®, or write your own software, and set up a Linux instance to read the monitor data as shown in Figure 37. A Linux instance can read the monitor data using the monreader device driver.

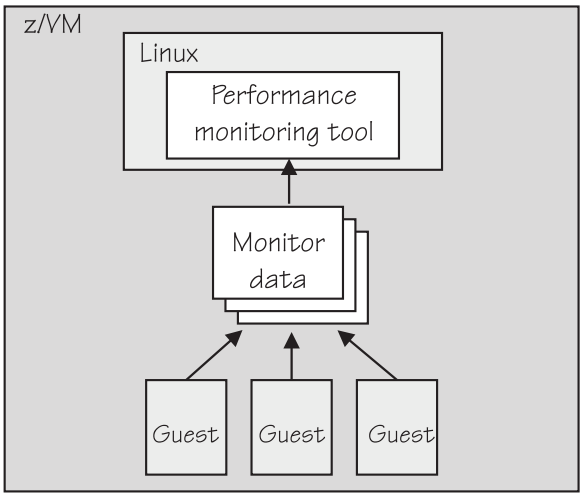


Figure 37. Performance monitoring using monitor DCSS data

In summary, Linux on System z supports reading performance data in the form of read access to z/VM monitor data for Linux instances. For more details, see Chapter 15, “Reading z/VM monitor records,” on page 185.

Further information

- See *z/VM Getting Started with Linux on System z*, SC24-6194, the chapter about monitoring performance for information on using the CP Monitor and the Performance Toolkit for VM.
- See *z/VM Saved Segments Planning and Administration*, SC24-6229 for general information about DCSSs (z/VM keeps monitor records in a DCSS).
- See *z/VM Performance*, SC24-6208 for information about how to create a monitor DCSS.
- See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about the CP commands used in the context of DCSSs and for controlling the z/VM monitor system service.
- For the layout of the monitor records see Chapter 13, “Writing kernel APPLDATA records,” on page 175 and visit

  `www.ibm.com/vm/pubs/mon520/index.html`

For more information about performance monitoring on z/VM, visit

`www.ibm.com/vm/perf`

Cooperative memory management background

This section gives some background information about cooperative memory management (CMM, or "cmm1"). For information about setting it up, see Chapter 23, “Cooperative memory management,” on page 225.

In a virtualized environment it is common practice to give the virtual machines more memory than is actually available to the hypervisor. Linux has the tendency to use all of its available memory. As a result, the hypervisor (z/VM) might start swapping.

To avoid excessive z/VM swapping the available Linux guest memory can be reduced. To reduce Linux guest memory size CMM allocates pages to page pools that make the pages unusable to Linux. Two such page pools exist for a Linux guest, as shown in Figure 38.

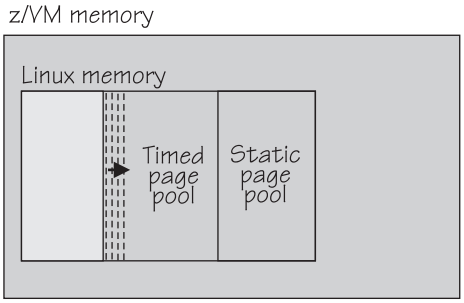


Figure 38. Page pools

The two page pools are:

A static page pool
The page pool is controlled by a resource manager that changes the pool size at intervals according to guest activity as well as overall memory usage on z/VM (see Figure 39).

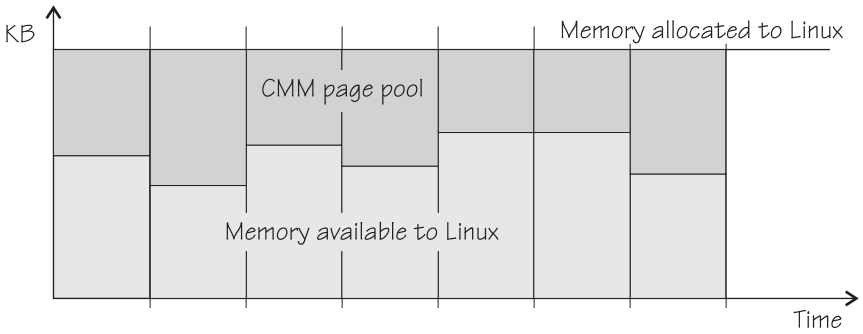


Figure 39. Static page pool. The size of the pool is static for the duration of an interval.

A timed page pool
Pages are released from this pool at a speed set in the *release rate* (see Figure 40 on page 174). 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 will empty.

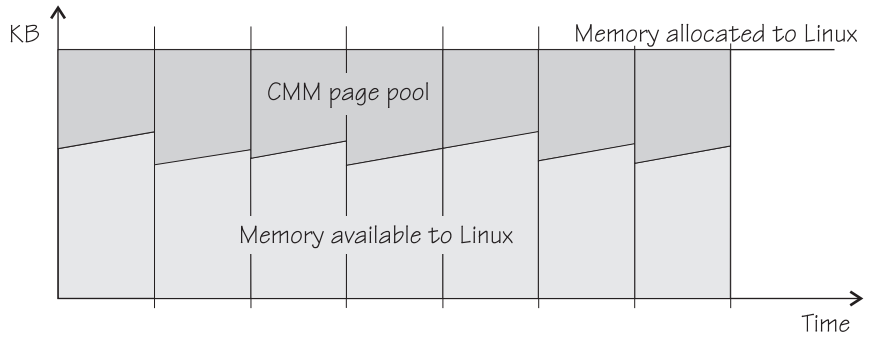


Figure 40. Timed page pool. Pages are freed at a set release rate.

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 book. For more information, see the chapter on VMRM in *z/VM Performance*, SC24-6109.

Third party tools can use a Linux deamon that receives commands for the memory allocation through TCP/IP. The deamon, in turn, uses the a /proc-based interface. You can use the /proc interface to read the pool sizes. This is useful for diagnostics.

Chapter 13. Writing kernel APPLDATA records

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 means of 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 guest to be monitored controls when data is collected. The timer only accounts for 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 177).

For an overview of performance monitoring support, see “Performance monitoring for z/VM guest virtual machines” on page 171.

Setting up the APPLDATA record support

There are no module parameters for the monitor stream support. This section describes how to load those components of the support that have been compiled as separate modules and how to set up your z/VM guest for the APPLDATA record support.

Loading data gathering modules

The data gathering components have been compiled as separate modules. Use the **modprobe** command to load any required modules. See the **modprobe** man page for command details.

APPLDATA record support module parameter syntax

```
►►─modprobe──┬─ appldata_mem──────┬──────────────────────────────────►◄
             ├─ appldata_os───────┤
             └─ appldata_net_sum──┘
```

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.

Enabling your z/VM guest for data gathering

To enable your Linux instance for data gathering ensure that the user directory of the guest virtual machine includes the option APPLMON.

Working with the APPLDATA record support

You control the monitor stream support through the procfs. You can set the timer interval and switch on or off data collection. APPLDATA monitor records are produced if both a particular data gathering module and the monitoring support in general are switched on.

Switching the support on or off

You switch on or off the monitoring support by writing “1” (on) or “0” (off) to `/proc/sys/appldata/timer`.

To read the current setting issue:

```
# cat /proc/sys/appldata/timer
```

To switch on the monitoring support issue:

```
# echo 1 > /proc/sys/appldata/timer
```

To switch off the monitoring support issue:

```
# echo 0 > /proc/sys/appldata/timer
```

Activating or deactivating individual data gathering modules

You can activate or deactivate the data gathering modules individually. 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. The entries are:

`/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 if a module is active look at the content of the corresponding procfs entry.

To activate a data gathering module write “1” to the corresponding procfs entry. To deactivate a data gathering module write “0” to the corresponding procfs entry. 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 switched on (see “Switching the support on or off”).

Example

To find out if 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. The time you set is measured by the virtual CPU timer. Because the virtual timer slows down as the guest idles, the time 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. 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. Valid input must be greater than 0 and less than $2^{31} - 1$. Input values greater than $2^{31} - 1$ produce unpredictable results.

Example

To set the sampling interval to 20 s (20000 ms) issue:

```
# echo 20000 > /proc/sys/appldata/interval
```

APPLDATA monitor record layout

This section describes the layout of the APPLDATA monitor records that can be provided to z/VM. Each of the modules that can be installed with the base module corresponds to a type of record:

- Memory data (see Table 32 on page 178)
- Processor data (see Table 33 on page 179)
- Networking (see Table 34 on page 180)

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 32. APPLDATA_MEM_DATA record (Record ID 0x01)

Offset		Type	Name	Description
Decimal	**Hex**			
0	0x0	u64	timestamp	TOD timestamp 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 should be the same. Otherwise, the record has been 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 read from disk (in KB)
24	0x18	u64	pgpgout	Data written to disk (in KB)
32	0x20	u64	pswpin	Pages swapped in
40	0x28	u64	pswpout	Pages swapped out
48	0x30	u64	sharedram	Shared RAM in KB, set to 0
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 reserved for buffers, free cache in KB
96	0x60	u64	cached	Size of used cache, without buffers 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 33. APPLDATA_OS_DATA record (Record ID 0x02)

Offset		Type (size)	Name	Description
Decimal	**Hex**			
0	0x0	u64	timestamp	TOD timestamp 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 should be the same. Otherwise, the record has been 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	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 composed of 10 bits integer and 11 bits fractional part. See note 1 at the end of this table.
48	0x30	u32	nr_iowait	Number of blocked threads (waiting for I/O).
52	0x34	See note 2.	per_cpu_data	Time spent in user, kernel, idle, nice, etc for every CPU. See note 3 at the end of this table.
52	0x34	u32	per_cpu_user	Timer ticks spent in user mode.
56	0x38	u32	per_cpu_nice	Timer ticks spent with modified priority.
60	0x3C	u32	per_cpu_system	Timer ticks spent in kernel mode.
64	0x40	u32	per_cpu_idle	Timer ticks spent in idle mode.
68	0x44	u32	per_cpu_irq	Timer ticks spent in interrupts.
72	0x48	u32	per_cpu_softirq	Timer ticks spent in softirqs.
76	0x4C	u32	per_cpu_iowait	Timer ticks 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.

Notes:

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 34. APPLDATA_NET_SUM_DATA record (Record ID 0x03)

Offset		Type	Name	Description
Decimal	Hex			
0	0x0	u64	timestamp	TOD timestamp 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 should be the same. Otherwise, the record has been 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	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 received
32	0x20	u64	tx_packets	Total packets transmitted
40	0x28	u64	rx_bytes	Total bytes received
48	0x30	u64	tx_bytes	Total bytes transmitted
56	0x38	u64	rx_errors	Number of bad packets received
64	0x40	u64	tx_errors	Number of packet transmit problems
72	0x48	u64	rx_dropped	Number of incoming packets dropped because of insufficient space in Linux buffers
80	0x50	u64	tx_dropped	Number of outgoing packets dropped because of insufficient space in Linux buffers
88	0x58	u64	collisions	Number of collisions while transmitting

Programming interfaces

This section provides information for those who want to program against the monitor stream.

The monitor stream support base module exports two functions:

- 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 177 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 14. Writing application APPLDATA records

Applications can easily write monitor data in the form of APPLDATA records to the z/VM monitor stream by using the monitor stream application device driver. This character device enables writing of z/VM monitor APPLDATA records.

For an overview of performance monitoring support, see “Performance monitoring for z/VM guest virtual machines” on page 171.

Features

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.

What you should know about the monitor stream application device driver

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 using this device driver.

Further information

- Refer to *z/VM Saved Segments Planning and Administration*, SC24-6229 for general information on DCSSs.
- Refer to *z/VM CP Programming Services*, SC24-6179 for information on the DIAG x'DC' instruction.
- Refer to *z/VM CP Commands and Utilities Reference*, SC24-6175 for information on the CP commands.
- Refer to *z/VM Performance*, SC24-6208 for information on monitor APPLDATA.

Setting up the monitor stream application device driver

This section describes the parameters that you can use to configure the monitor stream write support.

Module parameters

The monitor stream application device driver is compiled as a separate module that you need to load before you can work with it. This section describes how to load and configure the monwriter module.

Monitor stream application device driver module parameter syntax

```
                                  ┌─ max_bufs=255──────────┐
►►──modprobe── monwriter──┬──────────────────────────┬──────────────────────────►◄
                          └─ max_bufs=<NUMBUFS>──────┘
```

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 NUMBUFS, use the following command:

```
# modprobe monwriter max_bufs=NUMBUFS
```

Setting up the z/VM guest virtual machine

Set these options in the z/VM user directory entry of the virtual machine in which the application using this device driver will run:

- OPTION APPLMON

Issue the following CP commands in order to have CP collect the respective types of monitor data:

- `MONITOR SAMPLE ENABLE APPLDATA ALL`
- `MONITOR EVENT ENABLE APPLDATA ALL`

You can either log in to the z/VM console in order to issue the CP commands (in which case the commands would have to be preceded by #CP), or use the **vmcp** command for issuing CP commands from your Linux instance.

Refer to *z/VM CP Commands and Utilities Reference*, SC24-6175 for information on the CP MONITOR command.

Working with the monitor stream application device driver

This device driver writes to the z/VM monitor stream through the z/VM CP instruction DIAG X'DC'. See *z/VM CP Programming Services*, SC24-6179 for more information on the DIAG X'DC' instruction and the different monitor record types (sample, config, event).

The application writes monitor data by passing a monwrite_hdr followed by monitor data (except in the case of the STOP function, which requires no monitor data). The monwrite_hdr, as described in monwriter.h, is filled in by the application and includes the DIAG X'DC' function to be performed, the product identifier, the header length, and the data length.

All records written to the z/VM monitor stream begin with a product identifier. This device driver will use the product ID. The product ID is a 16-byte structure of the form ppppppffnvvrrmm, 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, but to reduce the possibility of conflicts with other applications, a request for an application number should be submitted to 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 could be for specifying version, release, and modification level information, allowing changes to a certain record number when the layout has been changed, without changing the record number itself.

The first seven bytes of the structure (LNXAPPL) will be filled in by the device driver when it writes the monitor data record to the CP buffer. The last nine 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

An application wishing to write APPLDATA must first issue `open()` to open the device driver. The application then needs to issue `write()` calls to start or stop the collection of monitor data and to write any monitor records to buffers that CP will have access to.

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 followed by monitor data (except if it is a STOP function, which is followed by no 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 should precede the new data on the `write()` call.

The monwrite_hdr also includes fields for the header length (useful for calculating the data offset from the beginning of the hdr) and the data length (length of the following monitor data, if any.) See /include/asm-s390/monwriter.h for the definition of monwrite_hdr.

Stopping data writing

When the application has finished writing monitor data, it needs to issue `close()` to close the device driver.

Chapter 15. Reading z/VM monitor records

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). 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 171.

Features

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 records provided by the Linux monitor stream (see Chapter 13, “Writing kernel APPLDATA records,” on page 175).

What you should know about the z/VM *MONITOR record reader device driver

The data that is collected by *MONITOR depends on how you have set up the 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 only supports a single 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 this is the case , you must also use MONDCSS. See “Assuring that the DCSS is addressable for your Linux guest” on page 186 for information on how to check if MONDCSS exists.

Further information

- See *z/VM Saved Segments Planning and Administration*, SC24-6229 for general information on DCSSs.
- See *z/VM Performance*, SC24-6208 for information on how to create a monitor DCSS.
- See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information on the CP commands 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 13, “Writing kernel APPLDATA records,” on page 175.

Setting up the z/VM *MONITOR record reader device driver

This section describes how to set up a Linux guest for accessing an existing monitor DCSS with the z/VM *MONITOR record reader device driver.

Set up the monitor system service and the monitor DCSS on z/VM is beyond the scope of this book. See “Further information” on page 185 for documentation on the monitor system service, DCSS, and related CP commands.

Before you start: Some of the CP commands you need to use for setting up the z/VM *MONITOR record reader device driver require class E authorization.

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. Ensure that the guest's entry in the USER DIRECT file includes the statement:

```
IUCV *MONITOR
```

If the DCSS is restricted you also need the 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 command from a CMS session with privilege class E:

```
#cp q monitor
```

Assuring that the DCSS is addressable for your Linux guest

The DCSS address range must not overlap with the storage of you z/VM guest virtual machine. To find out the start and end address of the DCSS by issuing the following CP command from a CMS session with 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 the DCSS overlaps with the guest storage follow the procedure in “Avoiding overlaps with your Linux guest storage” on page 203.

Specifying the monitor DCSS name

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 need to specify it. Specify the DCSS name as a module parameter when you load the module.

Module parameter

This section describes how to load the monitor read support. It also tells you how to specify a DCSS name, if applicable.

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

```
                          ┌─mondcss=MONDCSS─┐
►►─modprobe─monreader─┤                 ├──────────────────────────────►◄
                          └─mondcss=<dcss>──┘
```

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
```

z/VM *MONITOR record device node

Red Hat Enterprise Linux 6 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.

Working with the z/VM *MONITOR record reader device driver

This section describes how to work with the monitor read support.

- Opening and closing the character device
- Reading monitor records

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 need to close it to make it accessible to other users.

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*, SC24-6208 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

There are two alternative methods for reading:

- Non-blocking read in conjunction with polling
- Blocking read without polling

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 4K control blocks). The MCE contains information on:

- 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 4K page (frame) boundary.

See "Appendix A: *MONITOR" in *z/VM Performance*, SC24-6208 for a description of the monitor control element layout. The layout of the monitor records can be found on

`www.ibm.com/vm/pubs/ctlblk.html`

The layout of the data stream 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 may 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 guest” on page 186).

Error conditions are indicated by returning a negative value for the number of bytes read. In case of an error condition, the errno variable can be:

EIO Reply failed. All data read since the last successful read with 0 size is not valid. Data will be missing. The application must decide whether to continue reading subsequent data or to exit.

EFAULT
Copy to user failed. All data read since the last successful read with 0 size is not valid. Data will be 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. There is no data missing or damaged, retry or use polling for non-blocking reads.

EOVERFLOW

Message limit reached. The data 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 16. z/VM recording device driver

The z/VM recording device driver can be used by Linux systems that run as z/VM guest operating systems. The device driver enables the Linux guest 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 RECORDING command to collect records and IUCV to transmit them to the Linux guest.

Features

The z/VM recording device driver supports the following devices and functions:

- 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 193).

For general information about CP recording system services refer to *z/VM CP Programming Services*, SC24-6179.

What you should know about the z/VM recording device driver

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 as shown in Table 35:

Table 35. z/VM recording device names

z/VM recording service	Standard device name
*LOGREC	logrec
*ACCOUNT	account
*SYMPTOM	symptom

Reading records

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 containing 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. Theses bytes build the zero terminated ASCII string “EOR”, which is useful as an eye catcher.

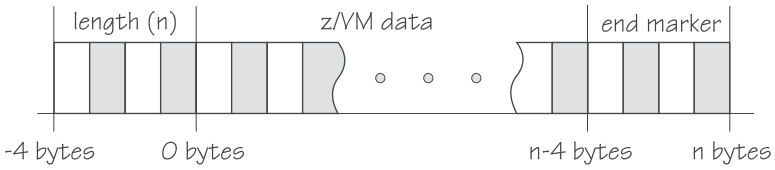


Figure 41. Record structure

Figure 41 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 described in the common header for error records HDRREC.

For more information on the z/VM record layout, refer to 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

This section provides information on the guest authorization you need to be able to collect records and on how to load the device driver module.

Authorizing the Linux guest

The Linux guest must be authorized to use the z/VM RECORDING command. Depending on the z/VM environment, this could be either of the following authorization classes: A, B, C, E, or F.

The guest must also be authorized to connect to those IUCV services it needs to use.

Loading the z/VM recording device driver

There are no module parameters for the z/VM recording device driver.

You need to 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
```

Working with z/VM recording devices

This section describes typical tasks that you need to perform when working with z/VM recording devices.

- Starting and stopping record collection
- Purging existing records
- Querying the z/VM recording status
- Opening and closing devices
- Reading records

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.

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.

Be aware that you cannot start record collection if a device is open and there are already existing records. 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 194).

To be able to open a device without starting record collection and to close a device without stopping record collection write “0” to the devices autorecording attribute. To restore the automatic starting and stopping of record collection write “1” to the devices 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 switch on record collection write “1” to the devices recording attribute. To explicitly switch off record collection write “0” to the devices 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 the 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 switched 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, use the record_status attribute as described in “Querying the z/VM recording status” on page 194.

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.

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.

To be able to open or close a device without purging existing records write “0” to the devices autopurge attribute. To restore automatic purging of existing records write “1” to the devices 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 devices 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

You can use the record_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 will not necessarily have an entry for all three services and there might be additional entries for other guests.

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
```

This will result in output similar to the following:

```
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 will result in a message to the operator
- The guest that is or was connected to that service and the current status of that connection

A detailed description of the QUERY RECORDING command can be found in the *z/VM CP Commands and Utilities Reference*, SC24-6175.

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.

You can use a device's autorecord attribute (see “Starting and stopping record collection” on page 193) to enable automatic record collection while a device is open.

You can use a device's autopurge attribute (see “Purging existing records” on page 194) to enable automatic purging of existing records when a device is opened and closed.

Scenario: Connecting to the *ACCOUNT service.

This scenario demonstrates autorecording, turning autorecording off, purging records, and starting recording.

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, but should be similar to the following:

   ```
   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 will connect the guest to the *ACCOUNT service and start recording. The entry for *ACCOUNT on guest LINUX31 will change 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. Note that 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 may 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.
```

Note that 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 we can start recording, 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 17. z/VM unit record device driver

The z/VM unit record device driver provides Linux 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 s390-tools package (see “vmur - Work with z/VM spool file queues” on page 448).

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.

z/VM unit record device nodes

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 the vmur device driver

After loading the vmur module, the required virtual unit record devices need to be set online. For example, to set the devices with device bus IDs 0.0.000c, 0.0.000d, and 0.0.000e online, issue:

```
# chccwdev -e 0.0.000c-0.0.000e
```

When unloading vmur (with rmmod) the respective unit record device nodes must not be open, otherwise the error message "Module vmur is in use" is displayed.

Serialization is implemented per device; only one process can open a given device node at a given time.

Chapter 18. z/VM DCSS device driver

The z/VM discontiguous saved segments (DCSS) device driver provides disk-like fixed block access to z/VM discontiguous saved segments.

Features

The DCSS device driver facilitates:

- Initializing and updating ext2 compatible file system images in z/VM saved segments for use with the xip option of the ext2 file system.
- Implementing a shared read-write RAM disk for Linux guests, for example, for a file system that can be shared among multiple Linux images that run as guest systems under the same z/VM.

Starting with z/VM 5.4, you can:

- Locate a DCSS above 2047 MB
- Set up DCSS devices with a size above 2047 MB by mapping multiple DCSSs to a single DCSS block device

What you should know about DCSS

This section provides information about the DCSS device names and nodes.

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.

z/VM DCSS device nodes

User space programs access DCSS devices by device nodes. Red Hat Enterprise Linux 6 provides udev to create standard DCSS device nodes of the form `/dev/<device_name>`, for example:

```
/dev/dcssblk0
/dev/dcssblk1
...
```

Accessing a DCSS in exclusive-writable mode

You need to access a DCSS in exclusive-writable mode, for example, when creating or updating 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*, SC24-6178 for more information about the NAMESAVE statement.
- The DCSS has been defined with the LOADNSHR operand. See “DCSS options” about how to save DCSSs with optional properties.

  See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about the LOADNSHR operand.

DCSS options

The z/VM DCSS device driver always saves DCSSs with default properties. Any options that have previously been defined are removed. For example, a DCSS that has been defined with the LOADNSHR operand no longer has this property after being saved through 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 208 for an example.

See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about DCSS options.

Further information

- For information on DCSS see *z/VM Saved Segments Planning and Administration*, SC24-6229
- For related z/VM information see *z/VM CP Commands and Utilities Reference*, SC24-6175.
- For an example of how the xip option for the ext2 file system and DCSS can be used see *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594 on developerWorks at:

  `www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

Setting up the DCSS device driver

Before you can load and use DCSSs, you must load the 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

```
                                  ,
                                  :
►►─modprobe── dcssblk── segments=──┬──┬──<dcss>──┬──────────┬──────────────►◄
                                   ▼  ▼          └─(local)──┘
```

<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 have been 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 Linux guest storage

Ensure that your DCSSs do not overlap with the memory of your z/VM guest virtual machine (guest storage). To find the start and end addresses of the DCSSs, enter the following CP command from a CMS session with 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.

If any DCSS that you intend to access with your guest machine overlaps with the guest storage, redefine the guest storage as two or more discontiguous storage extents such that the storage gap with the lowest address range covers all your DCSSs' address ranges.

Notes:

1. You cannot place a DCSS into a storage gap other than the storage gap with the lowest address range.

2. A z/VM guest that has been 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 required for memory management functions make the lower limit at least 128 MB. The lower limit for the DCSS increases with the total memory size and 128 MB is not an exact value but 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*, SC24-6175 for more information on 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 ranges from 140 MB to 160 MB and thus covers the entire DCSS range. The total guest storage is 140 MB + 372 MB = 512 MB.

Working with the DCSS device driver

This section describes typical tasks that you need to perform when working with DCSS devices:

- Adding a DCSS device
- Listing the DCSSs that map to a particular device
- Finding the minor number for a DCSS device
- Setting the access mode
- Saving updates to a DCSS or set of DCSSs
- Removing a DCSS device

Adding a DCSS device

Before you start:

- You need to have set up one or more DCSSs on z/VM and know the names assigned to the DCSSs 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 Linux guest storage” on page 203).
- If a z/VM guest has been defined with multiple storage gaps, you can only add DCSSs that are located in the storage gap with the lowest address range.
- If a z/VM guest has been defined with one or more storage gaps, you cannot add a DCSS that is located above the guest storage.

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 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

To list the DCSSs that map to a DCSS device, issue a command like this:

```
# 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 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. Unless you use dynamically created device nodes as provided by udev, you might need to know the minor device number that has been assigned to the DCSS (see “DCSS naming scheme” on page 201).

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 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. There are two possible write access modes to the DCSS device:

shared

In the shared mode, changes to DCSSs are immediately visible to all 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 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 207).

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 has stopped 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 201).

For either access mode the changes are volatile until they are saved (see “Saving updates to a DCSS or set of DCSSs”).

Set the access mode before you open the DCSS device. To set the access mode to exclusive-writable set the DCSS device's shared attribute to “0”. To reset the access mode to shared set the DCSS device's shared attribute to “1”.

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

Before you start:

- Saving a DCSS as described in this section results in a default DCSS, without optional properties. For DCSSs that have been defined with options (see “DCSS options” on page 202), see “Workaround for saving DCSSs with optional properties” on page 208.
- 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.

To place a request for saving changes permanently on the spool disk write “1” to the DCSS device's save attribute. If a set of DCSSs has been mapped to the DCSS device, the save request applies to all DCSSs in the set.

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

Note: 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. Only use this workaround if you really need a DCSS with special options.

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.

See *z/VM CP Commands and Utilities Reference*, SC24-6175 for details about the DEFSEG and SAVESEG CP commands.

Removing a DCSS device

Before you start: A DCSS device can only be removed when it is not in use.

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.

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.

Example

To remove a DCSS device that is represented by the DCSS name “MYDCSS” issue:

```
# echo MYDCSS > /sys/devices/dcssblk/remove
```

Changing the contents of a DCSS

The following scenario describes how you can use the DCSS block device driver to change the contents of a DCSS.

Assumptions:

- The Linux instance runs as a z/VM guest with class E user privileges.
- A DCSS has been 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 has been set up and is ready to be used.

Note: 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. The assumptions about the set of DCSSs would be that the contiguous memory space formed by the set does not overlap with the guest storage and that only the DCSSs in the set are added to the Linux instance.

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.

- Find out the major number used for DCSS block devices. Read `/proc/devices`:

```
# cat /proc/devices
...
Block devices
...
254 dcssblk
...
```

  The major number in the example is 254.
- Find out the minor number used for MYDCSS. If MYDCSS is the first DCSS that has been 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.
- 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.

   Example:

```
# 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. If you have created your own device node, you can optionally clean it up.

```
# rm -f /dev/dcssblk0
```

Chapter 19. Shared kernel support

You can save a Linux kernel in a z/VM named saved system (NSS). Through an NSS, z/VM makes operating system code in shared real memory pages available to z/VM guest virtual machines. Multiple Linux guest operating systems on the z/VM can then boot from the NSS and run from the single copy of the Linux kernel in memory.

For a z/VM guest virtual machine a shared kernel in an NSS amounts to a fast boot device. In a virtual Linux server farm with multiple z/VM guest virtual machines sharing the NSS, the NSS can help to reduce paging and enhance performance.

What you should know about NSS

Before you create an NSS you need to have a Linux system that supports kernel sharing installed on a conventional boot device, for example, a DASD or SCSI disk. You create the NSS when you use a special boot parameter to boot the Linux system from this original boot device.

For more information on NSS and the CP commands used in this section see:

- *z/VM CP Commands and Utilities Reference*, SC24-6175 at the IBM Publications Center (see “Finding IBM books” on page xi).
- *z/VM Virtual Machine Operation*, SC24-6241 at the IBM Publications Center (see “Finding IBM books” on page xi).

Kernel parameter for creating an NSS

You create an NSS with a shared kernel by booting a Linux system with shared kernel support with the savesys= parameter.

kernel parameter syntax

```
►►──savesys=<nss_name>──────────────────────────────────────────►◄
```

where *<nss_name>* is the name you want to assign to the NSS. The name can be one to eight characters long and must consist of alphabetic or numeric characters. Be sure not to assign a name that matches any of the device numbers used at your installation.

Note: If *<nss_name>* contains non-alphanumeric characters, the NSS might be created successfully. However, this name might not work in CP commands. Always use alphanumeric characters for the name.

Working with a Linux NSS

This section describes how you can create and maintain a Linux NSS. For information about booting Linux from an NSS see “Using a named saved system” on page 321. Note that Kexec is disabled for Linux instances booted from a kernel NSS.

For each task described in this section you need a z/VM guest virtual machine that runs with class E privileges.

Creating a Linux NSS using zipl

Perform these steps to create a Linux NSS:

1. Boot Linux.
2. Insert savesys=*<nssname>* into the kernel parameter file used by your boot configuration, where *<nssname>* is the name you want to assign to the NSS. The name can be 1-8 characters long and must consist of alphabetic or numeric characters. Examples of valid names include: 73248734, NSSCSITE, or NSS1234. Be sure not to assign a name that matches any of the device numbers used at your installation.
3. Issue a **zipl** command to write the modified configuration to the boot device.
4. Close down Linux.
5. Issue an IPL command to boot Linux from the device that holds the Linux kernel. During the IPL process, the NSS is created and Linux is actually rebooted from the NSS.

You can now use the NSS to boot Linux in other z/VM guest virtual machines. See "Using a named saved system" on page 321 for details.

Creating a Linux NSS from the CP command line

You can create a Linux NSS without booting Linux and without editing the **zipl** parameter file. To boot Linux and save it as an NSS issue an IPL command of this form:

```
IPL <devno> PARM savesys=<nssname>
```

where *<devno>* refers to a device that designates the Linux system that is to be saved as an NSS; and *<nssname>* is the name you want to assign to the NSS.

The NSS name can be 1-8 characters long and must consist of alphabetic or numeric characters. Examples of valid names include: 73248734, NSSCSITE, or NSS1234. Be sure not to assign a name that matches any of the device numbers used at your installation.

During the IPL process, the NSS is created and Linux is booted from the NSS.

Example: To create a Linux NSS from CP when a standard Linux system is installed on device 1234, use the following command:

```
IPL 1234 PARM savesys=lnxnss
```

Once the Linux NSS has been defined and saved, it can be booted using its name:

```
IPL lnxnss
```

For information about the PARM attribute, see "Specifying kernel parameters when booting Linux" on page 19.

Updating a Linux NSS

Perform these steps to update a Linux NSS:

1. Boot the updated version of your Linux system.
2. Include savesys=*<nssname>* into the kernel parameters used by your boot configuration, where *<nssname>* is the name of the NSS you want to update. See “Preparing a boot device” on page 293 for information about the boot configuration.
3. Issue a **zipl** command to write the modified configuration to the boot device.
4. Close down Linux.
5. Issue an IPL command to boot Linux from the device that holds the updated Linux kernel. During the IPL process, the NSS is updated and Linux is booted from the NSS.

Deleting a Linux NSS

Perform these steps to delete an obsolete Linux NSS:

1. Close down all Linux instances that use the NSS.
2. Issue a CP PURGE NSS NAME command to delete the NSS. For example, issue a command of this form

   ```
   PURGE NSS NAME <nssname>
   ```

   where *<nssname>* is the name of the NSS you want to delete.

Chapter 20. Watchdog device driver

The watchdog device driver provides Linux watchdog applications with access to the z/VM watchdog timer.

Watchdog applications can be used to set up automated restart mechanisms for Linux instances running as z/VM guest operating systems. Watchdog-based restart mechanisms are an alternative to a networked heartbeat in conjunction with STONITH.

A watchdog application that communicates directly with the z/VM control program (CP) does not require a third operating system to monitor a heartbeat. The watchdog device driver enables you to set up a restart mechanism of this form.

Features

The watchdog device driver provides:

- Access to the z/VM watchdog timer.
- An API for watchdog applications (see “External programming interfaces” on page 217).

What you should know about the watchdog device driver

The watchdog function comprises the watchdog timer that runs on z/VM and a watchdog application that runs on the Linux guest being controlled. While the Linux guest operates satisfactory, the watchdog application reports a positive status to the z/VM watchdog timer at regular intervals. The watchdog application uses a miscellaneous character device to pass these status reports to the z/VM timer (Figure 42).

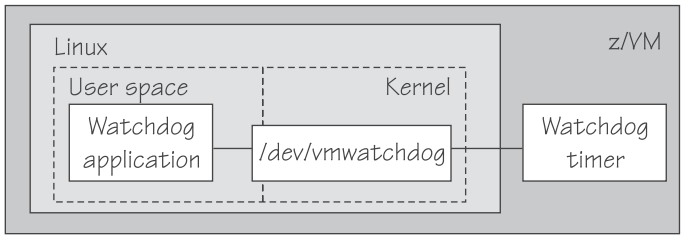


Figure 42. Watchdog application and timer

The watchdog application typically derives its status by monitoring, critical network connections, file systems, and processes on the Linux guest. If a given time elapses without a positive report being received by the watchdog timer, the watchdog timer assumes that the Linux guest is in an error state. The watchdog timer then triggers a predefined action from CP against the Linux guest. Examples of possible actions are: shutting down Linux, rebooting Linux, or initiating a system dump. For information on how to set the default timer and how to perform other actions, see “External programming interfaces” on page 217.

Note: 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. This may cause a watchdog to timeout and restart the guest. You are advised not to use the watchdog in combination with loading or saving DCSSs.

You can find an example watchdog application at
`www.ibiblio.org/pub/Linux/system/daemons/watchdog/!INDEX.html`

See also the generic watchdog documentation in the Linux kernel source tree under `Documentation/watchdog`.

Setting up the watchdog device driver

This section describes the parameters that you can use to configure the watchdog device driver and how to assure that the required device node exists.

Module parameters

This section describes how to load and configure the watchdog device driver module.

watchdog module parameter syntax

```
                                 ┌─ cmd="IPL CLEAR"─┐
►►─modprobe─vmwatchdog─┬──────────────────────────┴──────────────────────────►
                       └─ cmd=<command>─────────────┘

  ┌─ conceal=0──────────────┐
►─┼─────────────────────────┼──┬──────────────────────────────────────────►◄
  └─ conceal=<conceal_flag>─┘  └─ nowayout=<nowayout_flag>─┘
```

where:

<command>
is the command to be issued by CP if the Linux guest fails. The default “IPL” reboots the guest with the previous boot parameters.

Instead of rebooting the same system, you could also boot from an alternate IPL device (for example, a dump device). You can also specify multiple commands to be issued, see “Examples” on page 217 for details. For more information about CP commands see *z/VM CP Commands and Utilities Reference*, SC24-6175.

The specification for *<command>*:

- Can be up to 230 characters long
- Needs to be enclosed by quotes if it contains any blanks or newline characters
- Is converted from ASCII to uppercase EBCDIC

<conceal_flag>
turns on and off the protected application environment where the guest is protected from unexpectedly entering CP READ. “0” turns off the protected environment, “1” enables it. The default is “0”.

For details, see the “SET CONCEAL” section of *z/VM CP Commands and Utilities Reference*, SC24-6175.

<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 z/VM watchdog timer keeps running and triggers the command specified for *<command>* if no positive status report is received within the given 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 guest continues without the watchdog support.

Examples

The following command loads the watchdog module and determines that, on failure, the Linux guest 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 vmwatchdog cmd="ipl b1a0" nowayout=0
```

The following example shows how to specify multiple commands to be issued.

```
echo -en "cmd1\ncmd2\ncmd3" > /sys/module/vmwatchdog/parameters/cmd
```

To verify that your commands have been accepted, issue:

```
cat /sys/module/vmwatchdog/parameters/cmd
cmd1
cmd2
cmd3
```

You cannot specify multiple commands as module parameters while loading the module.

Watchdog device node

The watchdog application on Linux needs a misc character device to communicate with the z/VM watchdog timer. This device node is created by udev and is called `/dev/watchdog`.

External programming interfaces

This section provides information for those who want to program watchdog applications that work with the watchdog device driver.

For information about the API see the following files in the Linux source tree:

- `Documentation/watchdog/watchdog-api.txt`
- `include/linux/watchdog.h`

The default watchdog timeout is 60 seconds, the minimum timeout that can be set through the IOCTL SETTIMEOUT is 15 seconds.

The following IOCTLs are supported:

- WDIOC_GETSUPPORT
- WDIOC_SETOPTIONS (WDIOS_DISABLECARD, WDIOS_ENABLECARD)
- WDIOC_GETTIMEOUT
- WDIOC_SETTIMEOUT
- WDIOC_KEEPALIVE

Chapter 21. z/VM CP interface device driver

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 only works when Linux is running as a z/VM guest operating system.

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 console. There are two ways of using the z/VM CP interface driver:

- Through the `/dev/vmcp` device node
- Through a user space tool (see “vmcp - Send CP commands to the z/VM hypervisor” on page 446)

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, for example, add "`modprobe vmcp`" to /etc/rc.modules:

```
# echo modprobe vmcp >> /etc/rc.modules
# chmod +x /etc/rc.modules
```

The vmcp device driver only works under z/VM and cannot be loaded if the Linux system runs in an LPAR.

Differences between vmcp and a 3270 console

Most CP commands behave identically with vmcp and on a 3270 console. However, some commands show a different behavior:

- Diagnose X'08' (see *z/VM CP Programming Services*, SC24-6179) requires you to specify a response buffer in conjunction with the command. As the size of the response is not known beforehand the default response buffer used by vmcp might be too small to hold the full response and as a result the response is truncated.
- On a 3270 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) you should 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 console. For example, the command `vmcp link user1 1234 123 mw` might return the message "DASD 123 LINKED R/W" in a 3270 console. This message will not appear when using vmcp. For details, see the z/VM help system or *z/VM CP Commands and Utilities Reference*, SC24-6175.

Setting up the z/VM CP interface

There are no module parameters for the vmcp device driver.

You must load the vmcp module before you can work with the z/VM CP interface device driver. You can use the **modprobe** command to load the module:

```
# modprobe vmcp
```

Working with the device node

The `/dev/vmcp` device node is a character device node. You can use the device node directly from an application 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 36. 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; /sbin/vmcp calls this ioctl to set it to 8 KB instead).
VMCP_GETSIZE	_IOR(0x10, 3, int)	Queries the size of the response.

Chapter 22. AF_IUCV address family support

The Inter-User Communication Vehicle (IUCV) is a z/VM communication facility that enables a program running in one z/VM guest virtual machine to communicate with another z/VM guest virtual machine, or with a control program (CP), or even with itself.

The AF_IUCV address family provides communication and addressing in the IUCV domain. In the IUCV domain, address spaces or virtual machines can use the socket interface to communicate with other virtual machines or address spaces within the same z/VM guest operating system.

AF_IUCV connects socket applications running on different Linux guest operating systems, or it connects a Linux application to another socket application running in another z/VM guest operating system (for example, z/VM CMS or z/VSE).

The AF_IUCV address family supports stream-oriented sockets (SOCK_STREAM) and connection-oriented datagram sockets (SOCK_SEQPACKET). Stream-oriented sockets fragment data over several native IUCV messages, whereas sockets of type SOCK_SEQPACKET map a particular socket write or read operation to a single native IUCV message.

Features

The AF_IUCV address family provides:

- Multiple outgoing socket connections from a Linux guest operating system.
- Multiple incoming socket connections to a Linux guest operating system.
- Socket communication with applications utilizing CMS AF_IUCV support.

Setting up the AF_IUCV address family support

This section describes the IUCV authorization you need for your z/VM guest virtual machine. It also describes how to load those components that have been compiled as separate modules. There are no kernel or module parameters for the AF_IUCV address family support.

Setting up your z/VM guest virtual machine for IUCV

This section provides an overview of the required 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*, SC24-6179 and *z/VM CP Planning and Administration*, SC24-6178.

Granting IUCV autorizations

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 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 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 6 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 and the required iucv module:

```
# modprobe af_iucv
```

Working with the AF_IUCV address family support

To use the AF_IUCV support, specify AF_IUCV as the socket address family and AF_IUCV address information in the sockaddr structure. The AF_IUCV constant on Linux on System z is 32. The primary difference between AF_IUCV sockets and TCP/IP sockets is how partners are identified (for example, how they are named). The sockaddr structure for AF_IUCV is:

```
struct sockaddr_iucv {
    sa_family_t    siucv_family;     /* AF_IUCV */
    unsigned short siucv_port;       /* reserved */
    unsigned int   siucv_addr;       /* reserved */
    char           siucv_nodeid[8];  /* reserved */
    char           siucv_userid[8];  /* guest user id */
    char           siucv_name[8];    /* application name */
};
```

where:

siucv_family
is set to AF_IUCV (= 32).

siucv_port, siucv_addr, and siucv_nodeid
are reserved for future use. The siucv_port and siucv_addr fields must be zero. The siucv_nodeid field must be set to exactly eight blank characters.

siucv_userid
is set to the z/VM user ID of the Linux guest virtual machine running the application that owns the address. This field must be eight characters long, padded with blanks on the right.

For the bind operation, siucv_userid must contain blanks only to allow AF_IUCV to set the correct z/VM user ID of the Linux guest operating system.

siucv_name
is set to the application name by which the socket is known. Servers advertise application names and clients use these application names to connect to servers. This field must be eight characters long, padded with blanks on the right.

Similar to TCP or UDP ports, application names distinguish separate applications on the same z/VM guest virtual machine that is reachable over IUCV. Do not call bind for names beginning with `lnxhvc`. These names are reserved for the z/VM IUCV HVC device driver.

For further details see the af_iucv man page.

Chapter 23. Cooperative memory management

The cooperative memory management (CMM, or "cmm1") is a mechanism to reduce the available memory of a Linux instance. CMM allocates pages to a dynamic page pool not available to Linux. A diagnose code indicates to z/VM that the pages in the page pool are out of use. z/VM can then immediately reuse these pages for other guests.

To set up CMM, you need to:

1. Incorporate cmm by loading the cmm module.
2. Set up a resource management tool that controls the page pool. This 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 on CMM, see “Cooperative memory management background” on page 173.

You can also use the cpuplugd command to define rules for cmm behavior, see “Managing memory” on page 373.

Setting up the external resource manager is beyond the scope of this book. For more information, see the chapter on VMRM in *z/VM Performance*, SC24-6208.

Setting up cooperative memory management

This section describes how to set up a Linux instance to participate in the cooperative memory management when running as a z/VM guest operating system.

Loading the cooperative memory management module

The cooperative memory management support is compiled as a module, cmm. Use the **modprobe** command to load the module. Refer to the **modprobe** man page for command details.

cooperative memory management module parameter syntax

```
►►──modprobe── cmm──┬─sender=VMRMSVM──────┬──────────────────────────────────►◄
                    └─sender=<user_ID>────┘
```

where *<user_ID>* specifies the z/VM guest virtual machine that is permitted to send messages to the module through the special messages interface. Specify all alphabetic characters as uppercase characters. The default z/VM user ID is VMRMSVM, which is the default for the VMRM service machine.

Example

To load the cooperative memory management module and allow the guest TESTID to send messages:

```
# modprobe cmm sender=TESTID
```

Working with cooperative memory management

After set up, CMM works through the resource manager. No further actions are necessary. The following information is given for diagnostic purposes.

To reduce Linux guest memory size CMM allocates pages to page pools that make the pages unusable to Linux. There are two such page pools for a Linux guest, a static pool and a timed pool. You can use the /proc interface to read the sizes of the page pools.

Reading the size of the static page pool

To read the current size of the static page pool:

```
# cat /proc/sys/vm/cmm_pages
```

Reading the size of the timed page pool

To read the current size of the timed page pool:

```
# cat /proc/sys/vm/cmm_timed_pages
```

Part 5. System resources

This section describes device drivers and features that help to manage the resources of your real or virtual hardware.

Newest version: You can find the newest version of this book at
`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the Red Hat Enterprise Linux 6 release notes at
`http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/`

Chapter 24. Managing CPUs

Some attributes that govern CPUs are available in sysfs under:

`/sys/devices/system/cpu/cpu<N>`

where *<N>* is the number of the CPU. You can read CPU capability, activate standby CPUs, and examine the CPU topology using the CPU attributes in sysfs.

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 in:

`/sys/devices/system/cpu/cpu<N>/capability`

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 lower value indicates a proportionally higher CPU capacity. Beyond that, there is no formal description of the algorithm 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.

Activating standby CPUs and deactivating operating CPUs

A CPU on an LPAR can be in a configured, standby, or reserved state. Under Linux, on IPL only CPUs that are in a configured state are brought online and used. The kernel operates only with configured CPUs. You can change the state of standby CPUs to configured state and vice versa.

Reserved CPUs cannot be used without manual intervention and therefore are not recognized.

Before you begin:

- Sysfs needs to be mounted to /sys.
- To put a CPU into standby state the underlying hypervisor needs to support this operation.

To configure or deconfigure a CPU its physical address needs to be known. Because the sysfs interface is used to configure a CPU by its sysfs entry this requires a static mapping of physical to logical CPU numbers. The physical address of a CPU can be found in the address attribute of a logical CPU:

```
# cat /sys/devices/system/cpu/cpu<N>/address
```

For example:

```
# cat /sys/devices/system/cpu/cpu0/address
0
```

To activate a standby CPU:

1. Only present CPUs have a sysfs entry. If you add a CPU to the system the kernel automatically detects it . You can force the detection of a CPU using the rescan attribute. To rescan, write any string to the rescan attribute, for example:

```
echo 1 > /sys/devices/system/cpu/rescan
```

When new CPUs are found new sysfs entries are created and they are in the configured or standby state depending on how the hypervisor added them.

2. Change the state of the CPU to configured by writing "1" to its configure attribute:

```
echo 1 > /sys/devices/system/cpu/cpu<X>/configure
```

where *<X>* is any CPU in standby state.

3. Bring the CPU online by writing "1" to its online attribute:

```
echo 1 > /sys/devices/system/cpu/cpu<X>/online
```

To deactivate an operating CPU:

1. Bring the CPU offline by writing "0" to its online attribute:

```
echo 0 > /sys/devices/system/cpu/cpu<X>/online
```

2. Change the state of the CPU to standby by writing "0" to its configure attribute:

```
echo 0 > /sys/devices/system/cpu/cpu<X>/configure
```

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. Use this, for example, to optimize the Linux scheduler, which bases its decisions on which process gets scheduled to which CPU. Depending on the workload, this might increase cache hits and therefore overall performance.

Note: By default CPU topology support is disabled in the Linux kernel. If it is advantageous to your workload, enable it by specifying the kernel parameter `topology=on` in your parmfile or zipl.conf.

Before you begin:

- The sysfs needs to be mounted to /sys.

The common code attribute core_siblings will be visible for all online CPUs:

```
/sys/devices/system/cpu/cpu<N>/topology/core_siblings
```

It contains a CPU mask that tells you which CPUs (including the current one) are close to each other. If a machine reconfiguration causes the CPU topology to change, then change uevents will be created for each online CPU.

Note that when the kernel also supports standby CPU activation/deactivation (see “Activating standby CPUs and deactivating operating CPUs” on page 229) then the core_siblings CPU mask also contains the CPUs that are in a configured, but offline state. Updating the mask after a reconfiguration might take some time.

CPU polarization

You can optimize the operation of a vertical SMP environment by adjusting the SMP factor based on the workload demands. During peak workloads the operating system may operate on a large n-way, with all CPUs busy, whereas at other times it may fall back to a single processor. This limits the performance effects of context switches, TLB flushes, cache poisoning, as well as dispatcher workload balancing and the like, by delivering better processor affinity for particular workloads.

Before you begin:

- The sysfs needs to be mounted to /sys.

Horizontal CPU polarization means that the underlying hypervisor will dispatch each of the guests' virtual CPUs for the same amount of time.

If vertical CPU polarization is active then the hypervisor will dispatch 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.

Note: Running a system with different types of vertical CPUs may result in significant performance regressions. If possible, use only one type of vertical CPUs. Set all other CPUs offline and deconfigure them.

Use the dispatching attribute to switch between horizontal and vertical CPU polarization. To switch between the two modes write a `0` for horizontal polarization (the default) or a `1` for vertical polarization to the dispatching attribute.

```
/sys/devices/system/cpu/dispatching
```

The polarization of each CPU can be seen from the polarization attribute of each CPU:

```
/sys/devices/system/cpu/cpu<N>/polarization
```

Its contents is one of:

- `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`

When switching polarization the polarization attribute might contain the value `unknown` until the configuration change is done and the kernel has figured out the new polarization of each CPU.

Chapter 25. Managing hotplug memory

You can dynamically increase or decrease the memory for your running Linux system. 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.

What you should know about memory hotplug

This section explains how hotplug memory is represented in sysfs and how rebooting Linux affects hotplug memory.

How memory is represented 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 sections of equal size. Each section is represented as a directory of the form `/sys/devices/system/memory/memory<n>`, where *<n>* is an integer. You can find out the section size by reading the `/sys/devices/system/memory/block_size_bytes` attribute.

In the naming scheme, the memory sections with the lowest address ranges are assigned the lowest integer numbers. Accordingly, the core memory begins with memory0. The hotplug memory sections follow the core memory sections.

You can infer where the hotplug memory begins by calculating the number of core memory sections from the size of the base memory and the section size. For example, for a core memory of 512 MB and a section size of 128 MB, the core memory is represented by 4 sections, memory0 through memory3. In this example, the first hotplug memory section is memory4. Another Linux instance with a core memory of 1024 MB and access to the same hotplug memory, represents this first hotplug memory section as memory8.

The hotplug memory is available to all operating system instances within the z/VM system or LPARs to which it has been defined. The `state` sysfs attribute of a memory section indicates whether the section is in use by your own Linux system. The `state` attribute does not indicate whether a section is in use by another operating system instance. Attempts to add memory sections that are already in use fail.

Hotplug memory and reboot

The original core memory is preserved as core memory and hotplug memory is freed when rebooting a Linux instance.

When you perform an IPL after shutting down Linux, always use `ipl clear` to preserve the original memory configuration.

Further information

For more information on memory hotplug, see `Documentation/memory-hotplug.txt` in the Linux source tree.

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*. z/VM supports standby storage as of version 5.4. There is also *reserved storage* in z/VM, but other than reserved memory defined for an LPAR, reserved storage defined in z/VM is not available as hotplug memory.

For information about defining standby memory for z/VM guest operating systems see the "DEFINE STORAGE" section in *z/VM CP Commands and Utilities Reference*, SC24-6175.

Performing memory management tasks

This section describes typical memory management tasks.

- Finding out the memory section size
- Displaying the available memory sections
- Adding memory
- Removing memory

Finding out the memory section size

You can find out the size of your memory sections by reading `/sys/devices/system/memory/block_size_bytes`. This sysfs attribute contains the section size in byte in hexadecimal notation.

Example:

```
# cat /sys/devices/system/memory/block_size_bytes
8000000
```

This hexadecimal value corresponds to 128 MB.

Displaying the available memory sections

You can find out if a memory section is online or offline by reading its state attribute. The following example shows how you can get an overview of all available memory sections:

```
# 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:offline
/sys/devices/system/memory/memory5/state:offline
/sys/devices/system/memory/memory6/state:offline
/sys/devices/system/memory/memory7/state:offline
```

Online sections are in use by your Linux instance. An offline section can be free to be added to your Linux instance but it might also be in use by another Linux instance.

Adding memory

You add a hotplug memory section by writing `online` to its sysfs state attribute.

Example: Enter the following command to add a memory section memory5:

```
# echo online > /sys/devices/system/memory/memory5/state
```

Adding the memory section fails, if the memory section is already in use. The state attribute changes to online when the memory section has been added successfully.

Suspend and resume: Do not add hotplug memory if you intend to suspend the Linux instance before the next IPL. Any changes to the original memory configuration prevent suspension, even if you restore the original memory configuration by removing memory sections that have been added. See Chapter 35, “Suspending and resuming Linux,” on page 333 for more information about suspending and resuming Linux.

Removing memory

You remove a hotplug memory section by writing `offline` to its sysfs state attribute.

Avoid removing core memory. The Linux kernel requires core memory to allocate its own data structures.

Example: Enter the following command to remove a memory section memory5:

```
# echo offline > /sys/devices/system/memory/memory5/state
```

The hotplug memory functions first relocate memory pages to free the memory section and then remove it. The state attribute changes to offline when the memory section has been removed successfully.

The memory section is not removed if it cannot be freed completely.

Chapter 26. Large page support

Large page support entails support for the Linux hugetlbfs file system. This virtual file system is backed by larger memory pages than the usual 4 K pages; for System z the hardware page size is 1 MB.

Applications using large page memory will 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.

Setting up large page support

This section describes the parameters that you can use to configure large page support.

Kernel parameters

This section describes how to configure large page support. You configure large page support by adding parameters to the kernel parameter line.

Large page support kernel parameter syntax

```
►►──hugepages=<number>──────────────────────────────────────────►◄
```

where:

number
is the number of large pages to be allocated at boot time.

Note: If you specify more pages than available, Linux will reserve as many as possible. This will most probably leave too few general pages for the boot process and might stop your system with an out-of-memory error.

Working with large page support

This section describes typical tasks that you need to perform when working with large page support.

- The "hugepages=" kernel parameter should be specified with the number of large pages to be allocated at boot time. To read the current number of large pages, issue:

  ```
  cat /proc/sys/vm/nr_hugepages
  ```

- To change the number of large pages dynamically during run-time, write to the /proc file system:

  ```
  echo 12 > /proc/sys/vm/nr_hugepages
  ```

  If there is not enough contiguous memory available to fulfill the request, the maximum number of large pages will be reserved.

- To obtain information about amount of large pages currently available and the large page size, issue:

```
cat /proc/meminfo

...
HugePages_Total: 20
HugePages_Free: 14
Hugepagesize: 1024 KB
...
```

- To see if hardware large page support is enabled (indicated by the word "edat" in the "features" lin), issue:

```
cat /proc/cpuinfo
...
features : esan3 zarch stfle msa ldisp eimm dfp edat
...
```

The large 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.

To make a Java program use the large page feature, specify the Java `-Xlp` option.

Chapter 27. S/390 hypervisor file system

The S/390® hypervisor file system 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 with a full set of attribute files containing the CPU information.

The recommended mount point for the hypervisor file system is `/sys/hypervisor/s390`.

Figure 43 illustrates the file system tree that is created for LPAR.

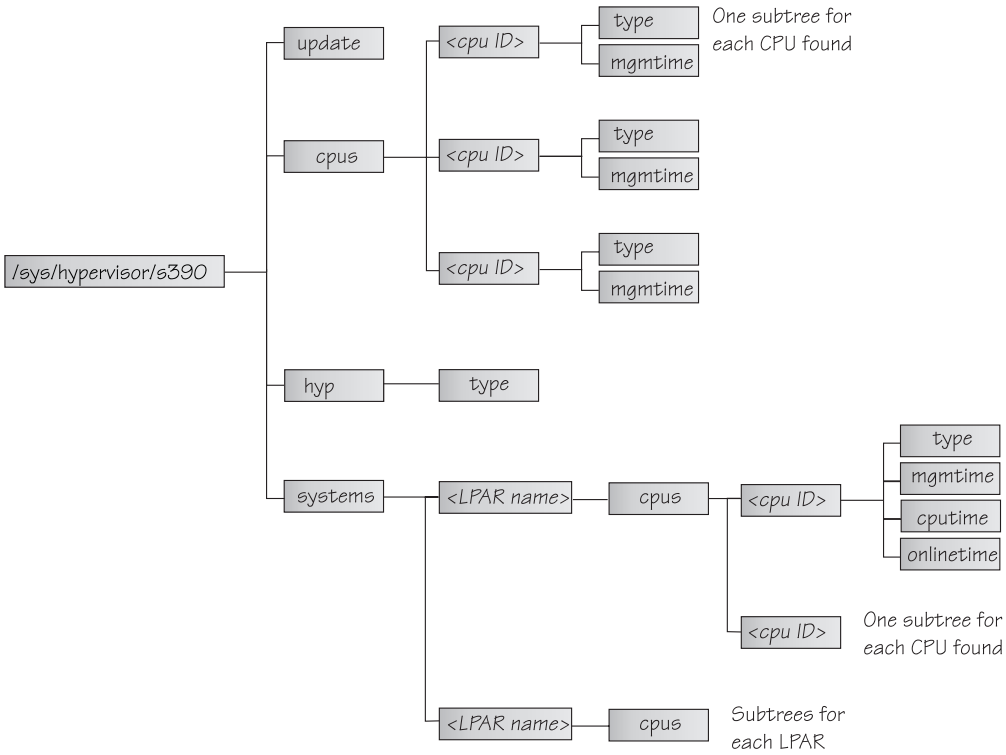


Figure 43. The hypervisor file system for LPAR

LPAR directories and attributes

The directories and attributes have the following meaning for LPARs:

update
: Write only file to trigger an update of all attributes.

cpus/ Directory for all physical CPUs.

cpus/*<cpu ID>*
: Directory for one physical CPU. *<cpu ID>* is the logical (decimal) CPU number.

  type Type name of physical CPU, 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/<cpu ID>/
Directory for the virtual CPUs for one LPAR. The *<cpu ID>* is the logical (decimal) cpu number.

type Type of the logical CPU, such as CP or IFL.

mgmtime
LPAR-management time. Accumulated number of microseconds during which a physical CPU was assigned to the logical cpu and the cpu time was consumed by the hypervisor and was not provided to the LPAR (LPAR overhead).

cputime
Accumulated number of microseconds during which a physical CPU was assigned to the logical cpu and the cpu time was consumed by the LPAR.

onlinetime
Accumulated number of microseconds during which the logical CPU has been online.

Note: For older machines the `onlinetime` attribute might be missing. In general, user space applications should be prepared that attributes are missing or new attributes 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

The directories and attributes have the following meaning for z/VM guests:

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 guests.

systems/*<guest name>*/
Directory for one guest.

systems/<*guest name*>/onlinetime_us
Time in microseconds that the guest has been logged on.

systems/<*guest name*>/cpus/
Directory for the virtual CPUs for one guest.

capped
Flag that shows whether CPU capping is on for guest (0 = off, 1 = soft, 2 = hard).

count Total current virtual CPUs in the guest.

cputime_us
Number of microseconds where the guest virtual CPU was running on a physical CPU.

dedicated
Flag that shows if the guest has at least one dedicated CPU (0 = no, 1 = yes).

weight_cur
Current share of guest (1-10000); 0 for ABSOLUTE SHARE guests.

weight_max
Maximum share of guest (1-10000); 0 for ABSOLUTE SHARE guests.

weight_min
Minimum share of guest (1-10000); 0 for ABSOLUTE SHARE guests.

systems/<*guest name*>/samples/
Directory for sample information for one guest.

cpu_delay
Number of CPU delay samples attributed to the guest.

cpu_using
Number of CPU using samples attributed to the guest.

idle Number of idle samples attributed to the guest.

mem_delay
Number of memory delay samples attributed to the guest.

other Number of other samples attributed to the guest.

total Number of total samples attributed to the guest.

systems/<*guest name*>/mem/
Directory for memory information for one guest.

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

In order to use the file system, it has to be mounted. You can do this either manually 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.

If you want to put hypfs into your /etc/fstab you can add the following line:

```
none <mount point> s390_hypfs defaults 0 0
```

Note that if your z/VM system does not support DIAG 2fc, the s390_hypfs will not be activated and it is not possible to mount the file system. You will see an error message like the following:

```
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, only data for the local guest is provided.

Working with the S/390 hypervisor file system

This section describes typical tasks that you need to perform when working with the S/390 hypervisor file system.

- Defining access rights
- Updating hypfs information

Defining access rights

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 (normally root). It is possible to explicitly define uid and gid 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. For example, you can do this by writing the following:

```
echo 1 > update
```

During the update the whole directory structure is deleted and rebuilt. If a file was open before the update, subsequent reads will return the old data until the file is opened again. Within one second only one update can be done. If within one second more than one update is triggered, only the first one is done and the subsequent write system calls return -1 and errno is set to EBUSY.

If an application wants to ensure consistent data, the following should be done:

1. Read modification time through `stat(2)` from the update attribute.
2. If data is too old, write to the update attribute and go to 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 then go to 2.

Chapter 28. ETR and STP based 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 timing network.

Linux on System z supports external time reference (ETR) and system time protocol (STP) based TOD synchronization. ETR and STP work independently of one another. If both ETR and STP are enabled, Linux might use either to synchronize the clock.

For more information about ETR see the IBM Redbooks® technote at
`www.ibm.com/redbooks/abstracts/tips0217.html`

For information about STP see
`www.ibm.com/systems/z/advantages/pso/stp.html`

Both ETR and STP support are included in the Linux kernel. No special build options are required.

ETR requires at least one ETR unit that is connected to an external time source. For availability reasons, many installations use a second ETR unit. The ETR units correspond to two ETR ports on Linux. Always set both ports online if two ETR units are available.

Attention: Be sure that a reliable timing signal is available before enabling clock synchronization. With enabled clock synchronization, Linux expects regular timing signals and might stop indefinitely to wait for such signals if it does not receive them.

Setting up clock synchronization

This section describes the kernel parameters that you can use to set up synchronization for your Linux TOD clock. These kernel parameters specify the initial synchronization settings. On a running Linux instance you can change these settings through attributes in sysfs (see “Switching clock synchronization on and off” on page 246).

Enabling ETR based clock synchronization

Use the etr= kernel parameter to set ETR ports online when Linux is booted. ETR based clock synchronization is enabled if at least one ETR port is online.

etr syntax

```
      ┌─etr=off──┐
►►────┼──────────┼──────────────────────────────────────────►◄
      ├─etr=on───┤
      ├─etr=port0┤
      └─etr=port1┘
```

The values have the following effect:

on sets both ports online.

port0 sets `port0` online and `port1` offline.

port1 sets `port1` online and `port0` offline.

off sets both ports offline. With both ports offline, ETR based clock synchronization is not enabled. This is the default.

Example: To enable ETR based clock synchronization with both ETR ports online specify:

```
etr=on
```

Enabling STP based clock synchronization

Use the stp= kernel parameter to enable STP based clock synchronization when Linux is booted.

stp syntax

```
►►─┬─stp=off─┬──────────────────────────────────────────────────────────────────────►◄
   └─stp=on──┘
```

By default, STP based clock synchronization is not enabled.

Example: To enable STP based clock synchronization specify:

```
stp=on
```

Switching clock synchronization on and off

You can use the ETR and STP sysfs interfaces to switch clock synchronization on and off on a running Linux instance.

Switching ETR based clock synchronization on and off

ETR based clock synchronization is enabled if at least one of the two ETR ports is online. ETR based clock synchronization is switched off if both ETR ports are offline.

To set an ETR port online, set its sysfs online attribute to “1”. To set an ETR port offline, set its sysfs online attribute to “0”. Enter a command of this form:

```
# echo <flag> > /sys/devices/system/etr/etr<n>/online
```

where *<n>* identifies the port and is either `0` or `1`.

Examples:

- To set ETR port `etr1` offline enter:

```
# echo 0 > /sys/devices/system/etr/etr1/online
```

Switching STP based clock synchronization on and off

To switch on STP based clock synchronization set `/sys/devices/system/stp/online` to “1”. To switch off STP based clock synchronization set this attribute to “0”.

Example: To switch off STP based clock synchronization enter:

```
# echo 0 > /sys/devices/system/stp/online
```

Part 6. Security

This part describes device drivers and features that support security aspects of Red Hat Enterprise Linux 6 for System z.

Newest version: You can find the newest version of this book at
`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the Red Hat Enterprise Linux 6 release notes at
`http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/`

Chapter 29. Generic cryptographic device driver

Some cryptographic processing in Linux can be off-loaded from the CPU and performed by dedicated coprocessors or accelerators. Several of these coprocessors and accelerators are available offering a range of features. The generic cryptographic device driver (z90crypt) is required when one or more of these devices is available in the hardware.

Features

The cryptographic device driver supports a range of hardware and software functions:

Supported devices

The supported coprocessors and accelerators are:

- Crypto Express2 Coprocessor (CEX2C)
- Crypto Express2 Accelerator (CEX2A)
- Crypto Express3 Coprocessor (CEX3C)
- Crypto Express3 Accelerator (CEX3A)

Notes:

1. When Linux is running as a z/VM guest operating system and an accelerator card (CEX2A or CEX3A) is present, any cryptographic coprocessor cards will be hidden.
2. For z/VM 6.1 and 5.4 the PTF for APAR VM64656 is required for support of CEX3C and CEX3A cards. To correct a shared feature problem, the PTF for APAR VM64727 is required. To use the protected key functionality under z/VM and CCA you require APAR VM64793.

For information on how to set up your cryptographic environment on Linux under z/VM, refer to *Security on z/VM*, SG24-7471 and *Security for Linux on System z*, SG24-7728.

Supported facilities

The cryptographic device driver supports these cryptographic operations:

- Clear key encryption and decryption using the Rivest-Shamir-Adleman (RSA) exponentiation operation using either a modulus-exponent (Mod-Expo) or Chinese-Remainder Theorem (CRT) key.
- Secure key encryption and decryption - see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294. You can obtain this book at `www.ibm.com/security/cryptocards/pciecc/library.shtml`.
- Generation of long random numbers, see “Generating and accessing long random numbers” on page 259.

Elements of z90crypt

This section provides information about the software that you need to use z90crypt and the use it makes of cryptographic hardware.

Software components

To run programs that use the z90crypt device driver for clear key encryption, you need:

- The device driver module z90crypt
- The libica library, unless applications call the device driver directly.

  You can use the libica library for generation of RSA key pairs, symmetric and asymmetric encryption, and message hashing.
- The openCryptoki library if applications use the PKCS #11 API.

To run programs that use the z90crypt device driver for secure key encryption, you need:

- The device driver module
- The CCA library, see “The CCA library” on page 256

Figure 44 shows a simplified overview of the software relationships.

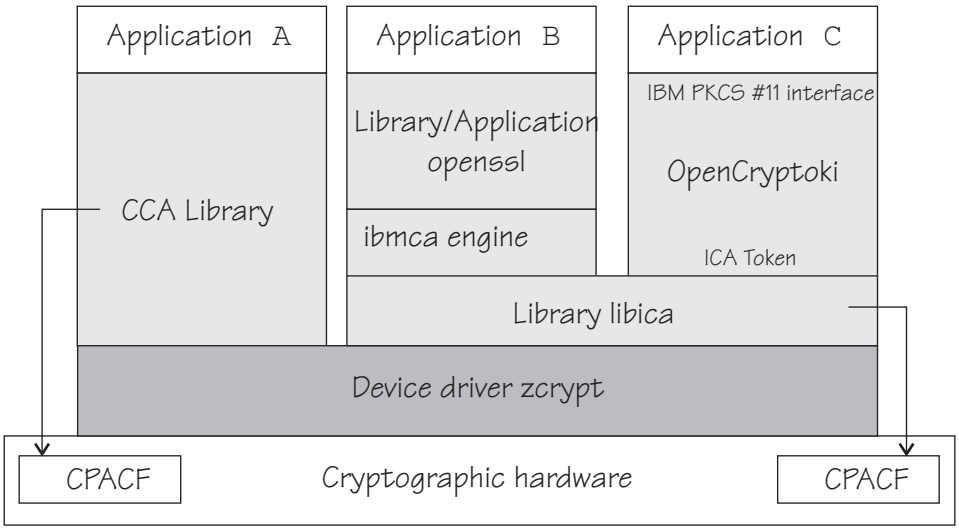


Figure 44. z90crypt device driver interfaces

In Figure 44, applications A, B, and C exemplify three common configurations.

Application A
: uses secure key encryption. See *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294 for more details and specific setups. You can obtain this book at `www.ibm.com/security/cryptocards/pciecc/library.shtml`.

Application B
: uses clear key cryptography through the openssl engine and the libica library. This setup requires the openssl -ibmca RPM.

Application C
: uses clear key cryptography through the openCryptoki PKCS #11 API and the libica library. Java applications need the IBM PKCS #11 provider to access this API.

  You can obtain the provider from developerWorks: Go to `www.ibm.com/developerworks/java/jdk/security/index.html`, click the link for your Java version, and search for “PKCS”.

Independent of the cryptographic device driver, the CCA library and libica can address CP Assist for Cryptographic Function (CPACF).

See “The libica library” on page 256, “The openCryptoki library” on page 256, and “The CCA library” on page 256 for more information about these libraries.

See “Setting up the z90crypt device driver” on page 255 for information about how to set up the different components.

CP Assist for Cryptographic Function (CPACF)

The libica library includes CPACF instructions that allow applications to use hardware-accelerated cryptography. The following functions are included in libica 2:

Table 37. CPACF functions included in libica 2

Function	Name	Supported on System z9	Supported on System z10	Supported on System z196
DES	ica_des_encrypt, ica_des_decrypt	Yes	Yes	Yes
TDES / 3TDS	ica_3des_encrypt, ica_3des_decrypt	Yes	Yes	Yes
SHA-1	ica_sha1	Yes	Yes	Yes
SHA-224	ica_sha224	No	Yes	Yes
SHA-256	ica_sha256	Yes	Yes	Yes
SHA-384	ica_sha384	No	Yes	Yes
SHA-512	ica_sha512	No	Yes	Yes
AES with 128 bit keys	ica_aes_encrypt, ica_aes_decrypt	Yes	Yes	Yes
AES with 192 bit keys	ica_aes_encrypt, ica_aes_decrypt	No	Yes	Yes
AES with 256 bit keys	ica_aes_encrypt, ica_aes_decrypt	No	Yes	Yes
Pseudo Random Number Generation	ica_random_number_generate	Yes	Yes	Yes

See *libica Programmer’s Reference*, SC34-2602 for details about the libica functions.

There is a software fallback provided within libica for CPACF functions (see Table 37) that are not supported on your hardware.

The function prototypes are provided in the header file, ica_api.h. Applications using these functions must link libica and libcrypto. The libcrypto library is available from the OpenSSL package.

See *Security on z/VM*, SG24-7471 for setup information for the openssl engine.

To ascertain what functions are available on your system, use the **icainfo** command, for example:

```
# icainfo
The following CP Assist for Cryptographic Function (CPACF) operations are
supported by libica on this system:
SHA-1:    yes
SHA-256:  yes
SHA-512:  yes
DES:      yes
TDES-128: yes
TDES-192: yes
AES-128:  yes
AES-192:  yes
AES-256:  yes
PRNG:     yes
```

Hardware and software prerequisites

The hardware supports the Crypto Express2 and Crypto Express3 features as follows:

- The CEX2A and CEX2C features are supported on z9 and z10.
- The CEX3A and CEX3C features are supported on z10 (as of October 2009) and z196.

You require the following software:

- For the CEX3C and CEX3A features, you require APAR VM64656 if Linux is running as a z/VM guest operating system on z/VM 6.1 or 5.4. To correct a shared coprocessor problem, APAR VM64727 is required.
- For the secure key cryptographic functions on the CEX2C and CEX3C features, you must use the CCA library. To use the protected key functionality under z/VM and CCA you require APAR VM64793. You can download the CCA library from the IBM cryptographic coprocessor web page at

  `www.ibm.com/security/cryptocards/`

  Note: The CCA library works with 64-bit applications only.
  For information about CEX2C and CEX3C feature coexistence 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 book at `www.ibm.com/security/cryptocards/pciecc/library.shtml`.
- For the clear key cryptographic functions, you should use the libica library. This library is part of the openCryptoki project (see “The libica library” on page 256).

Ensuring the correct length for RSA encryption requests

Cryptographic coprocessors might reject RSA encryption requests for which the numerical value of the data to be encrypted is greater than the modulus. Such requests are then processed in software by libica functions instead and no performance gain can be expected through hardware acceleration.

Performance considerations

Load balancing

To maximize performance, the device driver uses a load balancing algorithm to distribute requests across all available AP bus devices. The algorithm uses a list holding all AP bus devices sorted by increasing utilization. A new request will be submitted to the AP bus device with the lowest utilization. The increased load will move this device further towards the end of the device list after a re-sort is done. When a device completes processing a request, the device will move up towards

the beginning of the device list. To take in account different processing speeds per device type, each device has a speed rating assigned which is also used to calculate the device utilization.

The z90crypt device driver assigns work to cryptographic devices according to device type in the following order:
1. CEX3A
2. CEX2A
3. CEX3C
4. CEX2C

Setting up for the 31-bit compatibility mode

31-bit applications can access the 64-bit z90crypt driver by using the 31-bit compatibility mode.

Note: The CCA library works with 64-bit applications only.

Setting up the z90crypt device driver

This section describes the z90crypt kernel parameters and the z90crypt module, and how to install additional components required by the device driver. This section also describes the z90crypt device node.

For information about how to set up cryptographic hardware on your mainframe, see *zSeries Crypto Guide Update*, SG24-6870.

Monolithic module parameters

This section describes how to load and configure the z90crypt device driver.

z90crypt module syntax

```
                                  ┌─ domain=-1 ──────────┐   ┌─poll_thread=0─┐
►►─modprobe── z90crypt──┤                                ├──┤               ├──────────────►◄
                                  └─ domain=<domain>─────┘   └─poll_thread=1─┘
```

where

<domain>
is an integer in the range from 0 to 15 that identifies the cryptographic domain for the Linux instance.

The default (“domain=-1”) causes the device driver to attempt to autodetect and use the domain index with the maximum number of devices.

You need to specify the domain parameter only if you are running Linux in an LPAR for which multiple cryptographic domains have been defined.

<poll_thread>
is an integer argument and enables a polling thread to increase cryptographic performance. Valid values are 1 (enabled) or 0 (disabled, this is the default).

The z90crypt driver can run with or without polling thread. When running with polling thread one CPU with no outstanding workload is constantly polling the cryptographic cards for finished cryptographic requests. The polling thread will

sleep when no cryptographic requests are being processed. This mode uses the cryptographic cards as much as possible at the cost of blocking one CPU during cryptographic operations.

Without polling thread the cryptographic cards are polled at a much lower rate, resulting in higher latency and reduced throughput for cryptographic requests but without a noticeable CPU load.

Note: If you are running Linux in an LPAR on a z10 EC or later, 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 258.

Refer to the **modprobe** man page for command details.

Examples

- This example loads the z90crypt device driver module if Linux runs in an LPAR with only one cryptographic domain:

```
# modprobe z90crypt
```

- This example loads the z90crypt device driver module and makes z90crypt operate within the cryptographic domain "1":

```
# modprobe z90crypt domain=1
```

The libica library

The libica RPMs are included with Red Hat Enterprise Linux 6.

Use the **icainfo** ("icainfo - Show available libica functions" on page 395) command to find out which libica functions are available to your Linux system. Use **icastats** (see "icastats - Show libica functions" on page 396) to find out how your Linux system uses these libica library functions.

See *libica Programmer's Reference*, SC34-2602 for details about the libica functions.

The openCryptoki library

The openCryptoki RPMs are included with Red Hat Enterprise Linux 6.

Note: To be able to configure openCryptoki (with pkcsconf) user root must be a member of group pkcs11.

See *Security on z/VM*, SG24-7471 for setup information about the openCryptoki library.

The CCA library

Note that two CCA libraries are involved in secure key cryptography; one comes with the CEX2C or CEX3C hardware feature, the other needs to be installed and run on Linux. The two libraries communicate through the device driver.

You can obtain the CCA library from the IBM Cryptographic Hardware website at

`www.ibm.com/security/cryptocards`

The library is available from the software download page for the PCI-X Cryptographic Coprocessor. Install the RPM and see the readme file in the `/doc` subdirectory. The readme explains where files are located, what users are defined, and how to proceed.

See *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294, for additional installation and setup instructions, feature coexistence information, and how to use CCA functions. You can obtain this book at

`www.ibm.com/security/cryptocards/pciecc/library.shtml`

z90crypt device node

User space programs address cryptographic devices through a single device node. In Red Hat Enterprise Linux 6 udev creates the device node `/dev/z90crypt` for you. The device node z90crypt is assigned to the miscellaneous devices.

Working with the z90crypt device driver

Typically, cryptographic devices are not directly accessed by users but through user programs. Some tasks can be performed through the sysfs interface. This section describes the following tasks:

Starting z90crypt

In Red Hat Enterprise Linux 6 you start the z90crypt device driver using the modprobe command:

```
# modprobe z90crypt
```

This command loads the z90crypt device driver module if Linux runs in an LPAR with only one cryptographic domain.

Setting devices online or offline

Use **chzcrypt** to set cryptographic devices online or offline (see “chzcrypt - Modify the zcrypt configuration” on page 370).

Examples

- 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
```

Alternatively, use the sysfs attribute online to set devices online or offline by writing 1 or 0 to it, respectively.

Examples

- To set a cryptographic device with bus device 0x3e online issue:

```
# echo 1 > /sys/bus/ap/devices/card3e/online
```

- To set a cryptographic device with bus device 0x3e offline issue:

```
# echo 0 > /sys/bus/ap/devices/card3e/online
```

- To check the online status of the cryptographic device with bus ID 0x3e issue:

```
# cat /sys/bus/ap/devices/card3e/online
```

  The value is '1' if the device is online and '0' otherwise.

Setting the polling thread

This section applies to IBM mainframe systems prior to z10. For IBM mainframe systems as of z10, see “Using AP adapter interrupts.” If AP interrupts are available, it is not possible to activate the polling thread. See “Using AP adapter interrupts.”

To increase cryptographic performance use the poll_thread attribute. If Linux is running as a guest on z/VM, the poll_thread attribute is disabled by default.

The z90crypt device driver can run in two modes: with or without the polling thread. When running with the polling thread, one CPU with no outstanding workload is constantly polling the cryptographic cards for finished cryptographic requests. The polling thread will sleep when no cryptographic requests are currently being processed. This mode will utilize the cryptographic cards as much as possible at the cost of blocking one CPU during cryptographic operations. Without the polling thread, the cryptographic cards are polled at a much lower rate, resulting in higher latency and reduced throughput for cryptographic requests, but without a noticeable CPU load.

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 0x3e issue:

```
# echo 0 > /sys/bus/ap/devices/card3e/poll_thread
```

Using AP adapter interrupts

To increase cryptographic performance on a IBM System z10 Enterprise Class (z10 EC) system or later, use the AP interrupts mechanism.

If you are running Linux in an LPAR on a z10 EC or later, use AP interrupts instead of the polling mode (described in “Setting the polling thread”). Using AP interrupts instead of the polling frees up one CPU while cryptographic requests are processed.

During module initialization the z90crypt device driver checks whether AP adapter interrupts are supported by the hardware. If so, AP polling is disabled and the interrupt mechanism is automatically used.

To tell whether AP adapter interrupts are used, a sysfs attribute called ap_interrupt is defined. The read-only attribute can be found at the AP bus level.

Example

To read the ap_interrupt attribute for a device 0x3e issue:

```
# cat  /sys/bus/ap/devices/card3e/ap_interrupt
```

The attribute shows 1 if interrupts are used, 0 otherwise.

Using the high resolution polling timer

If you are running Red Hat Enterprise Linux 6 in an LPAR or z/VM, a high resolution timer is used instead of the standard timer. The high resolution timer enables polling at nanosecond intervals rather than the 100 Hz intervals used by the standard timer.

You can set the polling time by using the sysfs attribute poll_timeout. The read-write attribute can be found at the AP bus level.

Example

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
```

Generating and accessing long random numbers

The support of long random numbers enables user-space applications to access large amounts of random number data through a character device.

Before you begin:

- At least one CEX3C or CEX2C feature must be installed in the system and be configured as coprocessor. The CCA library on the CEX3C or CEX2C feature must be version 3.30 or later.
- Under z/VM, at least one CEX3C or CEX2C feature must be configured as DEDICATED to the z/VM guest operating system.
- Automatic creation of the random number character device requires udev.
- The cryptographic device driver z90crypt must be loaded.

If z90crypt detects at least one CEX3C or CEX2C feature capable of generating long random numbers, a new miscellaneous character device is registered and can be found under /proc/misc as hw_random.

Reading from the character device or the symbolic link returns the hardware generated long random numbers. However, do not read excess amounts of random number data from this character device as the data rate is limited due to the cryptographic hardware architecture.

Removing the last available CEX3C or CEX2C feature while z90crypt is loaded automatically removes the random number character device. Reading from the random number character device while all CEX3C or CEX2C features are set offline results in an input/output error (EIO). After at least one CEX3C or CEX2C feature is set online again reading from the random number character device continues to return random number data.

Dynamically adding and removing cryptographic adapters

On an LPAR, you can add or remove cryptographic adapters without the need to reactivate the LPAR after a configuration change. z/VM does not support dynamically adding or removing cryptographic adapters.

Linux attempts to detect new cryptographic adapters and set them online every time a configuration timer expires. Read or modify the expiration time through the sysfs attribute `/sys/bus/ap/config_time`.

Adding or removing of cryptographic adapters to or from an LPAR is transparent to applications using clear key functions. If a cryptographic adapter is removed while cryptographic requests are being processed, z90crypt automatically re-submits lost requests to the remaining adapters. Special handling is required for secure key.

Secure key requests are usually submitted to a dedicated cryptographic coprocessor. If this coprocessor is removed, lost or 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 book at

`www.ibm.com/security/cryptocards/pciecc/library.shtml`

Displaying z90crypt information

Use **lszcrypt** to display status information about your cryptographic devices (see “lszcrypt - Display zcrypt devices” on page 412).

Alternatively, you can use sysfs. Each cryptographic adapter is represented in a sysfs directory of the form

`/sys/bus/ap/devices/card<XX>`

where *<XX>* is the device index for each device. The valid device index range is hex 00 to hex 3f. 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 cryptographic adapter.

Table 38. Cryptographic adapter attributes

Attribute	Explanation
depth	Read-only attribute representing the input queue length for this device.

Table 38. Cryptographic adapter attributes (continued)

Attribute	Explanation
hwtype	Read-only attribute representing the hardware type for this device. The following values are defined: **6** CEX2A cards **7** CEX2C cards **8** CEX3A cards **9** CEX3C cards
modalias	Read-only attribute representing an internally used device bus-ID.
request_count	Read-only attribute representing the number of requests already processed by this device.
type	Read-only attribute representing the type of this device. The following types are defined: • CEX2C • CEX2A • CEX3A • CEX3C

Alternatively, you can enter the following command to read information from the proc interface:

```
# cat /proc/driver/z90crypt/
```

External programming interfaces

This section provides information for those who want to program against the cryptographic device driver or against the available cryptographic libraries.

If you want to circumvent libica and directly access the zcrypt device driver, see he cryptographic device driver header file in the Linux source tree:

`/usr/include/asm-s390/zcrypt.h`

For information about the library APIs, see the following files in the Linux source tree:

- The libica library `/usr/include/ica_api.h`
- The openCryptoki library `/usr/include/opencryptoki/pkcs11.h`
- The CCA (V4.00) library `/opt/IBM/CEX3C/include/csulincl.h`

`ica_api.h`, `pkcs11.h`, and `csulincl.h` are present after the respective library has been installed. Use RPMs opencryptoki-devel-*<version>* and libica-devel-*<version>*.

Chapter 30. Pseudo-random number device driver

The pseudo-random number device driver is a character device driver that provides user-space applications with pseudo-random numbers generated by the pseudo-random number generator of the System z CP Assist for Cryptographic Function (CPACF).

What you should know about the pseudo-random number device driver

The pseudo-random number device provides pseudo-random numbers similar to the Linux pseudo-random number device /dev/urandom but provides a better performance.

Setting up the pseudo-random number device driver

There are no module parameters for the pseudo-random number device driver device driver.

You must load the pseudo-random number module before you can work with it. Use the **modprobe** command to load the module:

```
# modprobe prng
```

Device node

User-space programs access the pseudo-random-number device through a device node, /dev/prandom. Red Hat Enterprise Linux 6 provides udev to create it for you.

The /dev/prandom device node is a character device node (major number 10) with a dynamic minor number. During load, a sysfs folder called class/misc/prandom/ is created, which contains the dev file for getting the major and minor number of the pseudo-random number device.

Making the device node accessible to non-root users

By default, only user root can read from the pseudo-random number device.

If access to the device is restricted to root on your system, add the following udev rule to automatically extend access to the device to other users.

```
KERNEL=="prandom",            MODE="0444", OPTIONS="last_rule"
```

Reading pseudo-random numbers

The pseudo-random number device is read-only. You can obtain random numbers by using any of these function:

- read (/dev/prandom, buffer, bytes)
- cat
- dd

Example: In this example `bs` specifies the block size in bytes for transfer, and `count` 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>
```

Chapter 31. Data execution protection for user processes

The data execution protection feature, similarly to the NX feature on other architectures, provides data execution protection for user processes. The data execution protection prevents, for example, stack-overflow exploits and generally makes a system insensitive to buffer-overflow attacks in user space. Using this feature you can switch the addressing modes of kernel and user space. The switch of the addressing modes is a prerequisite to enable the execute protection.

Features

The data execution protection feature provides the following functions:

- Switch the kernel/user space addressing modes
- Data execution protection for user processes

What you should know about the data execution protection feature

This feature is implemented in software, with some hardware support on IBM System z9-109 EC and BC hardware. The hardware support is an instruction that allows copying data between arbitrary address spaces. Without this hardware support, a manual page-table walk is used for kernel-user-copy functions. A manual page-table walk has a negative performance impact if you enable the feature through the kernel parameter. Selecting the config options does not have this negative effect.

Enabling the data execution protection feature

Use the `noexec` kernel parameter to enable the data execution protection feature.

Kernel parameter for data execution protection

```
      ┌noexec=off┐
►►────┤          ├──────────────────────────────────────────►◄
      └noexec=on─┘
```

If set to `on`, `noexec` enables data execution protection and sets the address mode for user processes to `secondary`. This address mode is required for data execution protection. Do not override this setting with a subsequent `user_mode` parameter (see “user_mode - Set address mode for user space processes” on page 471). If you specify both `noexec` and `user_mode`, the address mode is set according to the parameter specified last.

If set to `off`, `noexec` switches off data execution protection and uses `home` as the address mode for user space processes. This is the default.

Working with the data execution protection feature

This section describes typical tasks that you need to perform when working with the data execution protection feature.

- Enabling and disabling stack execution protection

Enabling and disabling stack execution protection

To prevent stack overflow exploits, the stack of a binary or shared library must be marked as not executable. Do this with the **execstack** user-space tool (part of the prelink package) which sets, clears, or queries the executable stack flag of ELF binaries and shared libraries (GNU_STACK).

Examples

Set and query the executable stack flag (stack is executable):

```
# execstack -s /usr/bin/find
# execstack -q /usr/bin/find
X /usr/bin/find
```

Clear and query the executable stack flag (stack is not executable):

```
# execstack -c /usr/bin/find
# execstack -q /usr/bin/find
- /usr/bin/find
```

To determine the presence of the flag, use the **readelf** command, which is part of the binutils package. To change the flag, however, you need the **execstack** utility.

Set and query the executable stack flag (stack is executable, note the "RWE" meaning "read/write/execute"):

```
# execstack -s /usr/bin/find
# readelf -a /usr/bin/find | grep GNU_STACK -A 1
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RWE    8
```

Clear and query the executable stack flag (stack is not executable, note the "RW" meaning "read/write"):

```
# execstack -c /usr/bin/find
# readelf -a /usr/bin/find | grep GNU_STACK -A 1
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     8
```

Part 7. Booting and shutdown

This section describes device drivers and features that are used in the context of booting and shutting down Linux.

Newest version: You can find the newest version of this book at
`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the Red Hat Enterprise Linux 6 release notes at
`http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/`

Chapter 32. Console device drivers

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

The only interface to a Linux instance in an LPAR before the boot process is completed is the Hardware Management Console (HMC), see Figure 45. 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.

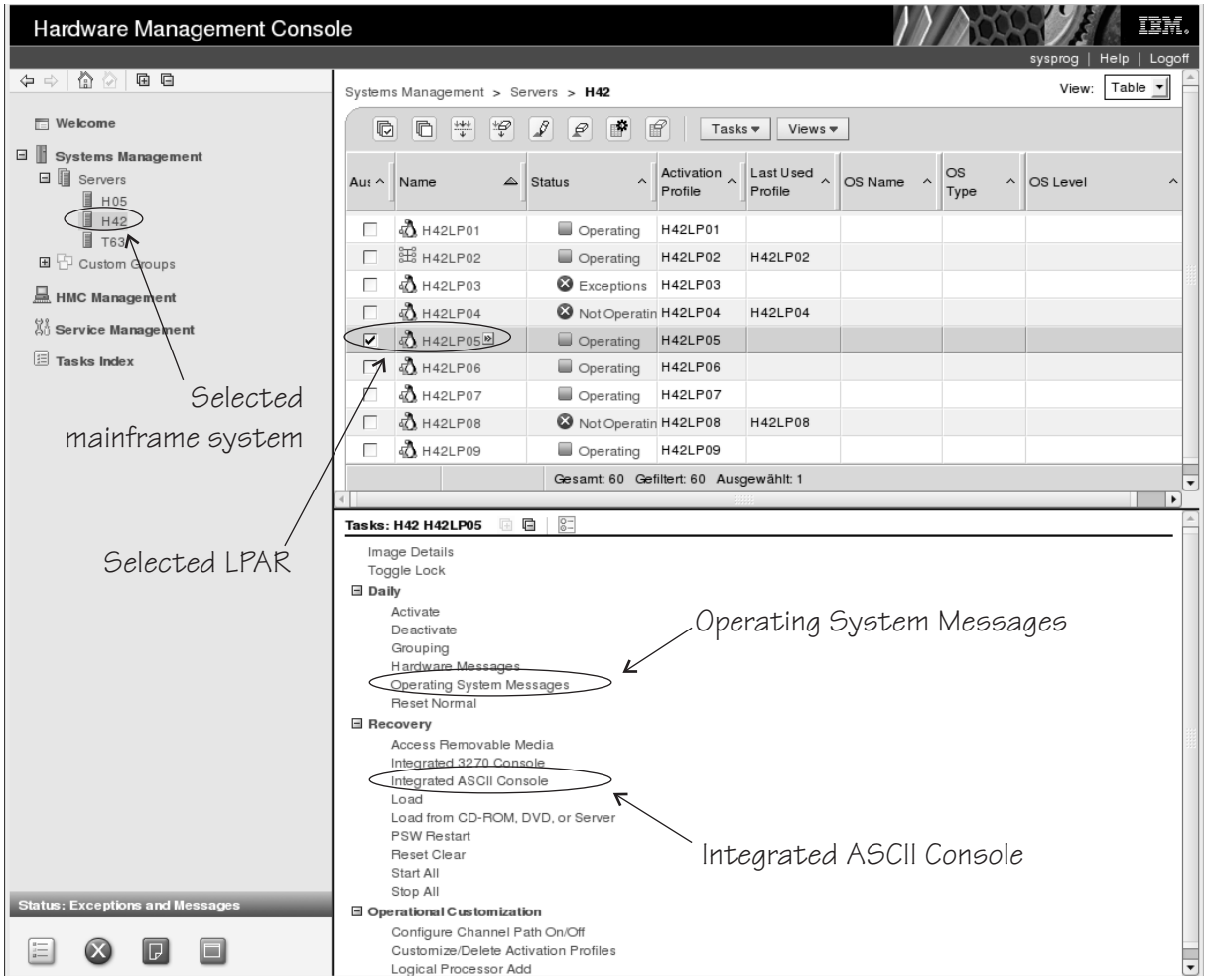


Figure 45. Hardware Management Console

If you run Linux as a z/VM guest operating system, you typically log in to z/VM first, using a 3270 terminal or terminal emulator. 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.

Console features

The console device drivers support the following:

HMC applets

You can use two applets.

Operating System Messages

This is a line-mode terminal. See Figure 46 for an example.

Integrated ASCII Console

This is a full-screen mode terminal.

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

3270 terminal

This can be 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).

The iucvconn program

You can use the iucvconn program on a Linux instance that runs as a z/VM guest operating system to access terminal devices on other Linux instances that also run as guest operating systems of the same z/VM instance.

See *How to Set up a Terminal Server Environment on z/VM*, SC34-2596 for information about the iucvconn program.

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

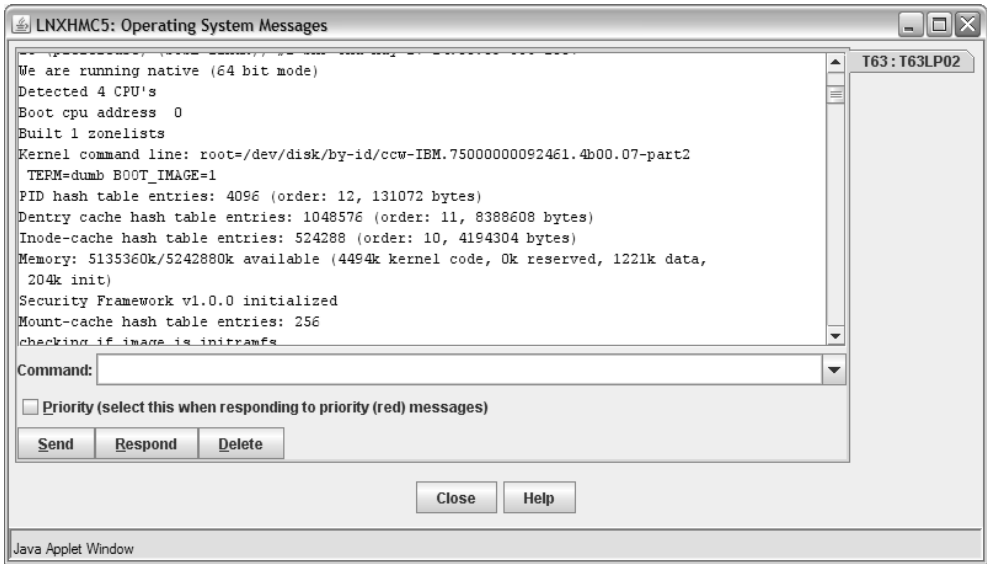


Figure 46. Linux kernel messages on the HMC Operating System Messages applet

What you should know about the console device drivers

This section defines some of the terms used in the context of the console device drivers and provides information about console device names and nodes, about terminal modes, and about how console devices are accessed.

About the terminology

Terminal and *console* have special meanings in Linux.

A Linux terminal
is 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.

The Linux console
is an output device that displays Linux kernel messages.

A mainframe terminal
is any device that gives a user access to operating systems and applications running on the mainframe. This could be a physical device such as a 3270 terminal hardware linked to the mainframe through a controller, or it can be a terminal emulator on a workstation connected through a network. For example, you access z/OS through a mainframe terminal.

The HMC
is a device that gives a system programmer control over the hardware resources, for example the 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.

Console device
in the context of the console device drivers, a device, as seen by Linux, to which Linux kernel messages can be directed.

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

Depending on your setup, a zipl boot menu might be displayed when you IPL. The zipl boot menu is part of the boot loader that loads the Linux kernel. Do not confuse the zipl boot menu with the Linux terminal, which has not been set up at this point. The zipl boot menu is very limited in its functionality, for example, there is no way to specify uppercase letters as all input is converted to lowercase. For more details about booting Linux, see Chapter 34, “Booting Linux,” on page 315. For more details about the zipl boot menu, see Chapter 33, “Initial program loader for System z - zipl,” on page 289.

Device and console names

Each terminal device driver can provide a single console device. Table 39 lists the terminal device drivers with the corresponding device names and console names.

Table 39. 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	tty0.0.009	tty3270
z/VM IUCV HVC device driver	hvc0 to hvc7	hvc0

As shown in Table 39 on page 271, 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 parameter in “Console kernel parameter syntax” on page 275).

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

You require a device node to make a terminal device available to applications, for example to a login program (see “Device nodes”).

Device nodes

Applications access console devices by *device nodes*. For example, with the default conmode settings, udev creates the following device nodes for console devices:

Table 40. Device nodes created by udev

Device driver	On LPAR	On z/VM
SCLP line-mode terminal device driver	/dev/sclp_line0	n/a
SCLP VT220 terminal device driver	/dev/ttysclp0	/dev/ttysclp0
3215 line-mode terminal device driver	n/a	/dev/ttyS0
3270 terminal device driver	/dev/tty0.0.0009	/dev/tty0.0.0009
z/VM IUCV HVC device driver	n/a	/dev/hvc0 to /dev/hvc7

Terminal modes

The Linux terminals 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 allows for advanced interactive capability when using terminal based applications like the vi editor.

On a line-mode terminal, the user first types a full line and then presses Enter to let the system know that a line has been completed. 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 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 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 41 on page 273 summarizes when to expect which terminal mode.

Table 41. 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		
Integrated ASCII Console applet on the HMC	z/VM or LPAR	SCLP VT220 terminal device driver	Full-screen mode
3270 terminal hardware or emulation	z/VM with CONMODE=3215	3215 line-mode terminal device driver	Line mode
	z/VM with CONMODE=3270	3270 terminal device driver	Block mode
iucvconn program	z/VM	z/VM IUCV HVC 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 281 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

Figure 47 shows the possible terminal devices for Linux instances that run directly in an LPAR.

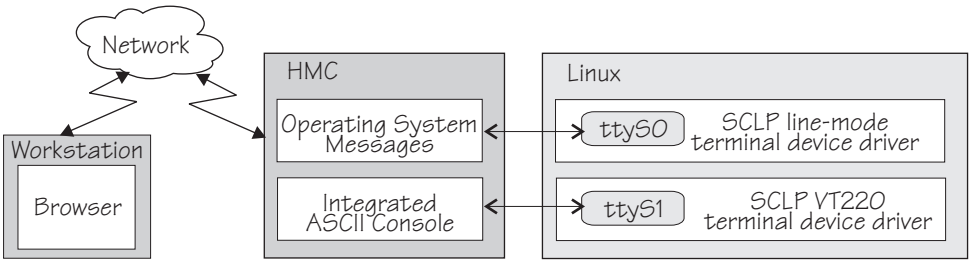


Figure 47. Accessing terminal devices on Linux in an LPAR from the HMC

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

Using the HMC when running Linux as a z/VM guest operating system

If the ASCII system console has been 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 48 on page 274).

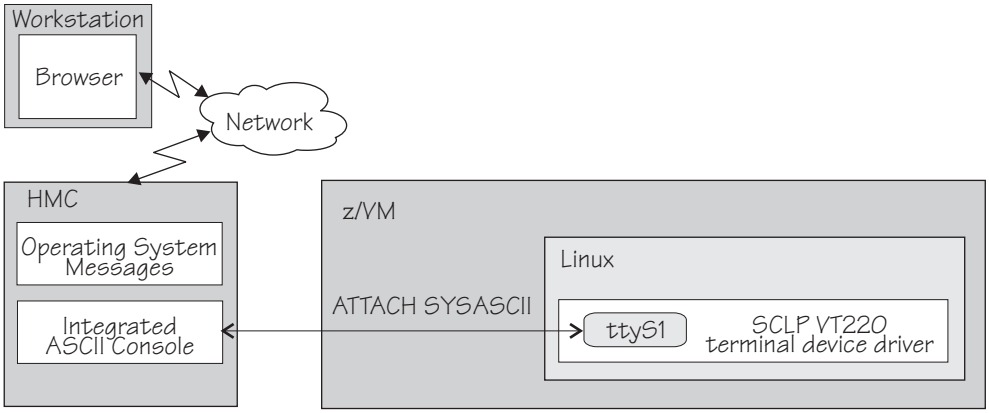


Figure 48. Accessing terminal devices from the HMC when running Linux as a z/VM guest operating system

Using 3270 terminal hardware or a 3270 terminal emulation

For a Linux instance that runs as a z/VM guest operating system, you can use 3270 terminal hardware or a 3270 terminal emulation to access a console device. Figure 49 illustrates how z/VM can handle the 3270 communication.

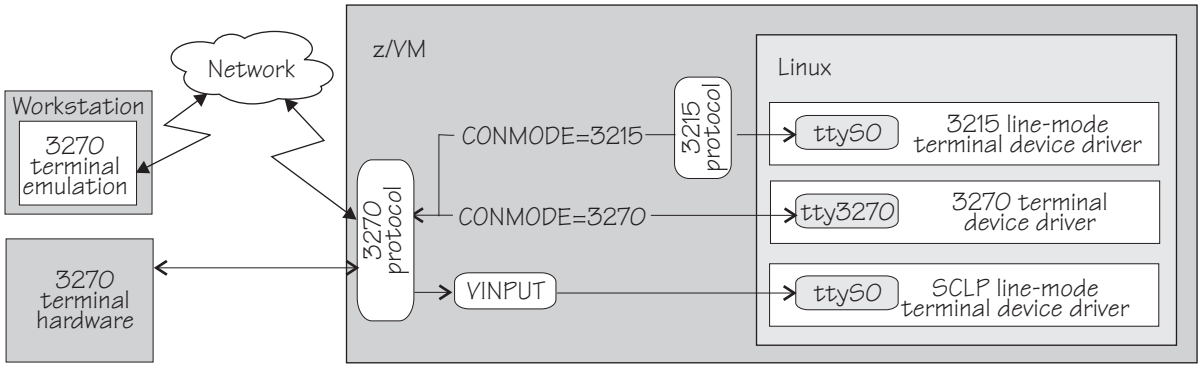


Figure 49. Accessing terminal devices from a 3270 device

Note: Figure 49 shows two console devices with the name ttyS0. Only one of these devices can be present at any one time.

CONMODE=3215
performs a translation between the 3270 protocol and the 3215 protocol and connects the 3270 terminal hardware or emulation to the 3215 line-mode terminal device driver in the Linux kernel.

CONMODE=3270
connects the 3270 terminal hardware or 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 275).

Using iucvconn when running Linux as a z/VM guest operating system

On a Linux instance that runs as a z/VM guest operating system, you can access the terminal devices that are provided by the z/VM IUCV Hypervisor Console (HVC) device driver.

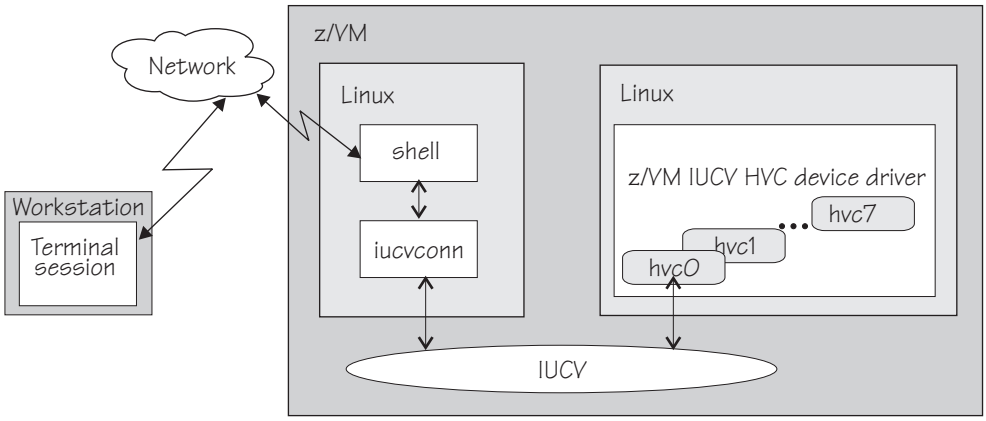


Figure 50. Accessing terminal devices from peer Linux guest operating system

As illustrated in Figure 50, you access the devices with the iucvconn program from another Linux instance that runs as a guest operating system of the same z/VM instance. 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 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

This section describes the kernel parameters that you can use to configure the console device drivers. It also describes settings for initializing terminal devices for user logins.

Console kernel parameter syntax

You can use the conmode= and console= kernel parameters to configure the console device drivers. The hvc_iucv= and hvc_iucv_allow= kernel parameters apply to terminal devices that are provided by the z/VM IUCV HVC device driver only.

Console kernel parameter syntax

```
►►──┬──────────────────────────────┬──────────────────────────────────────►
    │                              ▼                                       │
    │                              console=<console_name>                  │
    └─ conmode=──┬─hwc──┬──┘
                 ├─sclp─┤
                 ├─3215─┤
                 └─3270─┘

►──┬─ hvc_iucv=1──────────────────┬──────────────────────────────────────►
   │                              │
   └─ hvc_iucv=<number_of_devices>─┘

►──┬──────────────────────────────────────────────────────────────────────►◄
   │                    ┌─,──────────────┐
   │                    ▼                │
   └─ hvc_iucv_allow=────<z/VM user ID>──┘──┘
```

Note: If you specify both the conmode= and the console= parameter, specify them in the sequence shown, conmode= first.

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 42 shows how the device driver that is used by default depends on the environment.

Table 42. Default device driver for the line-mode terminal device

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.

You can use the conmode parameter to override the default.

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

You need this specification if you want to use the z/VM VINPUT command (“Using a z/VM emulation of the HMC Operating System Messages applet” on page 285).

3270
specifies the 3270 device driver.

3215
specifies the 3215 device driver.

console=*<console_name>*
specifies which devices are 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.

The preferred console is used as an initial terminal device, beginning at the stage of the boot process when the 'init'-program is called. Messages 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.

Be aware that there is no ttyS0 if you specify conmode=3270.

If you want terminal 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 associated with the terminal device to be activated to receive Linux kernel messages. Of the terminal devices provided by the z/VM IUCV HVC device driver only hvc0 can be activated. Specify the console names as shown in Table 39 on page 271.

hvc_iucv=*<number_of_devices>*
specifies the number of terminal devices provided by the z/VM IUCV HVC device driver. *<number_of_devices>* is an integer in the range 0 to 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
  ```

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 221 for details.

For information about how to access 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 Red Hat Enterprise Linux 6 instance is through a user login that runs, for example, in a telnet or ssh session. See “Terminal modes” on page 272 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 272 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 provided by the 3270 terminal device driver is neither a line-mode terminal nor 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 a terminal for user logins using Upstart

You can use Upstart to allow user logins from a terminal. To enable user logins with Upstart, create an Upstart job file with a file name of the form *<job-name>*.conf and with the following content:

```
start on stopped rc RUNLEVEL=[2345]
stop on runlevel [01]

respawn
exec /sbin/mingetty --noclear <dev>
```

Place the job file in `/etc/init/`.

In the sample file, *<dev>* specifies the device node of the terminal, omitting the leading `/dev/` (see "Device nodes" on page 272). For example, instead of specifying `/dev/sclp_line0`, specify `sclp_line0`.

With mingetty you must explicitly export the TERM environment variable with the terminal name as explained in "Setting up a full-screen mode terminal" on page 278. The terminal name indicates the capabilities of the terminal device. Examples for terminal names are `linux`, `dumb`, `xterm`, or `vt220`.

Instead of mingetty, you can use agetty, which can set the TERM environment variable at startup.

To set the TERM environment variable to `linux` and enable user logins with Upstart create an Upstart job file with the following content:

```
start on stopped rc RUNLEVEL=[2345]
stop on runlevel [01]

respawn
exec /sbin/agetty -L 9600 linux <dev>
```

Example

To enable a device `hvc0` for user logins with mingetty the Upstart job file could, for example, look like this:

```
start on stopped rc RUNLEVEL=[2345]
stop on runlevel [01]

respawn
exec /sbin/mingetty --noclear hvc0
```

Setting up the code page for an x3270 emulation on Linux

If you are accessing z/VM from Linux by using the x3270 terminal emulation, add the following settings to the `.Xdefaults` file to get the correct code translation:

```
! 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

This section describes typical tasks that you need to perform when working with Linux terminals.

- "Using the terminal applets on the HMC" on page 280
- "Accessing terminal devices over z/VM IUCV" on page 280
- "Switching the views of the 3270 terminal device driver" on page 281

Using the terminal applets on the HMC

This section applies to both the line-mode terminal and the full-screen mode terminal on the HMC:

- On an HMC you can only open each applet once.
- Within an LPAR, there can only be 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.
- 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 opening the applet resumes the existing session without a logon.
- Slow performance of the HMC is often due to a busy console or increased network traffic.

The following applies 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 only shows 24 lines and does not provide a scroll bar. To scroll up press Shift+PgUp, to scroll down press Shift+PgDn.

Accessing terminal devices over z/VM IUCV

This section describes how to access hypervisor console (HVC) terminal devices, which are provided by the z/VM IUCV HVC device driver. 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 guest operating systems of the same z/VM instance. The two z/VM guest virtual machines must be configured such that z/VM IUCV communication is permitted between them.

Perform these steps to access a 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 like this:

   ```
   # 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 device hvc0 on a Linux 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 has been activated for Linux kernel messages leaves the device active and the terminal session can be reopened without a login.

Switching the views of the 3270 terminal device driver

The 3270 terminal device driver provides three different views. Use function key 3 (PF3) to switch between the views (see Figure 51).

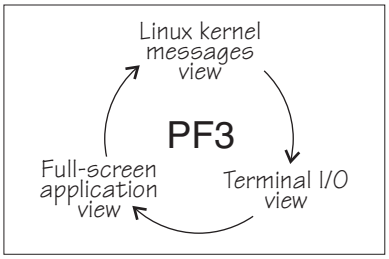


Figure 51. Switching views of the 3270 terminal device driver

The Linux kernel messages view is available only if the terminal device has been 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 only provides 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 needs to 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 4 pages (16 KB) for the Linux kernel messages view and 5 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 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

This section applies to Linux instances that run as z/VM guest operating systems.

The 3270 terminal device driver uses CCW devices and provides them as CCW terminal devices. 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*, SC24-6175 for more information about the DEF GRAF and DIAL commands.

You can use the **chccwdev** command (see “chccwdev - Set a CCW device online” on page 362) 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.

Another problem on line-mode terminals is how to enter a character string without a newline character at the end. Pressing the Enter key adds a newline character to your string which is not expected by some applications.

Table 43 summarizes how you can use the caret character (^) to enter some control characters and to enter strings without appended newline characters.

Table 43. Control and special characters on line-mode terminals

For the key combination	Type this	Usage
`Ctrl+C`	`^c`	Cancel the process that is currently 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. This makes it possible to enter single characters, for example those characters that are needed for yes/no answers in the ext2 file system utilities.

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

Using the magic sysrequest functions

To call the magic sysrequest functions on a line-mode terminal enter the two characters "^-" (caret and hyphen) followed by a third character that specifies the particular function.

You can also call the magic sysrequest functions from the hvc0 terminal device if it is present and has been activated to receive Linux kernel messages. To call the magic sysrequest functions from hvc0 enter the single character Ctrl+o followed by the character for the particular function.

Table 44 provides an overview of the commands for the magic sysrequest functions:

Table 44. Magic sysrequest commands

On line-mode terminals enter	On hvc0 enter	To
`^-b`	`Ctrl+o` `b`	Re-IPL immediately (see "lsreipl - List IPL and re-IPL settings" on page 407).
`^-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 to 9)	`Ctrl+o` followed by a digit (0 to 9)	Set the console log level.

Table 44. Magic sysrequest commands (continued)

On line-mode terminals enter	On hvc0 enter	To
^-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.

Note: In Table 44 on page 283 Ctrl+o means pressing o while holding down the control key.

Table 44 on page 283 lists the main magic sysrequest functions that are known to work on Linux on System z. For a more complete 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 function

From a Linux terminal or a command prompt, enter the following command to activate the magic sysrequest function:

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

Enter the following command to deactivate the magic sysrequest function:

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

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

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

In Red Hat Enterprise Linux 6 the magic sysrequest function is turned on by default. To turn it off using sysctl, issue:

```
kernel.sysrq = 0
```

Triggering magic sysrequest functions from procfs

If you are working from a terminal that does not support a key sequence or combination to call magic sysrequest functions, you can trigger the functions through procfs. Write the character for the particular function to `/proc/sysrq-trigger`.

You can use this interface even if the magic sysrequest functions have not been activated as described in “Activating and deactivating the magic sysrequest function.”

Example: To set the console log level to 1 enter:

```
# echo 1 > /proc/sysrq-trigger
```

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

The preferred terminal devices for Linux instances that run as z/VM guest operating systems are the devices provided by the 3215 or 3270 terminal device drivers. If you need to use the "Operating System Messages" applet emulation, for example, because the 3215 terminal is not operational, you must use the **CP VINPUT** command to prefix any input.

The VINPUT command accesses the ttyS0 terminal device. VINPUT requires that this device is provided by the SCLP line-mode terminal device driver. To be able to use VINPUT, you have to override the default device driver for z/VM environments (see "Console kernel parameter syntax" on page 275).

VINPUT is a z/VM CP command. It can be abbreviated to **VI** but must not be confused with the Linux command **vi**.

If you use the SCLP console driver when running Linux as a z/VM guest operating system (as a line-mode terminal, full-screen mode is not supported), it is important to consider how the input is handled. Instead of writing into the suitable field within the graphical user interface at the service element or HMC, you have to use the **VINPUT** command provided by z/VM. The following examples are written at the input line of a 3270 terminal or terminal emulator (for example, x3270).

If you are in the CP READ mode, omit the leading "#CP" from the commands.

For more information on VINPUT refer to *z/VM CP Commands and Utilities Reference*, SC24-6175.

Priority and non-priority commands

VINPUT commands require a VMSG (non-priority) or PVMSG (priority) specification. Operating systems that honour this specification process priority commands with a higher priority than non-priority commands.

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

Linux does not distinguish 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, 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.

This behavior and the delimiter are adjustable at build-time by editing the driver sources.

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, and the third line shows the command 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 a 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 (see “Using a 3270 terminal in 3215 mode” on page 287). To include the escape character in a command passed to Linux, you need to type it twice.

Example: The following command passes an string in quotation marks to be echoed.

```
#cp vinput pvmsg echo ""%H%ello, here is ""$0
CP VINPUT PVMSG ECHO "%H%ELLO, HERE IS "$0
echo "Hello, here is "$0
Hello, 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 passed to Linux, you need to 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 287, you need to 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 but 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 end of line 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 45 shows the most commonly used settings:

Table 45. Line edit characters

Character	Symbol	Usage
#	`LINEND`	The end of line character allows you to 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 allows you to enter a line edit symbol as a normal character.

To enter a line edit symbol you need to 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 45 on page 287 with the opening bracket character ([) as the delete line character.

- To specify a tab character specify:

  ```
  "|
  ```

- To specify a the double quote 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 received by CP are:

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

Chapter 33. Initial program loader for System z - zipl

zipl can be used to prepare a device for one of the following purposes:

- Booting Linux (as a Linux program loader)
- Dumping

  For more information on the dump tools that **zipl** installs and on using the dump functions, refer to *Using the Dump Tools on Red Hat Enterprise Linux 6*, SC34-2607.
- Loading a data file to initialize a discontiguous saved segment (DCSS)

You can simulate a **zipl** command to test a configuration before you apply the command to an actual device (see "dry-run" on page 292).

zipl supports the following devices:

- 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 (boot and dump devices only)
- SCSI with PC-BIOS disk layout

Usage

zipl base functions

The **zipl** base functions can be invoked with one of the following options on the command line or in a configuration file:

Table 46. zipl base functions

Base function	Command line short option	Command line long option	Configuration file option
Install a boot loader See "Preparing a boot device" on page 293 for details.	`-i`	`--image`	`image=`
Prepare a DASD or tape dump device See "Preparing a DASD or tape dump device" on page 299 for details.	`-d`	`--dumpto`	`dumpto=`
Prepare a list of ECKD volumes for a multi-volume dump See "Preparing a multi-volume dump on ECKD DASD" on page 300 for details.	`-M`	`--mvdump`	`mvdump=`
Prepare a SCSI dump device See "Preparing a dump device on a SCSI disk" on page 302 for details.	`-D`	`--dumptofs`	`dumptofs=`

Table 46. zipl base functions (continued)

Base function	Command line short option	Command line long option	Configuration file option
Prepare a device to load a file to initialize discontiguous named saved segments See “Installing a loader to initialize a discontiguous named saved segment (DCSS)” on page 304 for details.	`-s`	`--segment`	`segment=`
Install a menu configuration See “Installing a menu configuration” on page 305 for details.	`-m`	`--menu`	(None)

zipl modes

zipl operates in one of two modes:

Command-line mode

If a **zipl** command is issued with a base function other than installing a menu configuration (see “Installing a menu configuration” on page 305), the entire configuration must be defined using command-line parameters. See the following base functions for how to specify command-line parameters:

- “Preparing a boot device” on page 293
- “Preparing a DASD or tape dump device” on page 299
- “Preparing a multi-volume dump on ECKD DASD” on page 300
- “Preparing a dump device on a SCSI disk” on page 302
- “Installing a loader to initialize a discontiguous named saved segment (DCSS)” on page 304

Configuration-file mode

If a **zipl** command is issued either without a base function or to install a menu configuration, a configuration file is accessed. See “Configuration file structure” on page 309 for more information.

zipl syntax overview

```
►►─zipl──┬──────┬──┬──────────┬──┬──────────────────────────────────────────┬──┬──────────────────────────────────────┬──►◄
         └─ -V─┘  └─ --dry-run─┘  │ parameters when omitting base function │  │                                      │
                                  ├─ -i──┬─ i_parameters ─┬──────────────────┤  │                                      │
                                  ├─ -d──┤ d_parameters   │                  │  │                                      │
                                  ├─ -M──┤ M_parameters   │                  │  │                                      │
                                  ├─ -D──┤ D_parameters   │                  │  │                                      │
                                  ├─ -s──┤ s_parameters   │                  │  │                                      │
                                  └─ -m──┴─ m_parameters ─┴──────────────────┴──┴──────────────────────────────────────┴
```

parameters when omitting base function:

```
                                  (1)                 (2)
  ┌─ -c /etc/zipl.conf──────┐   ┌─ [default]──────────┐
├─┤                         ├───┤                      ├──────────────────────────────────►
  └─ -c <config_file>───────┘   └─ <configuration>─────┘

►──┬──────────────────────────┬──┬──────────────┬──┬──────┬──┬──────────────────────────┤
   │                   (3)    │  │        (4)   │  │      │  │                          │
   └─ -P <parameters>─────────┘  └─ -a──────────┘  └─ -n─┘  │                          │
```

Notes:

1. You can change the default configuration file with the ZIPLCONF environment variable.
2. If no configuration is specified, **zipl** uses the configuration specified in the [defaultboot] section of the configuration file (see “Configuration file structure” on page 309).
3. In conjunction with a boot configuration or with a SCSI dump configuration only.
4. In conjunction with a boot configuration or a menu configuration only.

Where:

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

<configuration>
specifies a single configuration section in a configuration file.

-P *<parameters>*

can optionally be used to provide:

kernel parameters
in conjunction with a boot configuration section. See “How kernel parameters from different sources are combined” on page 295 for information on how kernel parameters specified with the -P option are combined with any kernel parameters specified in the configuration file.

SCSI system dumper parameters
in conjunction with a SCSI dump configuration section. See “How SCSI system dumper parameters from different sources are combined” on page 304

page 304 for information on how parameters specified with the -P option are combined with any parameters specified in the configuration file.

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

-a in conjunction with 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 created in the target directory.

-n suppresses confirmation prompts that require operator responses to allow unattended processing (for example, when 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 “Parameters” on page 306 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

zipl command line syntax for preparing a boot device

```
►►─zipl── -i <image>──┬─────────────────┬──────────────────────────────────►
                      │,0x10000         │
                      └,<image_addr>────┘

►─┬─ -t <directory>──┬──────────────────────────────────────────────────►
  │                  │                                   (1)
  │                  └─┤ Base device parameters ├────────┘
  └─ -T <tape_node>──┘

►──┬──────────────────────────────────────────────────────────────────►
   │                   ┌─────────────────┐
   └─ -r <ramdisk>─────┬,0x800000──────────┬──┘
                       └,<initrd_addr>─────┘

►──┬──────────────────────────────────────────────────────────────────────►◄
   │                   ┌─────────────────┐   │ -P <parameters>│  │ -a│
   └─ -p <parmfile>────┬,0x1000───────────┬──┘                │  │   │
                       └,<parm_addr>──────┘                   │  │   │
```

Notes:

1 Additional parameters used only if -t specifies a logical device as a target. See “Using base device parameters” on page 297.

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 containing the specified directory *<directory>* or to the specified tape device *<tape_node>*.

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. The default address is 0x800000.

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

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 295 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 -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 created in the target directory.

See “Parameters” on page 306 for a summary of the parameters including the long options you can use on the command line.

Figure 52 summarizes how you can specify a boot configuration within a configuration file section. Required specifications are shown in bold. See “Configuration file structure” on page 309 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>
```

Figure 52. zipl syntax for preparing a boot device — configuration file mode

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 the default address 0x800000. 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:

```
[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 needs to 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 there are two possible sources of kernel parameters that are processed in the order:

1. Kernel parameter file (specified with the -p or --parmfile option)
2. Parameters specified on the command line (specified with the -P or --parameters option)

In configuration file mode there are three possible sources of kernel parameters that are processed in the order:

1. Kernel parameter file (specified with the parmfile= option)
2. Parameters specified in the configuration section (specified with the parameters= option)
3. Parameters 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 a more detailed discussion of various sources of kernel parameters see “Including kernel parameters in a boot configuration” on page 18.

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 located on a logical DASD or SCSI device, zipl cannot detect all required information about the underlying real device or devices and needs additional input.

Logical devices can be, for example, two DASDs combined into a logical mirror volume, or 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, that is, if two blocks on the logical device are adjacent, they need to be adjacent on the underlying real devices as well. This 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 53 on page 296.

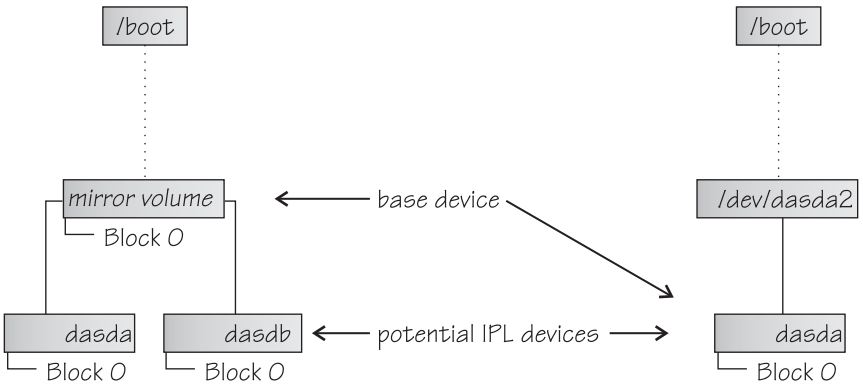


Figure 53. Definition of base device

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 as long as the 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, there is a helper script that automatically provides all the required information to **zipl** for you (see “Using a helper script”).

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 297 and “Writing your own helper script” on page 298.

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. To use the helper script run **zipl** as usual, specifying the parameters for the kernel image, parameter file, initial RAM disk, and target. See “Preparing a boot device” on page 293 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 extends the **zipl** command as shown in “Preparing a boot device” on page 293.

zipl - base device parameters for the command line

Base device parameters:

```
├── --targetbase <targetbase_node>──────────────────────────────────────────►

►── --targettype──┬─┬─ LDL─┬── --targetgeometry <cylinders>,<heads>,<sectors>─┬──────►
                 │ │ CDL─┘                                                   │
                 │ └─ FBA──────────────────────────────────────────────────┘
                 └─ SCSI─┘

►── --targetblocksize <targetblocksize>── --targetoffset <targetoffset>──────────┤
```

The device information you must specify is:

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

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

The device type
of the base device. Valid specifications are:
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

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 54 on page 298 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 54. zipl syntax for preparing a logical device as a boot device — configuration file mode

Example for using base device parameters

The example command in this section 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:

```
[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 the following specifications:

- 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 297.
- 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 297 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 located in `/lib/s390-tools`.

Preparing a DASD or tape dump device

zipl command line syntax for preparing a DASD or tape dump device

```
►►─zipl─ -d  <dump_device>──┬──────────┬──┬────┬──────────────────────►◄
                            └─,<size>──┘  └ -n┘
```

To prepare a DASD or tape dump device you must specify:

The device node <*dump_device*>
of the DASD partition or 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.

Notes:

1. 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.
2. If the dump device is a tape, FBA disk, or ECKD disk with the compatible disk layout (cdl), you do not need to reinstall the dump utility after every dump.

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 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. Refer to *Using the Dump Tools on Red Hat Enterprise Linux 6*, SC34-2607 for details on how to process these dumps.

See “Parameters” on page 306 for a summary of the parameters including the long options you can use on the command line.

Figure 55 summarizes how you can specify a DASD or tape dump configuration in a configuration file. See “Configuration file structure” on page 309 for a more comprehensive discussion of the configuration file.

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

Figure 55. zipl syntax for preparing a DASD or tape dump device — configuration file mode

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:

```
[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 needs to 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
```

Preparing a multi-volume dump on ECKD DASD

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

```
►►─zipl──┬──────┬── -M <dump_device_list>──┬──────────┬──┬────┬──────────►◄
         └─ -f ─┘                          └,<size>───┘  └─ -n─┘
```

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

A file -M *<dump_device_list>*
: containing 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. You can use any block size, even mixed block sizes. However, to speed up the dump process and to reduce wasted disk space, use block size 4096.

Optionally, you can also specify:

An option -f or --force
: to force that no signature checking will take place when dumping. Any data on all involved partitions will be overwritten without warning.

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 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. Refer to *Using the Dump Tools on Red Hat Enterprise Linux 6*, SC34-2607 for details on how to process these dumps.

See “Parameters” on page 306 for a summary of the parameters including the long options you can use on the command line.

Figure 56 summarizes how you can specify a multi-volume DASD dump configuration in a configuration file. See “Configuration file structure” on page 309 for a more comprehensive discussion of the configuration file.

```
[<section_name>]
mvdump=<dump_device_list>,<size>
```

Figure 56. 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 sample_dump_conf -n
```

where the sample_dump_conf file contains the two partitions separated by line breaks:

```
/dev/dasdc1
/dev/dasdd1
```

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

```
[multi_volume_dump]
mvdump=sample_dump_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 needs to 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
```

Preparing a dump device on a SCSI disk

Before you start: At least one partition, the *target partition*, must be available to **zipl**.

zipl command line syntax for preparing a SCSI dump device

```
►►─zipl─ -D <dump_partition>──┬──────────┬─ -t <directory>──────────────►
                              └─,<size>──┘

►─┬──────────────────────┬─┬──────────────────────┬─────────────────────►◄
  └─ -P <parameters>─────┘ └─ -p <parmfile>───────┘
```

The target partition contains the target directory and is accessed to load the SCSI system dumper tool at IPL time. Dumps are written as files to a *dump partition*.

The dump and target partition can but need not be the same partition. Preferably, dump and target partition are two separate partitions.

The target and dump partitions must be formatted with a file system supported by the SCSI Linux system dumper tool. Unlike DASD and tape, creating a dump device on SCSI disk does not destroy the contents of the target partition. Refer to *Using the Dump Tools on Red Hat Enterprise Linux 6*, SC34-2607 for more details.

To prepare a SCSI disk as a dump device, you must specify:

The dump partition *<dump_partition>*
: to which the dumps are written.

A target *<directory>*
: to which the SCSI system dumper components are written. **zipl** uses the target directory to determine the dump device (target partition).

Optionally, you can also specify:

SCSI system dumper parameters
: You can specify parameters *<parameters>* directly on the command line. Instead or in addition, you can specify a location *<parmfile>* of a parameter file on the file system. See "How SCSI system dumper parameters from different sources are combined" on page 304 for a discussion of how multiple parameter specifications are combined.

dump_dir=/*<directory>*
: Path to the directory (relative to the root of the dump partition) where the dump file is to be written. This directory is specified with a leading slash. The directory must exist when the dump is initiated.

  Example: If the dump partition is mounted as `/dumps`, and the parameter "dump_dir=/mydumps" is defined, the dump directory would be accessed as "/dumps/mydumps".

  The default is "/" (the root directory of the partition).

dump_compress=gzip|none
: Dump compression option. Compression can be time-consuming on slower systems with a large amount of memory.

  The default is "none".

dump_mode=interactive|auto
Action taken if there is no room on the file system for the new dump file. “interactive” prompts the user to confirm that the dump with the lowest number is to be deleted. “auto” automatically deletes this file.

The default is “interactive”.

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

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 used by the system to be dumped, the resulting dump is incomplete.

See “Parameters” on page 306 for a summary of the parameters including the long options you can use on the command line.

Figure 57 summarizes how you can specify a SCSI dump configuration in a configuration file. Required specifications are shown in bold. See “Configuration file structure” on page 309 for a more comprehensive discussion of the configuration file.

```
[<section_name>]
dumptofs=<dump_partition>
parmfile=<parmfile>,<parm_addr>
parameters=<parameters>
target=<directory>
```

Figure 57. zipl syntax for preparing a SCSI dump device — configuration file mode

Example

The following command prepares a SCSI partition `/dev/sda2` as a dump device and a directory `/boot` as the target directory. Dumps are to be written to a directory mydumps, relative to the mount point. There is to be no compression but instead the oldest dump will be automatically deleted if there is not enough space for the new dump.

```
# zipl -D /dev/sda2 -P 'dumpdir=/mydumps dump_compress=none dump_mode=auto' -t /boot
```

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

```
[dumpscsi]
dumptofs=/dev/sda2
parmeters='dumpdir=/mydumps dump_compress=none dump_mode=auto'
target=/boot
```

In both the command line and configuration file examples the parameter specifications “dump_compress=none dump_mode=auto” could be omitted because they correspond to the defaults.

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

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

How SCSI system dumper parameters from different sources are combined

zipl allows for multiple sources of SCSI system dumper parameters.

In command-line mode there are two possible sources of parameters that are processed in the order:

1. Parameter file (specified with the -p or --parmfile option)
2. Parameters specified on the command line (specified with the -P or --parameters option)

In configuration file mode there are three possible sources of parameters that are processed in the order:

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

Parameters from different sources are concatenated and passed to the SCSI system dumper in one string. If the same parameter is specified in multiple sources, the value that is encountered last is honored. At IPL time, the combined parameter string is loaded to address (0x1000).

Installing a loader to initialize a discontiguous named saved segment (DCSS)

zipl command line syntax for loading a DCSS

```
►►──zipl── -s <segment_file>,<seg_addr>── -t <directory>──────────────────►◄
```

To prepare a device for loading a data file to initialize discontiguous named saved segments, you must specify:

The source file *<segment_file>*
: to be loaded at IPL time.

The segment address *<seg_addr>*
: to which the segment is to be written at IPL time.

A target *<directory>*
: **zipl** installs the boot loader code on the device containing the specified directory *<directory>*.

After the segment has been loaded, the system is put into the *disabled wait state*. No Linux instance is started.

See “Parameters” on page 306 for a summary of the parameters including the long options you can use on the command line.

Figure 58 summarizes how you can specify a file to be loaded to a DCSS within a configuration file section. See “Configuration file structure” on page 309 for a more comprehensive discussion of the configuration file.

```
[<section_name>]
segment=<segment_file>,<seg_addr>
target=<directory>
```

Figure 58. zipl syntax for loading a DCSS — configuration file mode

Example

The following command prepares a device for loading a file `/boot/segment` to a DCSS at address 0x40000000 when IPLed. The boot loader code is written to `/boot`:

```
# zipl -s /boot/segment,0x40000000 -t /boot
```

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

```
[segment]
segment=/boot/segment,0x40000000
target=/boot
```

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

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

Installing 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

```
                                        (1)
                          ┌ -c /etc/zipl.conf──────┐
►►──zipl── -m <menu_name>─┤                        ├──┬──────────────────────►◄
                          └ -c <config_file>───────┘  └ -a──┘
```

Notes:

1 You can change the default configuration file with the ZIPLCONF 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.

-a or **--add-files**
specifies that the kernel image file, parmfile, and initial RAM disk image are added to the bootmap files in the respective target directories rather than being referenced. 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 created in the target directory.

Example

Using the example of a configuration file in “Example” on page 311, you could install a menu configuration with:

```
# zipl -m menu1
```

Parameters

This section provides an overview of the options and how to specify them on the command line or in the configuration file.

Command line short option Command line long option Configuration file option	Explanation
-a **--add-files** n/a	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 created in the target directory.
-c *<config_file>* **--config**=*<config_file>* n/a	Specifies the configuration file. You can change the default configuration file `/etc/zipl.conf` with the environment variable ZIPLCONF.
<configuration> n/a n/a	Specifies a configuration section to be read and processed from the configuration file.
-d *<dump_device>*[,*<size>*] **--dumpto**=*<dump_device>*[,*<size>*] **dumpto**=*<dump_device>*[,*<size>*]	Specifies the DASD 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 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 DASD or tape dump device” on page 299 and *Using the Dump Tools on Red Hat Enterprise Linux 6*, SC34-2607 for details.

Command line short option Command line long option Configuration file option	Explanation
-D *<dump_partition>*[,*<size>*] or **--dumptofs**=*<dump_partition>*[,*<size>*] **dumptofs**=*<dump_partition>*[,*<size>*]	Specifies the partition to which a SCSI dump file is to be written. This partition must be formatted with a file system supported by the SCSI Linux system dumper tool (for example, ext2 or ext3). The dump partition must be on the same physical SCSI disk as the target partition. It can but need not be the partition that also contains the target directory (target partition). 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 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 a SCSI disk” on page 302 and *Using the Dump Tools on Red Hat Enterprise Linux 6*, SC34-2607 for details.
-h **--help** n/a	Displays help information.
-i *<image>*[,*<image_addr>*] **--image=***<image>*[,*<image_addr>*] **image=***<image>*[,*<image_addr>*]	Specifies the location of the Linux kernel image on the file system and, optionally, in memory after IPL. The default memory address is 0x10000. See “Preparing a boot device” on page 293 for details.
-m *<menu_name>* **--menu**=*<menu_name>* n/a	Specifies the name of the menu that defines a menu configuration in the configuration file (see “Menu configurations” on page 310).
-M *<dump_device_list>*[,*<size>*] **--mvdump**=*<dump_device_list>*[,*<size>*] **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 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 300 and *Using the Dump Tools on Red Hat Enterprise Linux 6*, SC34-2607 for details.
-n **--noninteractive** n/a	Suppresses all confirmation prompts (for example, when preparing a DASD or tape dump device).

Command line short option Command line long option Configuration file option	Explanation
-p *<parmfile>*[,*<parm_addr>*] **--parmfile**=*<parmfile>*[,*<parm_addr>*] **parmfile**=*<parmfile>*[,*<parm_addr>*]	In a boot configuration, specifies the location of a kernel parameter file. In a SCSI dump configuration, specifies the location of a parameter file with SCSI system dumper parameters (see "Preparing a dump device on a SCSI disk" on page 302). You can specify multiple sources of kernel or SCSI system dumper parameters. See "How SCSI system dumper parameters from different sources are combined" on page 304 and "How kernel parameters from different sources are combined" on page 295 for more information. The optional *<parm_addr>* specifies the memory address where the combined kernel parameter list is to be loaded at IPL time. This specification is ignored for SCSI dump configuration, SCSI system dumper parameters are always loaded to the default address 0x1000.
-P *<parameters>* **--parameters**=*<parameters>* **parameters**=*<parameters>*	In a boot configuration, specifies kernel parameters. In a SCSI dump configuration, specifies SCSI system dumper parameters (see "Preparing a dump device on a SCSI disk" on page 302) 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 quotes (') or double quotes ("). You can specify multiple sources of kernel or SCSI system dumper parameters. See "How SCSI system dumper parameters from different sources are combined" on page 304 and "How kernel parameters from different sources are combined" on page 295 for more information.
-r *<ramdisk>*[,*<initrd_addr>*] **--ramdisk**=*<ramdisk>*[,*<initrd_addr>* **ramdisk**=*<ramdisk>*[,*<initrd_addr>*	Specifies the location of the initial RAM disk (initrd) on the file system and, optionally, in memory after IPL. The default memory address is 0x800000.
-s *<segment_file>*,*<seg_addr>* or **--segment**=*<segment_file>*,*<seg_addr>* **segment**=*<segment_file>*,*<seg_addr>*	Specifies the segment file to load at IPL time and the memory location for the segment. See "Installing a loader to initialize a discontiguous named saved segment (DCSS)" on page 304 for details.
-t *<directory>* **--target**=*<directory>* **target**=*<directory>*	Specifies the target directory where **zipl** creates boot-relevant files. The boot loader is installed on the disk containing the target directory. For a SCSI dump device, this partition must have been formatted with a file system supported by the SCSI system dumper (for example, ext2 or ext3).
none **--targetbase**=*<targetbase_node>* **targetbase**=*<targetbase_node>*	For logical boot devices, specifies the device node of the base device, either 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 297 for details.

Command line short option Command line long option Configuration file option	Explanation
none **--targetblocksize**=*<targetblocksize>* **targetblocksize**=*<targetblocksize>*	For logical boot devices, specifies the bytes per block of the base device. See “Using base device parameters” on page 297 for details.
none **--targetgeometry**=*<cylinders>,<heads>,<sectors>* **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 297 for details.
none **--targetoffset**=*<targetoffset>* **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 297 for details.
none **--targettype**=*<type>* **targettype**=*<type>*	For logical boot devices, specifies the device type of the base device. See “Using base device parameters” on page 297 for details.
-T *<tape_node>* **--tape**=*<tape_node>* **tape**=*<tape_node>*	Specifies the tape device where **zipl** installs the boot loader code.
-v **--version** n/a	Prints version information.
-V **--verbose** n/a	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.

Configuration file structure

A configuration file contains:

[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 beginning with '#' 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 consists of the section identifier **[defaultboot]** followed by a single option line. The option line specifies one of these mutually exclusive options:

default=*<section_name>*
: where *<section_name>* is one of the IPL configurations described in the configuration file. If the 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 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 293 for details.

dumpto=*<dump_device>*
: Defines a DASD or tape dump configuration. See “Preparing a DASD or tape dump device” on page 299 for details.

mvdump=*<dump_device_list>*
: Defines a multi-volume DASD dump configuration. See “Preparing a multi-volume dump on ECKD DASD” on page 300 for details.

dumptofs=*<dump_partition>*
: Defines a SCSI dump configuration. See “Preparing a dump device on a SCSI disk” on page 302 for details.

segment=*<segment_file>*
: Defines a DCSS load configuration. See “Installing a loader to initialize a discontiguous named saved segment (DCSS)” on page 304 for details.

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 configuration file and 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 configuration file. 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>*
in conjunction with 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>*
in conjunction with 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.

Example

Figure 59 on page 312 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
# Separate IPL and dump partitions
[dumpscsi]
target=/boot
dumptofs=/dev/sda2
parameters="dump_dir=/mydumps dump_compress=none dump_mode=auto"

# 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

# Configuration for initializing a DCSS
[segment]
segment=/boot/segment,0x800000
target=/boot
```

Figure 59. /etc/zipl.conf example

The following commands assume that the configuration file of our 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 305):

```
# 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 319 and "Example for a DASD menu configuration (LPAR)" on page 326 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 will load 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 will be created uncompressed in directory /mydumps on the dump partition. If space is required, the lowest-numbered dump file in the directory will be deleted.

- Call **zipl** to install a loader to initialize named saved segments:

```
# zipl segment
```

Result: zipl installs segment loader that will load the contents of file `/boot/segment` to address 0x800000 at IPL time and then put the processor into the disabled wait state.

Chapter 34. Booting Linux

This chapter provides a general overview of how to boot Linux in an LPAR or as a z/VM guest. For details on how to define a Linux virtual machine, see *z/VM Getting Started with Linux on System z*, SC24-6194, the chapter on creating your first Linux virtual machine.

IPL and booting

On System z, you usually start booting Linux by performing an Initial Program Load (IPL). Figure 60 summarizes the main steps.

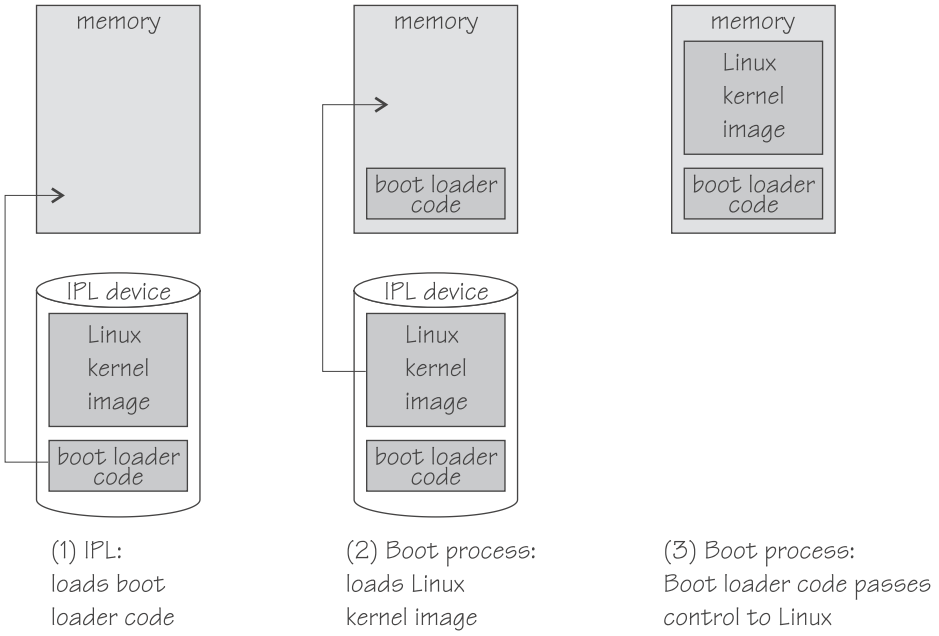


Figure 60. IPL and boot process

The IPL process accesses the IPL device and loads the Linux boot loader code to the mainframe memory. The boot loader code then gets control and loads the Linux kernel. At the end of the boot process Linux gets control.

If your Linux instance is to run in an LPAR, you can circumvent the IPL and use the service element (SE) to copy the Linux kernel to the mainframe memory (see "Loading Linux from a DVD or from an FTP server" on page 326).

Apart from starting a boot process, an IPL can also be used for:

- Writing out system storage (dumping)

  Refer to *Using the Dump Tools on Red Hat Enterprise Linux 6*, SC34-2607 for more information on dumps.
- Loading a discontiguous saved segment (DCSS)

  Refer to *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594 for more information on DCSSs.

You can find the latest copies of these documents on developerWorks at:

`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

The **zipl** tool allows you to prepare DASD, SCSI, and tape devices as IPL devices for booting Linux, for dumping, or for loading a DCSS. See Chapter 33, “Initial program loader for System z - zipl,” on page 289 for more information on **zipl**.

Control point and boot medium

The control point from where you can start the boot process depends on the environment where your Linux is to run. If your Linux instance is to run in LPAR mode, the control point is the mainframe's Support Element (SE) or an attached Hardware Management Console (HMC). If your Linux instance is to run as a z/VM guest operating system,the control point is the control program (CP) of the hosting z/VM.

The media that can be used as boot devices also depend on where Linux is to run. Table 47 provides an overview of the possibilities:

Table 47. Boot media

	DASD	tape	SCSI	NSS	z/VM reader	CD-ROM/FTP
z/VM guest	✓	✓	✓	✓	✓	
LPAR	✓	✓	✓			✓

DASDs, tapes on channel-attached tape devices, and SCSI device that are attached through an FCP channel can be used for both LPAR and z/VM guests. A SCSI device can be a disk or an FC-attached CD-ROM or DVD drive. Named saved systems (NSS) and the z/VM reader are available only in a z/VM environment.

If your Linux runs in LPAR mode, you can also boot from a CD-ROM drive on the SE or HMC, or you can obtain the boot data from a remote FTP server.

Menu configurations

In Red Hat Enterprise Linux 6, you use **zipl** to prepare a DASD or SCSI boot disk. You can also define a menu configuration. A boot device with a menu configuration can hold the code for multiple boot configurations. For SCSI disks, the menu can also include one or more SCSI 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 319 and “Example for a DASD menu configuration (LPAR)” on page 326). 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 310 for information on how to define menu configurations.

Boot data

Generally, you need the following to boot Linux:

- A kernel image
- Boot loader code
- Kernel parameters
- 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.

Kernel image

On Red Hat Enterprise Linux 6, kernel images are installed into the `/boot` directory and are named `vmlinuz-<version>.s390x`. See *Red Hat Enterprise Linux 6 Installation Guide* for information about where to find the images and how to start an installation.

Boot loader code

Red Hat Enterprise Linux 6 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 3, “Kernel and module parameters,” on page 17, Chapter 33, “Initial program loader for System z - zipl,” on page 289, 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.

For example, booting from DASD requires the DASD device driver. If you want to boot from DASD but the DASD device driver has not been compiled into your kernel, you need to provide the DASD device driver module on an initial RAM disk.

Red Hat Enterprise Linux 6 provides a ramdisk located in `/boot` and named `initramfs-<kernel version>.s390x.img`. When a ramdisk is installed or modified, you must call **zipl** to update the boot record.

Booting a z/VM Linux guest virtual machine

You boot Linux in a z/VM guest virtual machine by issuing CP commands from a CMS or CP session.

This section provides summary information for booting Linux in a z/VM guest virtual machine. For more detailed information about z/VM guest environments for Linux see *z/VM Getting Started with Linux on System z*, SC24-6194.

Using tape

Before you start:

- You need a tape that is prepared as a boot device.

A tape boot device must contain the following in the specified order:

1. Tape boot loader code

   The tape boot loader code is included in the s390-tools package on developerWorks.
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.

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> parm <kernel_parameters>
   ```

   where

   <devno>
   is the device number of the boot device as seen by the guest.

   parm *<kernel_parameters>*
   is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters used by your boot configuration (see “Preparing a boot device” on page 293 for information about the boot configuration).

   See also “Specifying kernel parameters when booting Linux” on page 19.

Using DASD

Before you start:

- You need a DASD boot device prepared with **zipl** (see “Preparing a boot device” on page 293).

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> loadparm <n> parm <kernel_parameters>
```

where:

<devno>
specifies the device number of the boot device as seen by the guest.

loadparm *<n>*
is applicable to menu configurations only. Omit this parameter if you are not working with a menu configuration.

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.

Displaying the menu allows you to specify additional kernel parameters (see “Example for a DASD menu configuration on z/VM”). 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 310 for more details on menu configurations.

parm *<kernel_parameters>*
is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters used by your boot configuration (see “Preparing a boot device” on page 293 for information about the boot configuration).

See also “Specifying kernel parameters when booting Linux” on page 19.

Example for a DASD menu configuration on z/VM

This example illustrates how menu2 in the sample configuration file in Figure 59 on page 312 displays on the z/VM console:

```
00: zIPL v1.8.0 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 <input>'
00:
00: Please choose (default will boot in 30 seconds):
```

You choose a configuration by specifying its configuration number. For example, to boot configuration boot3, issue:

```
#cp vi vmsg 2
```

You can also specify additional kernel parameters by appending them to this command. For example:

```
#cp vi vmsg 2 maxcpus=1 mem=64m
```

Using a SCSI device

A SCSI device can be a disk or an FC-attached CD-ROM or DVD drive.

Before you start: You need a SCSI boot device prepared with **zipl** (see "Preparing a boot device" on page 293).

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. See "Menu configurations" on page 310 for more details about menu configurations.

   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 used by your boot configuration.

<kernel_parameters> must contain ASCII characters only. If characters other then 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 letters in the kernel parameters, run the SET LOADDEV command from a REXX script instead. In the REXX script, use the "address command" statement. See *REXX/VM Reference*, SC24-6221 and *REXX/VM User's Guide*, SC24-6222 for details.

Optional: APPEND
appends kernel parameters to existing SCPDATA. This is the default.

Optional: NEW
replaces existing SCPDATA.

Examples:

- To append kernel parameter `noresume` to the current SCPDATA:

```
#cp set loaddev scpdata 'noresume'
```

- To replace the current SCPDATA with the kernel parameters `resume=/dev/sda2` and `no_console_suspend`:

```
#cp set loaddev scpdata NEW 'resume=/dev/sda2 no_console_suspend'
```

For a subsequent IPL command, these kernel parameters are 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.

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*, SC24-6175 for more information about the SET LOADDEV command.

Using a named saved system

To boot your z/VM guest from an NSS, *<nss_name>*, enter an IPL command of this form:

```
#cp i <nss_name> parm <kernel_parameters>
```

where:

<nss_name>
The NSS name can be one to eight characters long and must consist of alphabetic or numeric characters. Examples of valid names include: 73248734, NSSCSITE, or NSS1234.

parm *<kernel_parameters>*
is an optional 56-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters used by your boot configuration (see "Preparing a boot device" on page 293 for information about the boot configuration).

See also "Specifying kernel parameters when booting Linux" on page 19.

Using the z/VM reader

This section provides 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.

Before you start:

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 6 kernel.
- Kernel parameters (optional)
- Initial RAM disk image (optional)

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 send file facility.
   - An FTP file transfer in binary mode.

   Files that are sent to your reader contain a file header that you need to 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. Optionally, 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 used by your boot configuration (see "Preparing a boot device" on page 293 for information about the boot configuration).

See also "Specifying kernel parameters when booting Linux" on page 19.

Booting Linux in LPAR mode

You can boot Linux in LPAR mode from a Hardware Management Console (HMC) or Support Element (SE). The following description refers to an HMC, but the same steps also apply to an SE.

Booting from DASD, tape, or SCSI

Before you start:

- You need a boot device prepared with **zipl** (see "Preparing a boot device" on page 293).
- For booting from a SCSI boot device, you need to have the SCSI IPL feature (FC9904) installed.

Perform these steps to boot from a DASD, tape, or SCSI boot device:

1. In the left navigation pane of the HMC expand **Systems Management** and **Servers** and select the mainframe system you want to work with. A table of LPARs is displayed in the upper content area on the right.
2. Select the LPAR where you want to boot Linux.
3. In the **Tasks** area, expand **Recovery** and click **Load** (see Figure 61 on page 324).

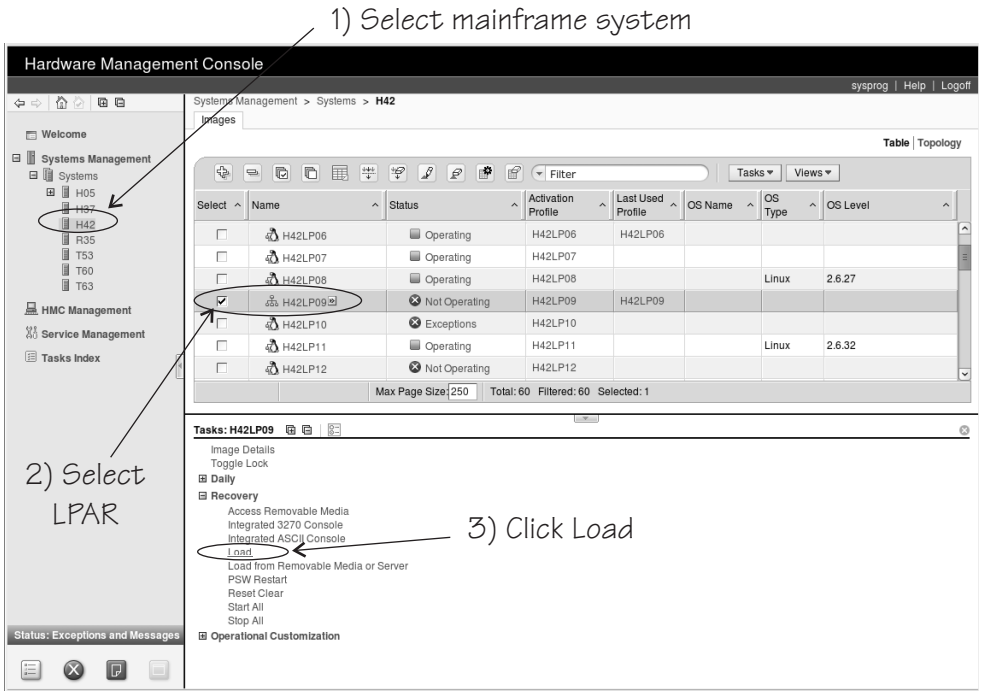


Figure 61. Load task on the HMC

4. Proceed according to your boot device.

   For booting from tape:

   a. Select **Load type** “Normal” (see Figure 62).

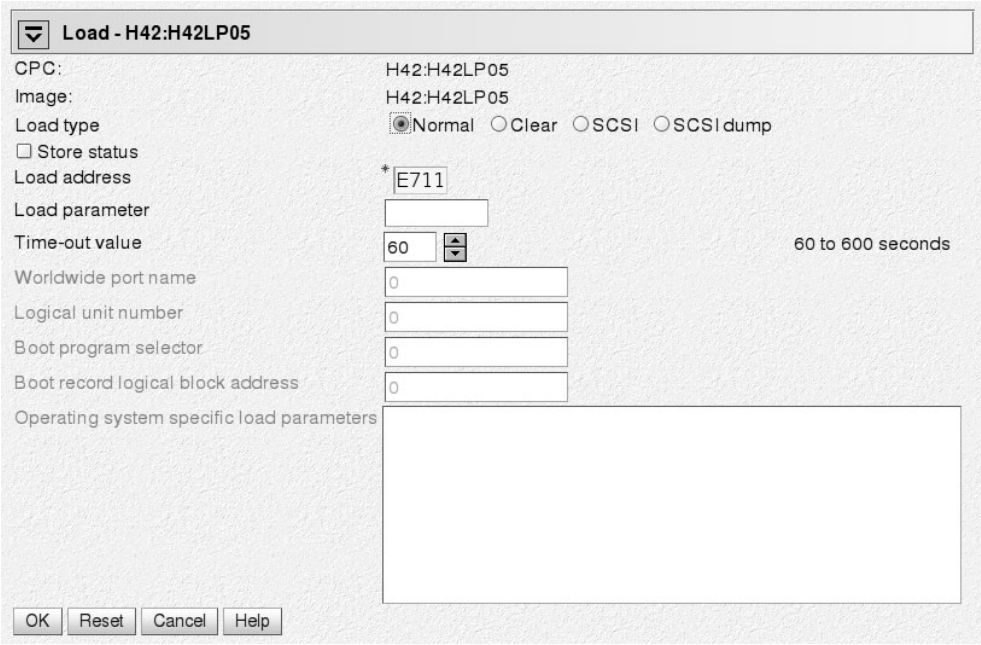


Figure 62. Load panel for booting from tape or DASD

   b. Enter the device number of the tape boot device in the **Load address** field.

   For booting from DASD:

   a. Select **Load type** “Normal” (see Figure 62).

b. Enter the device number of the DASD boot device in the **Load address** field.

c. 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.

   Displaying the menu allows you to specify additional kernel parameters (see “Example for a DASD menu configuration (LPAR)” on page 326). 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 310 for more details on menu configurations.

For booting from a SCSI device:

A SCSI device can be a disk or an FC-attached CD-ROM or DVD drive.

a. Select **Load type** “SCSI” (see Figure 63).

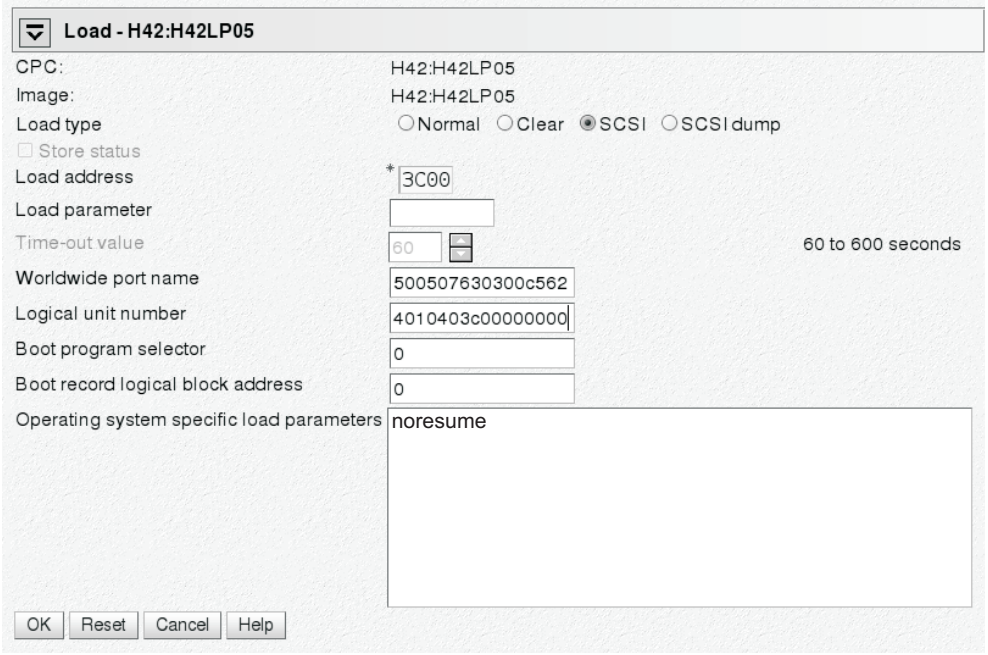


Figure 63. Load panel with SCSI feature enabled — for booting from a SCSI device

b. Enter the device number of the FCP channel through which the SCSI device is accessed in the **Load address** field.

c. Enter the WWPN of the SCSI device in the **World wide port name** field.

d. Enter the LUN of the SCSI device in the **Logical unit number** field.

e. 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. For example, an installation from DVD is typically done with boot program selector 2.

See “Menu configurations” on page 310 for more details on menu configurations.

f. **Optional:** Type kernel parameters in the **Operating system specific load parameters** field. These parameters are concatenated to the end of the existing kernel parameters used by your boot configuration when booting Linux.

   Use ASCII characters only. If you enter characters other then ASCII characters, the boot process ignores the data in the **Operating system specific load parameters** field.

g. Accept the defaults for the remaining fields.

5. Click **OK** to start the boot process.

Check the output on the preferred console (see “Console kernel parameter syntax” on page 275) to monitor the boot progress.

Example for a DASD menu configuration (LPAR)

This example illustrates how menu2 in the sample configuration file in Figure 59 on page 312 displays on the hardware console:

```
zIPL v1.3.0 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 mem=64m
```

Loading Linux from a DVD or from an FTP server

You can use the SE to copy the Linux kernel image directly to your LPARs memory. This process bypasses IPL and does not require a boot loader. The SE performs the tasks that are normally done by the boot loader code. When the Linux kernel has been loaded, Linux is started using restart PSW.

As a source, you can use the SE's CD-ROM/DVD drive or any device on a remote system that you can access through FTP from your SE. If you access the SE remotely from an HMC, you can also use the CD-ROM drive of the system where your HMC runs.

The installation process requires a file with a mapping of the location of installation data in the file system of the DVD or FTP server and the memory locations where the data is to be copied. For Red Hat Enterprise Linux 6 this file is called `generic.ins` and located in the root directory of the file system on the DVD.

1. In the left navigation pane of the HMC expand **Systems Management** and **Servers** and select the mainframe system you want to work with. A table of LPARs is displayed in the upper content area on the right.
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 64).

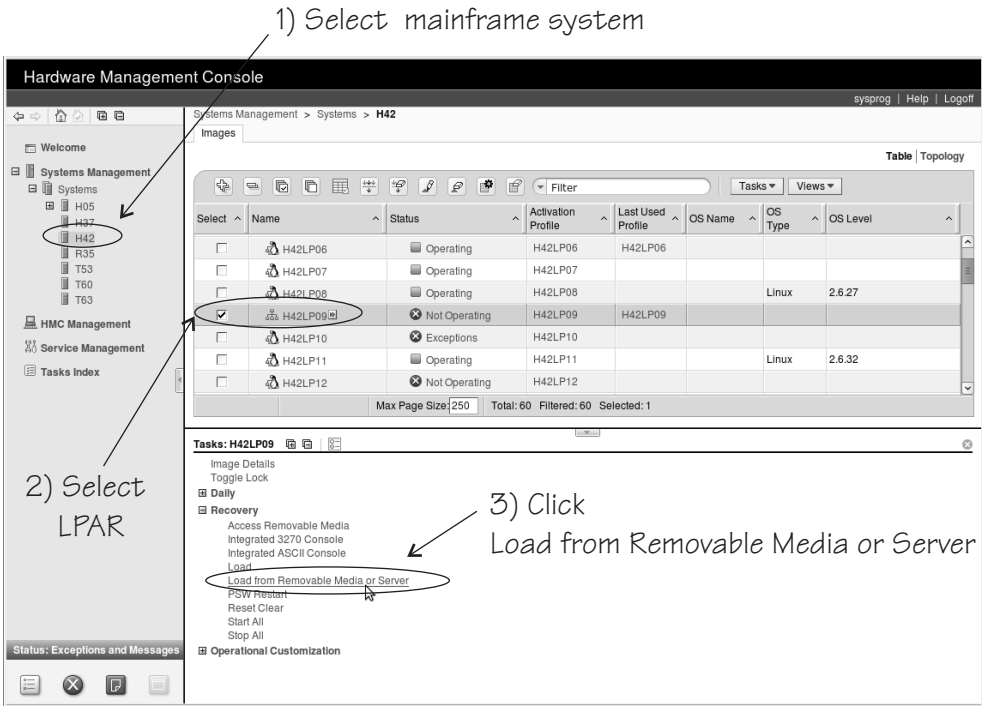


Figure 64. Load from Removable Media or Server task on the HMC

4. Specify the source of the code to be loaded.

   For loading from a CD-ROM drive:

   a. Select **Hardware Management Console CD-ROM/DVD** (see Figure 65 on page 328).

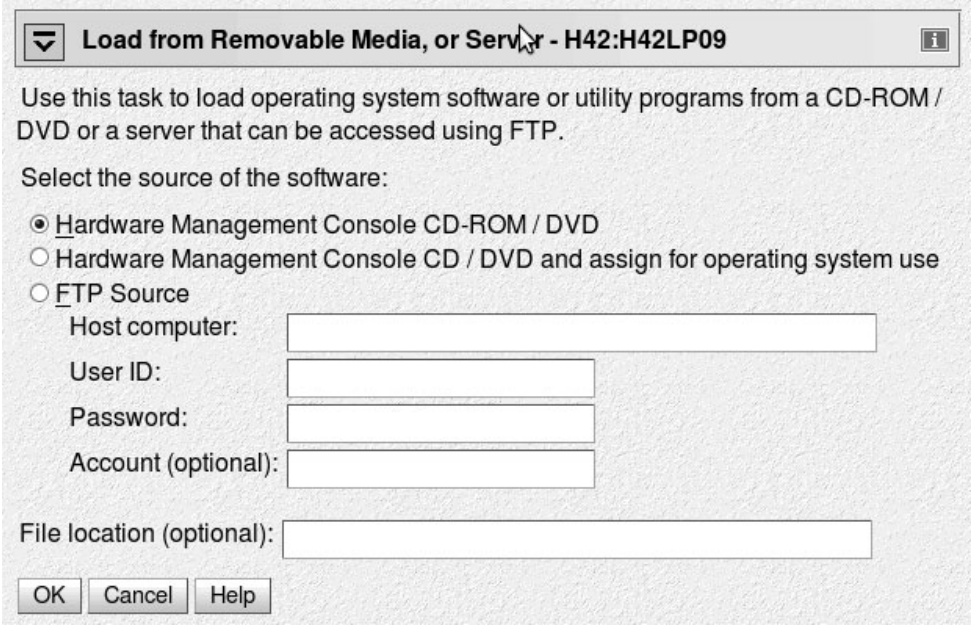


Figure 65. Load from Removable Media or Server panel

b. Leave the **File location** field blank.

For loading from an FTP server:

a. Select the **FTP Source** radio button.

b. Enter the IP address or host name of the FTP server where the install code resides in the **Host computer** entry field.

c. Enter your user ID for the FTP server in the **User ID** 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 for the directory where the `generic.ins` resides in the file location entry field. You can leave this field blank if the file resides in the FTP server's root directory.

5. Click **Continue** to display the “Select Software to Install” panel (Figure 66).

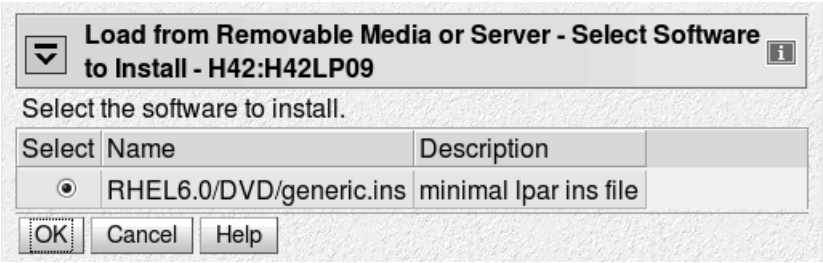


Figure 66. Select Software to Install panel

6. Select the `generic.ins` file.
7. Click **OK** to start loading Linux.

At this point the kernel has started and the Red Hat Enterprise Linux 6 boot process continues.

Displaying current IPL parameters

To display the IPL parameters, use the command **lsreipl** (see "lsreipl - List IPL and re-IPL settings" on page 407). Alternatively, a sysfs user-space interface is available:

```
/sys/firmware/ipl/ipl_type
```

The /sys/firmware/ipl/ipl_type ASCII file contains the device type from which the kernel was booted. The following values are possible:

ccw The IPL device is a CCW device.

fcp The IPL device is an FCP device.

unknown
The IPL device is not known.

Depending on the IPL type, additional files might reside in /sys/firmware/ipl/.

If the device is CCW, the additional files `device` and `loadparm` are present.

`device` Contains the bus ID of the CCW device used for IPL, for example:

```
# cat /sys/firmware/ipl/device
0.0.1234
```

`loadparm`
Contains the eight-character loadparm used for IPL, for example:

```
# cat /sys/firmware/ipl/loadparm
1
```

`parm`

Contains the current z/VM parameter string:

```
# cat /sys/firmware/ipl/parm
noresume
```

See also "Specifying kernel parameters when booting Linux" on page 19.

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 20.

If the device is FCP, a number of additional files are present (also see Chapter 5, "SCSI-over-Fibre Channel device driver," on page 49 for details):

`device` Contains the bus ID of the FCP adapter used for IPL, for example:

```
# cat /sys/firmware/ipl/device
0.0.50dc
```

`wwpn` Contains the WWPN used for IPL, for example:

```
# cat /sys/firmware/ipl/wwpn
0x5005076300c20b8e
```

`lun` Contains the LUN used for IPL, for example:

```
# cat /sys/firmware/ipl/lun
0x5010000000000000
```

br_lba Contains the logical block address of the boot record on the boot device (usually 0).

bootprog
Contains the boot program number.

scp_data
Contains additional kernel parameters that might have been used when booting from a SCSI device (see “Using a SCSI device” on page 320 and “Booting from DASD, tape, or SCSI” on page 323). 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
noresume
```

binary_parameter
Contains all of the above information in binary format.

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 366).

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 ESCON- or FICON-attached DASDs.
- fcp: For FCP SCSI devices, including SCSI disks and CD or DVD drives (Hardware support is required.)
- 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.

Attributes for ccw

The attributes for re-IPL type ccw under `/sys/firmware/reipl/ccw` are:

- device: Device number of the re-IPL device. For example 0.0.4711.

  Note: IPL is possible only from subchannel set 0.

- loadparm: An eight-character loadparm used to select the boot configuration in the zipl menu (if available). The loadparm parameter can only be set when running Linux as a z/VM guest operating system.
- parm: A 64-byte string containing kernel parameters that is concatenated to the boot command line. The PARM parameter can only be set when running Linux as a z/VM guest operating system. See also "Specifying kernel parameters when booting Linux" on page 19.

  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 20.

Attributes for fcp

The attributes for re-IPL type fcp under `/sys/firmware/reipl/fcp` are:

- device: Device number of the fcp adapter used for re-IPL. For example 0.0.4711.

  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 20.

Attributes for nss

The attributes for re-IPL type nss under `/sys/firmware/reipl/nss` are:

- name: Name of the NSS. The NSS name can be 1-8 characters long and must consist of alphabetic or numeric characters. Examples of valid names include: 73248734, NSSCSITE, or NSS1234.
- parm: A 56-byte string containing kernel parameters that is concatenated to the boot command line. (Note the difference in length compared to ccw.) See also "Specifying kernel parameters when booting Linux" on page 19.

  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 20.

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 in case of a kernel panic. See also the **dumpconf** tool described in *Using the Dump Tools on Red Hat Enterprise Linux 6*, SC34-2607 on the developerWorks website at

`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

Examples

- To configure an FCP re-IPL device 0.0.4711 with a LUN 0x4711000000000000 and a WWPN 0x5005076303004711 with an additional kernel parameter `noresume`:

```
# echo 0.0.4711 > /sys/firmware/reipl/fcp/device
# echo 0x5005076303004711 > /sys/firmware/reipl/fcp/wwpn
# echo 0x4711000000000000 > /sys/firmware/reipl/fcp/lun
# echo 0 > /sys/firmware/reipl/fcp/bootprog
# echo 0 > /sys/firmware/reipl/fcp/br_lba
# echo "noresume" > /sys/firmware/reipl/fcp/scp_data
# echo fcp > /sys/firmware/reipl/reipl_type
```

  Note: IPL is possible only from subchannel set 0.

- To set up re-IPL from a Linux NSS with different parameters:
  1. Change to the reipl sysfs directory:

```
# cd /sys/firmware/reipl/
```

  2. Set the reipl_type to nss:

```
# echo nss > reipl_type
```

  3. Setup the attributes in the nss directory:

```
# echo LNXNSS > name
# echo "dasd=0150 root=/dev/dasda1" > parm
```

- To specify additional kernel parameters for Linux re-IPL, follow these steps:
  1. Change to the sysfs directory appropriate for the next re-IPL:

```
# cd /sys/firmware/reipl/$(cat /sys/firmware/reipl/reipl_type)
/sys/firmware/reipl/ccw#
```

  2. Use the echo command to output the parameter string into the parm attribute:

```
# echo "noresume" > parm
```

Chapter 35. Suspending and resuming Linux

With suspend and resume support, you can stop a running Linux on System z instance and later continue operations.

When Linux is suspended, data is written to a swap partition. The resume process uses this data to make Linux continue from where it left off when it was suspended. A suspended Linux instance does not require memory or processor cycles.

Features

Linux on System z suspend and resume support applies to both Linux instances that run as z/VM guest operating systems and Linux instances that run directly in an LPAR.

After a Linux instance has been suspended, you can run another Linux instance in the z/VM guest virtual machine or in the LPAR where the suspended Linux instance was running.

What you should know about suspend and resume

This section describes the prerequisites for suspending a Linux instance and makes you aware of activities that can cause resume to fail.

Prerequisites for suspending a Linux instance

Before a Linux instance is suspended, suspend and resume support checks for conditions that might prevent resuming the suspended Linux instance. You cannot suspend a Linux instance if the check finds prerequisites that are not fulfilled.

The following prerequisites must be fulfilled regardless of whether a Linux instance runs directly in an LPAR or whether it runs as a z/VM guest operating system:

- All tape device nodes must be closed and online tape drives must be unloaded.
- The Linux instance must not have used any hotplug memory since it was last booted.
- 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 in a very short time and do not compromise suspend processing. An example of an I/O request that can take too long to complete is rewinding a tape.

For Linux instances that run as z/VM guest operating systems the following additional prerequisites must be fulfilled:

- No discontiguous saved segment (DCSS) device must be accessed in exclusive-writable mode.

  You must remove all DCSSs of segment types EW, SW, and EN by writing the DCSS name to the sysfs `remove` attribute.

  You must remove all DCSSs of segment types SR and ER that are accessed in `exclusive-writable` mode or change their access mode to `shared`.

  For details see “Removing a DCSS device” on page 209 and “Setting the access mode” on page 206.
- All device nodes of the z/VM recording device driver must be closed.
- All device nodes of the z/VM unit record device driver must be closed.

- No watchdog timer must run and the watchdog device node must be closed.

Precautions while a Linux instance is suspended

There are conditions outside the control of the suspended Linux instance that can cause resume to fail. In particular:

- The CPU configuration must remain unchanged between suspend and resume.
- The data that is written to the swap partition when the Linux instance is suspended must not be compromised.

  In particular, be sure that the swap partition is not used if another operating system instance runs in the LPAR or z/VM guest virtual machine after the initial Linux instance has been suspended.
- If the Linux instance uses expanded storage (XPRAM), this expanded storage must remain unchanged until the Linux instance is resumed.

  If the size or content of the expanded memory is changed before the Linux instance is resumed or if the expanded memory is unavailable when the Linux instance is resumed, resuming fails with a kernel panic.
- If the Linux instance runs as a z/VM guest operating system and uses one or more DCSSs these DCSSs must remain unchanged until the Linux instance is resumed.

  If the size, location, or content of a DCSS is changed before the Linux instance is resumed, resuming fails with a kernel panic.
- If the Linux instance runs as a z/VM guest operating system and the Linux kernel is a named saved system (NSS), this NSS must remain unchanged until the Linux instance is resumed.

  If the size, location, or content of the NSS is changed before the Linux instance is resumed, resuming fails.
- Take special care when replacing a DASD and, thus, making a different device available at a particular device bus-ID.

  You might intentionally replace a device with a backup device. Changing the device also changes its UID-based device nodes. Expect problems if you run an application that depends on UID-based device nodes and you exchange one of the DASD the application uses. In particular, you cannot use multipath tools when the UID changes.
- The SCSI configuration must remain unchanged until the Linux instance is resumed.
- Generally, avoid changes to the real or virtual hardware configuration between suspending and resuming a Linux instance.
- Disks that hold swap partitions or the root file system must be present when resuming the Linux instance.

Potential problems after resuming a Linux instance

Devices might become unavailable or change their device bus-ID after the Linux instance has been suspended. Linux de-registers any devices that are no longer available with the previous bus-ID.

During a scan that follows, available devices are registered with their new device bus-ID. The device that is accessed through a particular device bus-ID might not be the same before Linux is suspended and after Linux is resumed. In particular, disk devices that are accessed by bus-ID might not map to the expected disk space.

Setting up Linux for suspend and resume

This section describes the kernel parameters you can use for setting up suspend and resume support. It also provides information about the swap partition you need to suspend and resume a Linux instance.

Kernel parameters

This section describes the kernel parameters you need to configure support for suspend and resume.

suspend and resume kernel parameter syntax

```
►►──resume=<device_node>──┬──────────────────┬──┬──────────┬──────────►◄
                          └─ no_console_suspend─┘  └─ noresume─┘
```

where:

resume=<*device_node*>
: specifies the standard device node of the swap partition with the data that is required for resuming the Linux instance.

no_console_suspend
: prevents Linux consoles from being suspended early in the suspend process. Without this parameter, you cannot see the kernel messages that are issued by the suspend process.

noresume
: boots the kernel without resuming a previously suspended Linux instance. Add this parameter to circumvent the resume process, for example, if the data written by the previous suspend process is damaged.

Example:

- To use a partition `/dev/dasda2` as the swap partition and prevent Linux consoles from being suspended early in the suspend process specify:

  ```
  resume=/dev/dasda2 no_console_suspend
  ```

Setting up a swap partition

During the suspend process, Linux writes data to a swap partition. This data is required later to resume Linux. Set up a swap partition that is at least the size of the available LPAR memory or the memory of the z/VM guest virtual machine.

Do not use this swap partition for any other operating system that might run in the LPAR or z/VM guest virtual machine while the Linux instance is suspended.

You cannot suspend a Linux instance while most of the memory and most of the swap space are in use. If there is not sufficient remaining swap space to hold the data for resuming the Linux instance, suspending the Linux instance fails. To assure sufficient swap space you might have to configure two swap partitions, one partition for regular swapping and another for suspending the Linux instance. Configure the swap partition for suspending the Linux instance with a lower priority than the regular swap partition.

Use the `pri=` parameter to specify the swap partitions in `/etc/fstab` with different priorities. See the swapon man page for details.

The following example shows two swap partitions with different priorities:

```
# cat /etc/fstab
...
/dev/dasdb1 swap swap pri=-1 0 0
/dev/dasdc1 swap swap pri=-2 0 0
```

In the example, the partition to be used for the resume data is `/dev/dasdc1`.

You can check your current swap configuration by reading `/proc/swaps`.

```
# cat /proc/swaps
Filename          Type         Size       Used     Priority
/dev/dasdb1       partition    7212136    71056    -1
/dev/dasdc1       partition    7212136    0        -2
```

Configuring for fast resume

The more devices are available to a Linux instance, the longer it takes to resume the instance after it has been suspended. With a thousand or more available devices, the resume process can take longer than an IPL. If the duration of the resume process is critical for a Linux instance with many devices, include unused devices in the exclusion list (see “cio_ignore - List devices to be ignored” on page 460).

Suspending a Linux instance

Attention: Only suspend a Linux instance for which you have specified the resume= kernel parameter. Without this parameter, you cannot resume the suspended Linux instance.

Enter the following command to suspend a Linux instance:

```
# echo disk > /sys/power/state
```

On the Linux console you might see progress indications until the console itself is suspended. Most of these messages require log level 7 or higher to be printed. See “Using the magic sysrequest functions” on page 283 about how to set the log level. You cannot see such progress messages if you suspend the Linux instance from an ssh session.

Resuming a suspended Linux instance

Boot Linux to resume a suspended Linux instance. Use the same kernel, initial RAM disk, and kernel parameters that you used to first boot the suspended Linux instance.

You must reestablish any terminal session for HVC terminal devices and for terminals provided by the iucvtty program. You also must reestablish all ssh sessions that have timed out while the Linux instance was suspended.

If resuming the Linux instance fails, boot Linux again with the `noresume` kernel parameter. The boot process then ignores the data that was written to the swap partition and starts Linux without resuming the suspended instance.

Chapter 36. Shutdown actions

Use the **chshut** (see “chshut - Control the system behavior” on page 368) to control the action to take on shutdown.

Alternatively, you can specify the action to take on shutdown by setting the shutdown actions attributes. Figure 67 shows the structure of the `/sys/firmware/` directory.

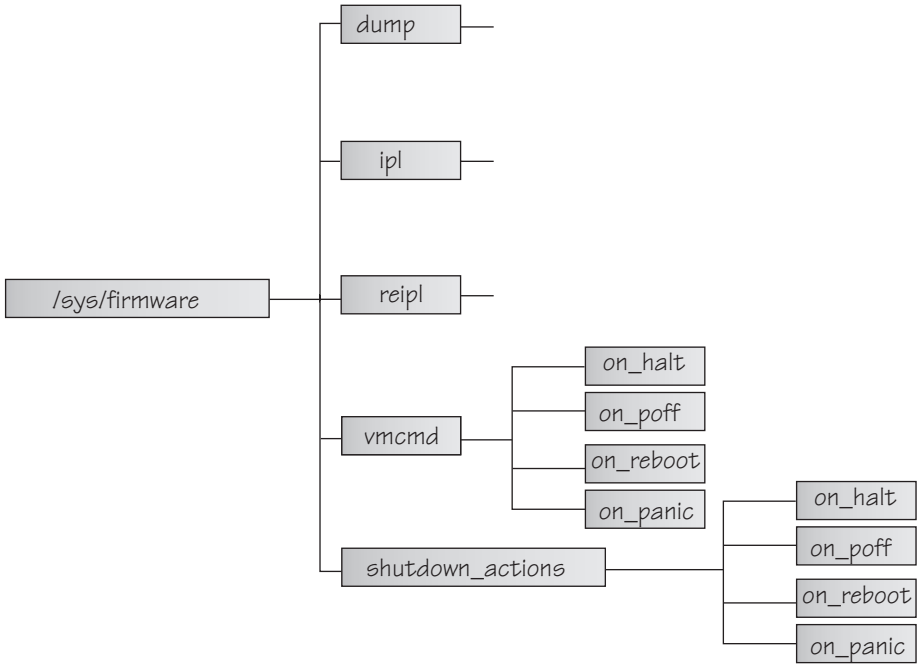


Figure 67. Firmware directory structure

The directories contain the following information:

ipl Information about the IPL device (see “Displaying current IPL parameters” on page 329).

reipl Information about the re-ipl device (see “Rebooting from an alternative source” on page 330).

dump Information about the dump device. Attributes are configured by the dumpconf script. For details, see the description of the dumpconf command in *Using the Dump Tools on Red Hat Enterprise Linux 6*, SC34-2607.

vmcmd
CP commands for halt, power off, reboot, and panic.

shutdown_actions
Configuration of actions in case of halt, poff, reboot and panic.

The shutdown_actions directory contains the following files:

- on_halt
- on_poff
- on_reboot
- on_panic

The shutdown_actions attributes can contain the shutdown actions 'ipl', 'reipl', 'dump', 'stop'', 'vmcmd', or 'dump_reipl'. These values specify what should be done in case of a halt, power off, reboot or kernel panic event. Default for on_halt, on_poff and on_panic is 'stop'. Default for on_reboot is 'reipl'. The attributes can be set by writing the appropriate string into the virtual files.

The vmcmd directory also contains the four files on_halt, on_poff, on_reboot, and on_panic. All theses files can contain CP commands.

For example, if CP commands should be executed in case of a halt, the on_halt attribute in the vmcmd directory must contain the CP commands and the on_halt attribute in the shutdown_actions directory must contain the string 'vmcmd'.

CP commands written to the vmcmd attributes must be uppercase. You can specify multiple commands using the newline character "\n" as separator. The maximum command line length is limited to 127 characters.

For CP commands that do not end or stop the virtual machine, halt, power off, and panic will stop the machine after the command execution. For reboot, the system will be rebooted using the parameters specified under /sys/firmware/reipl.

Note: Red Hat Enterprise Linux 6 maps the halt command to power off. The on_poff action is then performed instead of the on_halt action for the halt command.

Examples

If the Linux **poweroff** command is run, automatically log off the z/VM guest virtual machine:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_poff
# echo LOGOFF > /sys/firmware/vmcmd/on_poff
```

Because Red Hat Enterprise Linux 6 maps the halt command to power off, this action is performed both for **poweroff** and for **halt**.

If the Linux **poweroff** command is executed, send a message to guest MASTER and automatically log off the guest. Do not forget the **cat** command to ensure that the newline is processed correctly:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_poff
# echo -e "MSG MASTER Going down\nLOGOFF" | cat > /sys/firmware/vmcmd/on_poff
```

If a kernel panic occurs, trigger a re-ipl using the IPL parameters under /sys/firmware/ipl:

```
# echo ipl > /sys/firmware/shutdown_actions/on_panic
```

If the Linux **reboot** command is executed, send a message to guest MASTER and reboot Linux:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_reboot
# echo "MSG MASTER Reboot system" > /sys/firmware/vmcmd/on_reboot
```

If a kernel panic occurs, and you want to produce a system dump and trigger a subsequent re-ipl:

```
# echo dump_reipl > /sys/firmware/shutdown_actions/on_panic
```

Note that z/VM CP commands, device addresses, and z/VM user IDs must be uppercase.

Part 8. Diagnostics and troubleshooting

This section describes device drivers and features that are used in the context of diagnostics and problem solving.

Newest version: You can find the newest version of this book at
`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the Red Hat Enterprise Linux 6 release notes at
`http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/`

Chapter 37. Channel measurement facility

The System 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.

Features

The channel measurement facility provides the following features:

- Basic channel measurement format for concurrently collecting data on up to 4096 devices. (Note that 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

You can configure the channel measurement facility by adding parameters to the kernel parameter file.

Channel measurement facility kernel parameters

```
                  ┌─cmf.format=-1──────┐   ┌─cmf.maxchannels=1024──────────┐
►►──────────────────┼────────────────────┼───┼───────────────────────────────┼──────────────────────►◄
                  └─cmf.format=──┬─0─┬─┘   └─cmf.maxchannels=<no_channels>─┘
                                 └─1─┘
```

Note: 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”, assigns a format depending on the hardware. For System z9 and System z10 mainframes the extended format is used.

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 will be 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

This section describes typical tasks you need to perform when working with the channel measurement facility.

- Enabling, resetting, and switching off data collection
- Reading data

Enabling, resetting, and switching off data collection

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 has already been enabled, this 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”).

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

While data collection is enabled for a device, the directories that represent it in sysfs contain a subdirectory, `cmf`, with several read-only attributes. These attributes hold the collected data. 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 48 summarizes the available attributes.

Table 48. Attributes with collected I/O data

Attribute	Value
ssch_rsch_count	An integer representing the ssch rsch count value.

Table 48. Attributes with collected I/O data (continued)

Attribute	Value
sample_count	An integer representing the sample count value.
avg_device_connect_time	An integer representing the average device connect time, in nanoseconds, per sample.
avg_function_pending_time	An integer representing the average function pending time, in nanoseconds, per sample.
avg_device_disconnect_time	An integer representing the average device disconnect time, in nanoseconds, per sample.
avg_control_unit_queuing_time	An integer representing the average control unit queuing time, in nanoseconds, per sample.
avg_initial_command_response_time	An integer representing the average initial command response time, in nanoseconds, per sample.
avg_device_active_only_time	An integer representing the average device active only time, in nanoseconds, per sample.
avg_device_busy_time	An integer representing the average value device busy time, in nanoseconds, per sample.
avg_utilization	A percent value representing the fraction of time that has been spent in device connect time plus function pending time plus device disconnect time during the measurement period.
avg_sample_interval	An integer representing the average time, in nanoseconds, between two samples during the measurement period. Can be “-1” if no measurement data has been collected.
avg_initial_command_response_time	An integer representing 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 representing 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 38. Control program identification

This section applies to Linux instances in LPAR mode only.

If your Linux instance runs in LPAR mode, you can assign names to your Linux instance and sysplex using:

- The `/etc/sysconfig/cpi` configuration file provided by Red Hat Enterprise Linux 6
- The sysfs interface `/sys/firmware/cpi`
- The control program identification (CPI) module, sclp_cpi

The names are used, for example, to identify the Linux instance or the sysplex on the HMC. This section describes how to set the system and sysplex names using sysfs.

Working with the CPI support

This section describes typical tasks that you need to perform when working with CPI support.

- Loading the CPI module
- “Defining a sysplex name” on page 350
- “Defining a system name” on page 350
- “Displaying the system type” on page 350
- “Displaying the system level” on page 350
- “Sending system data to the SE” on page 351

Loading the CPI module

You can provide the system name and the sysplex name as parameters when you load the CPI module, but the preferred method is to use the sysfs interface. When loading the CPI module the following is sent to the SE:

- System name (if provided)
- Sysplex name (if provided)
- System type (automatically set to "LINUX")
- System level (automatically set to the value of LINUX_VERSION_CODE)

CPI module parameter syntax

```
►►─modprobe─ sclp_cpi─┬──────────────────────────┬─┬──────────────────────────────┬──────►◄
                      └─ system_name=<system>─┘ └─ sysplex_name=<sysplex>─┘
```

where:

system_name = *<system>*
specifies an 8-character system name of the following set: A-Z, 0-9, $, @, # and blank. The specification is converted to uppercase.

sysplex_name = *<sysplex>*
specifies an 8-character sysplex name of the following set: A-Z, 0-9, $, @, # and blank. The specification is converted to uppercase.

Defining a system name

Use the attribute system_name in sysfs to specify a system name:

`/sys/firmware/cpi/system_name`

The system name is a string consisting of up to 8 characters of the following set: A-Z, 0-9, $, @, # and blank.

Example:

```
# echo LPAR12 > /sys/firmware/cpi/system_name
```

This attribute is intended for setting the name only. To confirm the current system name, check the HMC.

Defining a sysplex name

Use the attribute sysplex_name in sysfs to specify a sysplex name:

`/sys/firmware/cpi/sysplex_name`

The sysplex name is a string consisting of up to 8 characters of the following set: A-Z, 0-9, $, @, # and blank.

Example:

```
# echo SYSPLEX1 > /sys/firmware/cpi/sysplex_name
```

This attribute is intended for setting the name only. To confirm the current sysplex name, check the HMC.

Displaying the system type

The attribute system_type in sysfs provides the system type:

`/sys/firmware/cpi/system_type`

Example:

```
# cat /sys/firmware/cpi/system_type
LINUX
```

For Red Hat Enterprise Linux the system type is LINUX.

Displaying the system level

The attribute system_level in sysfs provides the operating system version:

`/sys/firmware/cpi/system_level`

The information is displayed in the format:

`0x0000000000aabbcc`

where:

aa kernel version

bb kernel patch level

cc kernel sublevel

Example: Linux kernel 2.6.32 displays as

```
# cat /sys/firmware/cpi/system_level
0x0000000000020620
```

Sending system data to the SE

Use the attribute set in sysfs to send data to the service element:

`/sys/firmware/cpi/set`

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 39. Activating automatic problem reporting

You can activate automatic problem reporting for situations where Linux experiences a kernel panic. Linux then uses the Call Home function to send automatically collected problem data to the IBM service organization through the Service Element. Hence a system crash automatically leads to a new Problem Management Record (PMR) which can be processed by IBM service.

Before you start:

- The Linux instance must run in an LPAR.
- You need a hardware support agreement with IBM to report problems to RETAIN®.

Setting up the Call Home support

To set up the CALL Home support, load the sclp_async module with the modprobe command.

```
# modprobe sclp_async
```

There are no module parameters for sclp_async.

Activating the Call Home support

When the sclp_async module is loaded, you can control it through the sysctl interface or the proc file system.

To activate the support, set the callhome attribute to `1`. To deactivate the support, set the callhome attribute to `0`. Issue a command of this form:

```
# echo <flag> > /proc/sys/kernel/callhome
```

This is equivalent to:

```
# sysctl -w  kernel.callhome=<flag>
```

To persistently enable the callhome feature across reboots, add "`kernel.callhome=1`" to the `/etc/sysctl.conf` file.

Linux cannot check if the Call Home function is supported by the hardware.

Example

To activate the Call Home support issue:

```
# echo 1 > /proc/sys/kernel/callhome
```

To deactivate the Call Home support issue:

```
# echo 0 > /proc/sys/kernel/callhome
```

Chapter 40. Avoiding common pitfalls

This chapter lists some common problems and describes how to avoid them.

Ensuring correct channel path status

Before you perform a planned task on a path like:

- Pulling out or plugging in a cable on a path.
- Configuring a path off or on at the SE.

ensure that you have varied the path offline using:

```
echo off > /sys/devices/css0/chp0.<chpid>/status
```

After the operation has finished and the path is available again, vary the path online using:

```
echo on > /sys/devices/css0/chp0.<chpid>/status
```

If an unplanned change in path availability occurred (such as unplanned cable pulls or a temporary path malfunction), the PIM/PAM/POM values (as obtained through **lscss**) may not be as expected. To update the PIM/PAM/POM values, vary one of the paths leading to the affected devices using:

```
echo off > /sys/devices/css0/chp0.<chpid>/status
echo on > /sys/devices/css0/chp0.<chpid>/status.
```

Rationale: Linux does not always receive a notification (machine check) when the status of a path changes (especially a path becoming online again). To make sure Linux has up-to-date information on the usable paths, path verification is triggered through the Linux vary operation.

Determining channel path usage

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 12 for details.

Configuring LPAR I/O devices

A Linux LPAR should only contain those I/O devices that it uses. Achieve this by:

- Adding only the needed devices to the IOCDS
- Using the cio_ignore kernel parameter to ignore all devices that are not currently in use by this LPAR.

  If more devices are needed later, they can be dynamically removed from the list of devices to be ignored. For a description on how to use the cio_ignore kernel parameter and the `/proc/cio_ignore` dynamic control, see “cio_ignore - List devices to be ignored” on page 460 and “Changing the exclusion list” on page 461.

Rationale: Numerous unused devices can cause:

- Unnecessary high memory usage due to device structures being allocated.

- Unnecessary high load on status changes, since hot-plug handling must be done for every device found.

Using cio_ignore

With cio_ignore, essential devices might have been hidden. For example, 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 guest swapping

If a Linux guest seems to be swapping and not making any progress, you might try to set the timed page pool size and the static page pool size to zero:

```
# 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:

```
# rmmod cmm
```

See Chapter 23, “Cooperative memory management,” on page 225 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 service.

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:

```
# cat /proc/service_levels
VM: z/VM Version 5 Release 2.0, service level 0801 (64-bit)
qeth: 0.0.f5f0 firmware level 087d
```

Booting stops with disabled wait state

On Red Hat Enterprise Linux 6, a processor type check is automatically run at every kernel start up. If the check determines that Red Hat Enterprise Linux 6 is not compatible with the hardware, it stops the boot process with a disabled wait PSW with an address of zero.

If this happens, ensure that you are running Red Hat Enterprise Linux 6 on supported hardware. See the Red Hat Enterprise Linux 6 release notes at

`http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/`

Preparing a dump disk

You might want to consider setting up your system to automatically create a dump after a kernel panic. Configuring and using "dump on panic" has the following advantages:

- You have a dump disk prepared ahead of time.
- You do not have to reproduce the problem since a dump will be triggered automatically right after the failure.

See Chapter 36, “Shutdown actions,” on page 339 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.

To remedy this, try increasing the values for `fast_fail` and `dev_loss_tmo`, for example to 5 seconds and 120 seconds respectively.

Part 9. Reference

This section describes commands, kernel parameters, kernel options, and Linux use of z/VM DIAG calls.

Newest version: You can find the newest version of this book at
`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the Red Hat Enterprise Linux 6 release notes at
`http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/`

Chapter 41. Commands for Linux on System z

This chapter describes commands to configure and work with the Red Hat Enterprise Linux 6 for System z device drivers and features.

For the **zipl** command, see Chapter 33, “Initial program loader for System z - zipl,” on page 289.

Some commands come with an init script or a configuration file or both. These files are installed under `/etc/init.d/` or `/etc/sysconfig/` respectively. You can extract any missing files from the `etc` subdirectory in the s390utils RPM.

Commands described elsewhere:

- For the **zipl** command, see Chapter 33, “Initial program loader for System z - zipl,” on page 289.
- For commands and tools related to taking and analyzing system dumps, see *Using the Dump Tools on Red Hat Enterprise Linux 6*, SC34-2607.
- For commands related to terminal access over IUCV connections, see *How to Set up a Terminal Server Environment on z/VM*, SC34-2596.

Generic command options

The following options are supported by all commands described in this section and, for simplicity, have been 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

```
►►──<command>──┬─┬─ Other command options ─┬─┬──────────────────────────►◄
               │ │                         │ │
               │ ├─-h──────────────────────┤ │
               │ ├─--help──────────────────┤ │
               │ └─--version───────────────┘ │
```

where command can be any of the commands described in this section.

See “Understanding syntax diagrams” on page ix for general information on reading syntax diagrams.

chccwdev - Set a CCW device online

This command is used to set CCW devices (See “Device categories” on page 7) online or offline.

Format

chccwdev syntax

```
►►─chccwdev──┬─ -e ──────────────────┬──┬──────────────────────────────────┬──────────►◄
             ├─ -d ──────────────────┤  │  ┌──────────────────────────────┐│
             ├─ -f ──────────────────┤  ↓  │                              ↓│
             └─ -a <name> = <value>──┘  ─┬─<device_bus_id>──────────────┬─┘
                                         └─<from_device_bus_id>-<to_device_bus_id>─┘
```

Where:

-e or **--online**
sets the device online.

-d or **--offline**
sets the device offline.

-f or **--forceonline**
forces a boxed device online, if this is supported by the device driver.

-a or **--attribute** *<name>=<value>*
sets the attribute specified in *<name>* to the given *<value>*. When *<name>* is “online”, attribute will have the same effect as using the **-e** or **-d** options.

<device_bus_id>
identifies the device to be set online or offline. *<device_bus_id>* is a device number with a leading “0.n.”, where n is the subchannel set ID. Input will be converted to lower case.

<from_device_bus_id>-<to_device_bus_id>
identifies a range of devices. Note that if not all devices in the given range exist, the command will be limited to the existing ones. If you specify a range with no existing devices, you will get an error message.

Examples

- To set a CCW device 0.0.b100 online issue:

```
# chccwdev -e 0.0.b100
```

- Alternatively, using **-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
```

chchp - Change channel path status

Use this 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 as well as 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.

Format

chchp syntax

```
                                  ,
                                  ▼
►►─chchp──┬─ -c ─┬─ 0 ─┬──────────┬─┬─0.<id>──────────┬──────────────►◄
          │      └─ 1 ─┘          │ └─0.<id> -  0.<id>─┘
          ├─ -v ─┬─ 0 ─┬──────────┤
          │      └─ 1 ─┘          │
          └─ -a <key>=<value>─────┘
```

Where:

-c or **--configure** *<value>*
sets the device to configured (1) or standby (0). Note that setting the configured state to standby may cause a currently running I/O operation to be aborted.

-v or **--vary** *<value>*
changes the logical channel-path state to online (1) or offline (0). Note that setting the logical state to offline may cause a currently running I/O operation to be aborted.

-a 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"), while *<value>* can take any valid value that can be written to the attribute (for example, "0" or "offline"). This is a more generic way of modifying the state of a channel-path through the sysfs interface. 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, lower-case 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 using 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 using 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
```

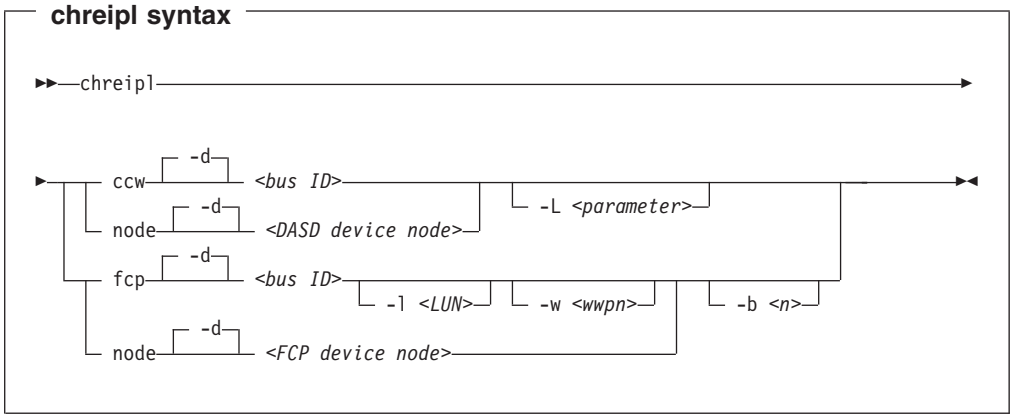
- To set channel-paths 0.12, 0.7f and 0.17 to 0.20 to the logical offline state issue:

```
# chchp -v 0 0.12,0.7f,0.17-0.20
```

chreipl - Modify the re-IPL configuration

Use the chreipl command to configure a disk or change a an entry in the boot menu for the next boot cycle.

Format



Where:

ccw selects a ccw device (DASD) for configuration.

fcp selects an FCP device (device) for configuration.

node specifies a boot target based on a device file.

-d or **--device** *<bus ID>*
specifies the device bus-ID of the re-IPL device.

<device node>
specifies the device node of the re-IPL device, either an FCP node (/dev/sd...) or a DASD node (/dev/dasd...).

-b or **--bootprog** *<n>*
specifies an optional configuration number that identifies the entry in the boot menu to use for the next reboot. The bootprog parameter only works in an FCP environment.

-L or **--loadparm** *<parameter>*
specifies an optional entry in the boot menu to use for the next reboot. This parameter must be an alphanumeric character or blank (" ") and only works if a valid zipl boot menu is present. The loadparm parameter only works in a DASD environment.

-l or **--lun** *<LUN>*
specifies the logical unit number (LUN) of the FCP re-IPL device.

-w or **--wwpn** *<wwpn>*
specifies the world-wide port name (WWPN) of the FCP re-IPL device.

-h or **--help**
displays short information on command usage.

-v or **--version**
displays version information.

Examples

This section illustrates common uses for **chreipl**.

- To boot from device /dev/dasda at next system start:

```
# chreipl node /dev/dasda
```

- To use /dev/sda as the boot device for the next boot:

```
# chreipl node /dev/sda
```

- To use the ccw device with the number 0.0.7e78 for the next system start:

```
# chreipl ccw 0.0.7e78
```

- After reboot, IPL from the ccw device with the number 0.0.7e78 using the first entry of the boot menu:

```
# chreipl ccw -d 0.0.7e78 -L 1
```

- Re-IPL from the fcp device number 0.0.1700 using WWPN 0x500507630300c562 and LUN 0x401040B300000000

```
# chreipl fcp --wwpn 0x500507630300c562 --lun 0x401040B300000000 -d 0.0.1700
```

chshut - Control the system behavior

The kernel configuration is controlled through entries below the /sys/firmware directory structure. Use the **chshut** command to change the entries pertaining to shutdown. Also see Chapter 36, “Shutdown actions,” on page 339 for more information on shutdown options.

The **chshut** command controls the system behavior in the following system states:

- Halt
- Power off
- Reboot

The **chshut** command handles up to three parameters. The first specifies the system state to which you want to change. The second argument specifies the action you want to execute in the previously specified system state. Valid arguments are ipl, reipl, stop and vmcmd.

If you have chosen vmcmd as action, a third parameter is used for the command to be executed inside z/VM.

Format

chshut syntax

```
►►─chshut──┬─ halt ─┬──┬─ ipl ──────────────────────┬──────────────────────►◄
           ├─ poff ─┤  ├─ reipl ─────────────────────┤
           └─ reboot┘  ├─ stop ───────────────────────┤
                       │   ┌──────────────────────┐  │
                       │   ▼                      │  │
                       └─── vmcmd <z/VM command>──┘──┘
```

Where:

halt specifies a system state of halt. In Red Hat Enterprise Linux 6, by default, "halt" is mapped to "poff". This can be changed by editing the file `/etc/sysconfig/shutdown` and replacing `HALT="auto"` with `HALT="halt"`.

poff specifies a system state of power off.

reboot
specifies a system state of reboot.

ipl sets the action to be taken to IPL.

reipl sets the action to be taken to re-IPL.

stop sets the action to be taken to stop.

vmcmd *<z/VM command>*
sets the action to be taken to run the specified z/VM command. The command must be in upper case. To issue several commands, repeat the vmcmd attribute with each command. The command string must be enclosed in quotation marks.

-h or **--help**
displays short information on command usage.

-v or **--version**
displays version information.

Examples

This section illustrates common uses for **chshut**.

- To make the system start again after a power off:

```
# chshut poff ipl
```

- To log off the z/VM guest if the Linux **poweroff** command was executed successfully:

```
# chshut poff vmcmd LOGOFF
```

- To send a message to guest MASTER and automatically log off the guest if the Linux power off command is executed:

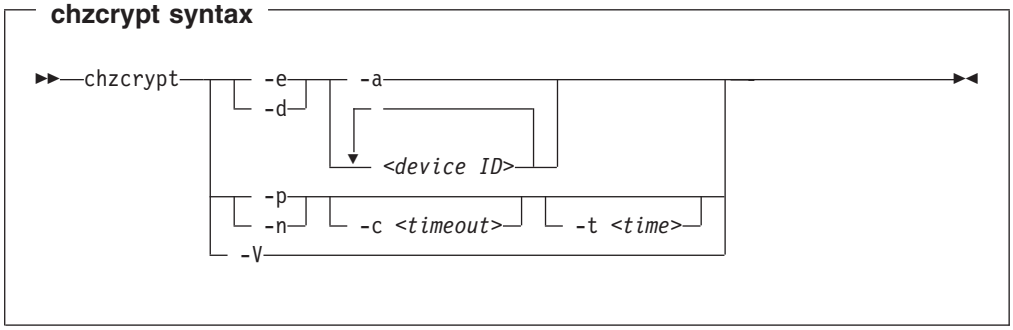
```
# chshut poff vmcmd "MSG MASTER Going down" vmcmd "LOGOFF"
```

chzcrypt - Modify the zcrypt configuration

Use the **chzcrypt** command to configure cryptographic adapters managed by zcrypt and modify zcrypt's AP bus attributes. To display the attributes, use “lszcrypt - Display zcrypt devices” on page 412.

Before you start: The sysfs file system must be mounted.

Format



Where:

-e or **--enable**
sets the given cryptographic adapters online.

-d or **--disable**
sets the given cryptographic adapters offline.

-a or **--all**
sets all available cryptographic adapters online or offline.

<device ID>
specifies a cryptographic adapter which will be set online or offline. A cryptographic adapter can be specified either in decimal notation or hexadecimal notation using a '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 re-scanning 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`.

-V or **--verbose**
displays verbose messages.

-h or **--help**
displays short information on command usage.

-v or **--version**
displays version information.

Examples

This section illustrates common uses for **chzcrypt**.

- To set the cryptographic adapters 0, 1, 4, 5, and 12 online (in decimal notation):

```
chzcrypt -e 0 1 4 5 12
```

- To set all available cryptographic adapters offline:

```
chzcrypt -d -a
```

- To set the configuration timer for re-scanning the AP bus to 60 seconds and disable zcrypt's poll thread:

```
chzcrypt -c 60 -n
```

cpuplugd - Activate CPUs and control memory

Use the **cpuplugd** command to:

- Enable or disable CPUs based on a set of rules. This increases the performance of single threaded applications within a z/VM or LPAR environment with multiple CPUs. The rules can incorporate certain system load variables.
- Manage memory when running Linux as a z/VM guest operating system.

Before you start:

- The sysfs file system must be mounted to /sys.
- The proc file system needs to be available at /proc

Format

cpuplugd syntax

```
►►─cpuplugd──┬─┬─ -c <config file>─┬─┬──────────────────────────────►◄
             │ │                   │ │
             │ └─ -f───────────────┘ │
             └─ -V───────────────────┘
```

Where:

-c or **--config** *<config file>*
: sets the path to the configuration file. The file can contain the following variables:

  - loadavg
  - idle
  - onumcpus
  - runable_proc (for hotplug)
  - apcr
  - freemem
  - swaprate (for memplug)

  To create rules you can use the operators +, *, (,), /, -, <, >, &, |, and !

  See “Examples” on page 373 below for details.

-f or **--foreground**
: runs in foreground.

-V or **--verbose**
: displays verbose messages.

-h or **--help**
: displays short information on command usage.

-v or **--version**
: displays version information.

Examples

Enabling and disabling CPUs

The following shows an example configuration file that dynamically adds or takes away CPUs according to the rules given:

```
CPU_MIN="2"
CPU_MAX="10"
UPDATE="60"

HOTPLUG = "(loadavg > onumcpus +0.75) & (idle < 10.0)"
HOTUNPLUG = "(loadavg < onumcpus -0.25) | (idle > 50)"
```

The first two lines specify the minimum and maximum numbers of CPUs. This example ensures that at least two CPUs and no more than ten CPUs are active at any time. Every 60 seconds the daemon checks if a given rule matched against the current system state. If the CPU_MAX variable equals zero, the maximum number of CPUs is equivalent to number of CPUs detected.

The hotplug line enables a CPU if the current load average (loadavg) is greater than the number of online CPUs (onumcpus) plus 0.75 and the current idle percentage (idle) is below 10 percent.

The hotunplug line disables a CPU if one of the following conditions is true:

- The load is below the number of active CPUs minus 0.25
- The idle percentage is above 50 percent.

You can also use the variable runable_proc, which represents the current number of running processes. For example:

```
HOTPLUG = "RUNABLE_PROC > (onumcpus+2)"
```

The idle percentage is extracted from /proc/stat, whereas the load average and the number of runnable processes is extracted from /proc/loadavg. Information on the current CPUs and their state can be found in the directories below /sys/devices/system/cpu (see Chapter 24, “Managing CPUs,” on page 229).

See the man page for more details.

Managing memory

You can use the **cpuplugd** command to react dynamically to changing requirements of the amount of main memory used within a Linux instance running on z/VM.

Before you begin:

- The sys file system needs to be mounted to /sys and proc needs to be available at /proc.
- You must load the cmm kernel module. For information about how to load the module, see Chapter 23, “Cooperative memory management,” on page 225.

An example configuration file might look like:

```
UPDATE="60"
CMM_MIN="0"
CMM_MAX="8192"
CMM_INC="256"

MEMPLUG = "swaprate > freemem+10 & freemem+10 < apcr"
MEMUNPLUG = "swaprate > freemem + 10000"
```

The example above illustrates the syntactic format of a memplug and menunplug rule. These two variables must be adjusted depending on the usage and workload of your Red Hat Enterprise Linux 6 instance. No general or all purpose example configuration can be provided as this does not provide a useful setup for production systems.

Every 60 seconds the daemon checks if a given rule matches against the current system state.

The cmm_min and cmm_max variables define the minimum and maximum size of the cmm static page pool respectively. For an explanation of the cmm page pools, see “Cooperative memory management background” on page 173.

The cmm_inc variable specifies the amount of pages the static page pool is increased (decreased) if a memplug (memunplug) rule is matched.

The memplug rule in the example is matched when:

1. The current swaprate (as shown in the output of the vmstat command) is greater than the current amount of free memory (in megabytes) plus 10.
2. The sum of the free memory (in megabytes) plus 10 is less than the current amount of page cache reads (apcr).

The amount of page-cache reads equals the sum of the bi and bo values shown in the output of **vmstat 1**. The swaprate equals the sum of the si and so fields of the same command. The size of the free memory is retrieved from /proc/meminfo.

For further details, see the man page.

dasdfmt - Format a DASD

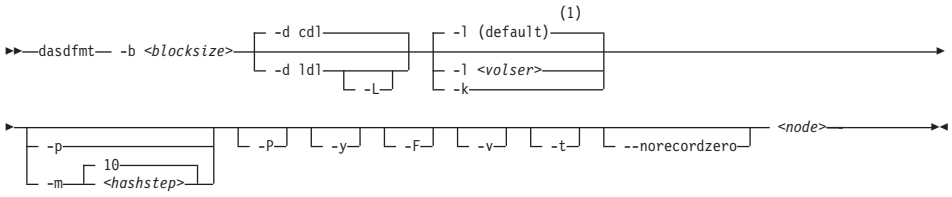
Use this tool to low-level format ECKD-type direct access storage devices (DASD).

dasdfmt uses an ioctl call to the DASD driver to format tracks. A blocksize (hard sector size) can be specified. Remember that the formatting process can take quite a long time (hours for large DASD). 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.

Format

dasdfmt syntax



Notes:

1 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.

Where:

-b *<block_size>* or **--blocksize=***<block_size>*
specifies one of the following block sizes in bytes: 512, 1024, 2048, or 4096.

If you do not specify a value for the block size, you are prompted. You can then press Enter to accept 4096 or specify a different value.

Tip: Set *<block_size>* to 1024 or higher (ideally 4096) because the ext2fs file system uses 1 KB blocks and 50% of capacity is unusable if the DASD block size is 512 bytes.

<node>
specifies the device node of the device to be formatted, for example, `/dev/dasdzzz`. See “DASD naming scheme” on page 31 for more details on 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” on page 28) to be

written to the disk. If the VOLSER contains special characters, it must be enclosed in single quotes. 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” on page 28). This is useful, for example, if the volume serial number has been written with a z/VM tool and should not be overwritten.

-p or **--progressbar**
displays a progress bar. Do not use this option if you are using a line-mode terminal console driver (for example, a 3215 terminal device driver or a line-mode hardware console device driver).

-P or **--percentage**
displays one line for each formatted cylinder showing the number of the cylinder and percentage of formatting process. Intended for use by higher level interfaces.

-m *<hashstep>* or **--hashmarks=***<hashstep>*
displays a hash mark (#) after every *<hashstep>* cylinders are formatted. *<hashstep>* must be in the range 1 to 1000. The default is 10.

The -m option is useful where the console device driver is not suitable for the progress bar (-p option).

-y starts formatting immediately without prompting for confirmation.

-F or **--force**
formats the device without checking if it is mounted.

-v displays extra information messages.

-t or **--test**
runs the command in test mode. Analyzes parameters and displays what would happen, but does not modify the disk.

-- norecordzero
prevents a format write of record zero. This is an expert option: Subsystems in DASD drivers are by default granted permission to modify or add a standard record zero to each track when needed. Before revoking 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.

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

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

--->> 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 the same disk with the compatible disk layout (using the default value of the `-d` option).

```
# dasdfmt -b 4096 -p /dev/dasdc
Drive Geometry: 100 Cylinders * 15 Heads =  1500 Tracks

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
#
```

dasdview - Display DASD structure

dasdview displays this DASD information on the system console:

- The volume label.
- VTOC details (general information, and FMT1, FMT4, FMT5, FMT7, and FMT8 labels).
- 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.

(See “The IBM label partitioning scheme” on page 26 for further information on partitioning.)

Format

dasdview syntax

```
                  ┌ -b 0 ──────┐  ┌ -s 128 ────┐  ┌ -1 ┐
►►─dasdview──┬────┼────────────┼──┼────────────┼──┼────┼──────────── <node> ──────►◄
             │    └ -b <begin> ┘  └ -s <size> ┘  └ -2 ┘
             ├ -i ──────────────────────────────────────┤
             ├ -x ──────────────────────────────────────┤
             ├ -j ──────────────────────────────────────┤
             ├ -l ──────────────────────────────────────┤
             ├ -c ──────────────────────────────────────┤
             └ -t <spec> ───────────────────────────────┘
```

Where:

-b *<begin>* or **--begin=***<begin>*

displays disk content on the console, starting from *<begin>*. The content of the disk are displayed as hexadecimal numbers, ASCII text and EBCDIC text. If *<size>* is not specified (see below), **dasdview** will take the default size (128 bytes). You can specify the variable *<begin>* as:

```
<begin>[k|m|b|t|c]
```

The default for *<begin>* is 0.

dasdview displays a disk dump on the console using the DASD driver. The DASD driver might suppress parts of the disk, or add information that is not relevant. This might occur, for example, when displaying the first two tracks of a disk that has been formatted as **cdl**. In this situation, the DASD driver will pad shorter blocks with zeros, in order to maintain a constant blocksize. All Linux applications (including **dasdview**) will 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 content of the dump are displayed as hexadecimal numbers, ASCII text, and EBCDIC text. If a start value (*begin*) is not specified, **dasdview** will take the default. You can specify the variable *<size>* as:

```
size[k|m|b|t|c]
```

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 using format 1 (as 16 Bytes per line in hexadecimal, ASCII and EBCDIC). A line number is not displayed. You can only use option **-1** together with **-b** or **-s**.

Option **-1** is the default.

-2
: displays the disk dump using format 2 (as 8 Bytes per line in hexadecimal, ASCII and EBCDIC). A decimal and hexadecimal byte count are also displayed. You can only use option **-2** 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 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).

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.
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 31 for more details on 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 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 displays:

```
encrypted disk         : no
solid state device     : no
```

- To include extended information:

```
# dasdview -x /dev/dasdzzz
```

This 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 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)
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 displays:

```
--- VTOC format 4 label -------------------------------------------------------
DS4KEYCD    : 0404040404040404040404040404040404040404040404040404040404040404...
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 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) using format 2:

```
# dasdview -b 14b -s 128 -2 /dev/dasdzzz
```

This 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 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 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 | ???????????????? | @@@@@@@@@@@@@@@@ |
+---------------------------------------+------------------+------------------+
#
```

fdasd – Partition a DASD

The compatible disk layout allows you to split DASD into several partitions. Use **fdasd** to manage partitions on a DASD. You can use **fdasd** to create, change and delete partitions, and also to change the volume serial number.

- **fdasd** checks that the volume has a valid volume label and VTOC. If either is missing or incorrect, **fdasd** recreates it.
- 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 may quit without altering the disk at any time prior to this. The items written to the disk will be the volume label, the “format 4” DSCB, a “format 5” DSCB, sometimes a “format 7” DSCB or a “format 8” DSCB depending on the DASD size, and one to three “format 1” DSCBs.

Note: To partition a SCSI disk, use **fdisk** rather than **fdasd**.

Before you start: The disk must be formatted with **dasdfmt** with the (default) `-d cdl` option.

For more information on partitions see “The IBM label partitioning scheme” on page 26.

Attention: Careless use of **fdasd** can result in loss of data.

Format

fdasd syntax

```
►►─fdasd─┬──────┬─┬──────┬─┤ partitioning options ├──────────────────►◄
         └─ -s ─┘ └─ -r ─┘
```

partitioning options:

```
├─┬─ -h ──────────────────────────────────────────────────────────┤
  ├─ -v ──────────────────────────────────────────────────────────┤
  └─┬──────────────────────────────────────────────────────────────
    │      (1)
    ├─ -a ──┬─ -k ──────────┬──────────────────────────────────────
    │       └─ -l <volser> ─┘
    ├─ -c <conf_file> ──────────────────────────────────────────────
    ├─ -i ──────────────────────────────────────────────────────────
    └─ -p ──────────────────────────────────────────────────────────
                                                          <node> ───┘
```

Notes:

1 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.

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_serial**
keeps the volume serial number when writing the volume 5 Label (see "VOLSER" on page 28). This is useful, for example, if the volume serial number has been written with a z/VM tool and should not be overwritten.

-l *<volser>* or **--label=***<volser>*
specifies the volume serial number (see "VOLSER" on page 28).

A volume serial consists of one through six alphanumeric characters or the following special characters: $, #, @, %. All other characters are ignored. Avoid using special characters in the volume serial. This may cause problems accessing a disk by VOLSER. If you must use special characters, enclose the VOLSER in single quotation marks. In addition, any '$' 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 being written to disk.

Do not use the following reserved volume serials:

- SCRTCH
- PRIVAT
- MIGRAT
- Lnnnnn (L followed by a five digit number)

These 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, controlled by the plain text configuration file *<conf_file>*.

For each partition you want to create, add one line of the following format to *<conf_file>*:

```
[x,y]
```

where x is the first track and y is the last track of that 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.

The following sample configuration file allows you to create three partitions:

```
[first,1000]
[1001,2000]
[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 31 for more details on device nodes.

Processing

fdasd menu

If you call **fdasd** in the interactive mode (that is, with just a node), the following menu appears:

```
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:

m
re-displays the **fdasd** command menu.

p Displays the following information about the DASD:

- Number of cylinders
- Number of tracks per cylinder
- Number of blocks per track
- Block size
- Volume label
- Volume identifier
- Number of partitions defined

and the following information about each partition (including the free space area):

- Linux node
- Start track
- End track
- Number of tracks
- Partition id

- Partition type (1 = file system, 2 = swap)

n adds a new partition to the DASD. You will be asked to give the start track and the length or end track of the new partition.

d deletes a partition from the DASD. You will be asked which partition to delete.

v changes the volume identifier. You will be asked to enter a new volume identifier. See “VOLSER” on page 28 for the format.

t changes the partition type. You will be asked to identify the partition to be changed. You will then be asked for the new partition type (Linux native or swap). Note that this type is a guideline; the actual use Linux makes of the partition depends on how it is defined with the `mkswap` or `mkxxfs` tools. The main function of the partition type is to describe the partition to other operating systems so that, for example, swap partitions can be skipped by backup programs.

r recreates the VTOC and thereby deletes all partitions.

u recreates all VTOC labels without removing all partitions. Existing partition sizes will be reused. This is useful to repair damaged labels or migrate partitions 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 you have made (in this session) will be discarded.

w writes your changes to disk and exits. After the data is written Linux will reread the partition table.

Examples

Example using the menu

This section gives an example of 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.

In this example, we will 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 have yet been created 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 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 using the `w` option:

```
Command (m for help): w
writing VTOC...
rereading partition table...
#
```

Example using options

You can partition using the **-a** or **-c** option without entering the menu mode. This is useful for partitioning using scripts, if you need to partition several hundred DASDs, for example.

With the **-a** parameter 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 will create a partition as follows:

```
         Device      start      end   length   Id  System
  /dev/dasdzzz1          2     1499     1498    1  Linux native
```

Using a configuration file 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 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
```

icainfo - Show available libica functions

Use this command to find out which libica functions are available on your Linux system.

Format

icainfo syntax

```
►►──icainfo──────────────────────────────────────────────────────────►◄
```

Where:

-q or **--quiet**
suppresses an explanatory introduction to the list of functions in the command output.

-v or **--version**
displays the version number of **icainfo**, then exits.

-h or **--help**
displays help information for the command.

Examples

- To show which libica functions are available on your Linux system enter:

```
# icainfo
The following CP Assist for Cryptographic Function (CPACF) operations are
supported by libica on this system:
SHA-1:    yes
SHA-256:  yes
SHA-512:  yes
DES:      yes
TDES-128: yes
TDES-192: yes
AES-128:  yes
AES-192:  yes
AES-256:  yes
PRNG:     yes
```

- To list the libica functions without the introduction enter:

```
# icainfo -q
SHA-1:    yes
SHA-256:  yes
SHA-512:  yes
DES:      yes
TDES-128: yes
TDES-192: yes
AES-128:  yes
AES-192:  yes
AES-256:  yes
PRNG:     yes
```

icastats - Show libica functions

This command is used to indicate whether libica uses hardware or works with software fallbacks. It shows also which specific functions of libica are used.

Format

icastats syntax

```
►►──icastats──── --reset ──────────────────────────────────────────►◄
```

Where:

--reset
sets the function counters to zero.

-h or **--help**
displays help information for the command.

Examples

- To display the current use of libica functions issue:

```
# icastats

function | # hardware | # software
----------+------------+------------
SHA1     |      33210 |      49815
SHA224   |     171992 |     328312
SHA256   |     189565 |     440615
SHA384   |     172081 |     323235
SHA512   |     205170 |     266679
RANDOM   |    6716896 |          0
MOD EXPO |         29 |         53
RSA CRT  |         15 |         18
DES ENC  |    2366808 |          0
DES DEC  |    2366808 |          0
3DES ENC |          0 |          0
3DES DEC |          0 |          0
AES ENC  |     576713 |     414708
AES DEC  |     576688 |     414700
```

lschp - List channel paths

Use this command to display information about channel paths.

Format

lschp syntax

```
►►─lschp──┬──────────┬──────────────────────────────────────────►◄
          ├─--help───┤
          └─--version┘
```

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

A column value of '-' indicates that a facility associated with the respective 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
===================================
.
.
0.40  1    1    1b   2   1
.
.
```

The value under Cfg. shows that the channel path is configured (1).

lscss - List subchannels

This command is used to gather subchannel information from sysfs and display it in a summary format.

Format

lscss syntax

```
►►─lscss──┬──────┬──┬──────────────────────────────────────────────┬──────────►◄
          └─ -s ─┘  │                  ┌──────────────────────────┐│
                    │                  ▼                          ,│
                    └─ -t ──<devicetype>──┬──────────────────────┬─┘
                                          └─/──<model>───────────┘
```

Where:

-s or **--short**
strips the “0.n.” from the device bus IDs in the command output.

-t or **--devtype**
limits the output to information on 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.

Examples

- This command lists all subchannels:

```
# lscss
Device   Subchan. DevType CU Type Use PIM PAM POM CHPIDs
----------------------------------------------------------------------
0.0.5C44 0.0.0000 3390/0A 3990/E9 yes C0  C0  FF  40410000 00000000
0.0.5C45 0.0.0001 3390/0A 3990/E9 yes C0  C0  FF  40410000 00000000
0.0.F5B4 0.0.0002 1732/01 1731/01 yes 80  80  FF  71000000 00000000
0.0.F5B5 0.0.0003 1732/01 1731/01 yes 80  80  FF  71000000 00000000
0.0.F5B6 0.0.0004 1732/01 1731/01 yes 80  80  FF  71000000 00000000
0.0.0191 0.0.0005 3390/0A 3990/E9     C0  C0  FF  40410000 00000000
0.0.0009 0.0.0006 0000/00 3215/00     80  80  FF  00000000 00000000
0.0.000C 0.0.0007 0000/00 2540/00     80  80  FF  00000000 00000000
0.0.000D 0.0.0008 0000/00 2540/00     80  80  FF  00000000 00000000
0.0.000E 0.0.0009 0000/00 1403/00     80  80  FF  00000000 00000000
0.0.0190 0.0.000A 3390/0A 3990/E9     C0  C0  FF  40410000 00000000
0.0.019D 0.0.000B 3390/0A 3990/E9     C0  C0  FF  40410000 00000000
0.0.019E 0.0.000C 3390/0A 3990/E9     C0  C0  FF  40410000 00000000
0.0.0592 0.0.000D 3390/0A 3990/E9     C0  C0  FF  40410000 00000000
0.0.0480 0.0.000E 3480/04 3480/01     80  80  FF  10000000 00000000
0.0.0A38 0.0.000F 3590/11 3590/50     80  80  FF  10000000 00000000
```

- This command lists subchannels with an attached 3480 model 04 or 3590 tape device and strips the “0.n.” from the device and subchannel bus-IDs in the command output:

```
# 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
```

lsdasd - List DASD devices

This command is used to gather information on DASD devices from sysfs and display it in a summary format.

Format

lsdasd syntax

```
►►─lsdasd──┬─────┬──┬────┬──┬────┬──┬────┬──┬────┬──┬────┬──►
           └─ -a─┘  └─ -b─┘  └─ -s─┘  └─ -v─┘  └─ -l─┘  └─ -c─┘  └─ -u─┘

►──┬──────────────────┬──────────────────────────────────────►◄
   │ ┌────────────────┐│
   └─┴─<device_bus_id>─┘
```

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 “0.n.” from the device bus IDs in the command output.

-v or **--verbose**
Obsolete. This option has no effect on the output.

-l or **--long**
extends the output to include UID and attributes.

-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.

--version
displays the version of the command.

<device_bus_id>
limits the output to information on 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 “0.n.” from the bus IDs in the output:

```
# lsdasd -s 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 previous format:

```
# 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
```

lsluns - Discover LUNs in Fibre Channel SANs

Use the lsluns command to discover and scan LUNs in Fibre Channel Storage Area Networks (SANs).

Format

lsluns syntax

```
                  ┌──────────────────┐
                  ▼                  │
►►─lsluns──┬──────────────────────┬──┴──────────────────────────────────────►◄
           ├─ -c── <name>─────────┤
           ├─ -p── <name>─────────┤
           └─ -a──────────────────┘
```

Where:

-c or **--ccw** *<name>*
shows LUNs for a specific adapter. The adapter name is of the form `0.0.XXXX`.

-p or **--port** *<name>*
shows LUNs for a specific port. The port name is an 8-byte hexadecimal value, for example, `0x500500ab0012cd00`.

-a or **--active**
shows the currently active LUNs. A bracketed "x" indicates that the corresponding disk is encrypted.

-v or **--version**
displays the version number of **lsluns** and exits.

-h or **--help**
displays an overview of the syntax.

Examples

- This example shows all LUNs for port 0x500507630300c562:

```
# lsluns --port 0x500507630300c562
Scanning for LUNs on adapter 0.0.5922
        at port 0x500507630300c562:
                0x4010400000000000
                0x4010400100000000
                0x4010400200000000
                0x4010400300000000
                0x4010400400000000
                0x4010400500000000
```

- This example shows all LUNs for adapter 0.0.5922:

```
# lsluns -c 0.0.5922
        at port 0x500507630300c562:
                0x4010400000000000
                0x4010400100000000
                0x4010400200000000
                0x4010400300000000
                0x4010400400000000
                0x4010400500000000
        at port 0x500507630303c562:
                0x4010400000000000
                0x4010400100000000
                0x4010400200000000
                0x4010400300000000
                0x4010400400000000
                0x4010400500000000
```

- This example shows all active LUNs:

```
# lsluns -a
adapter = 0.0.5922
     port = 0x500507630300c562
          lun = 0x401040a200000000    /dev/sg0    Disk IBM:2107900
          lun = 0x401040a300000000(x) /dev/sg1    Disk IBM:2107900
          lun = 0x401040a400000000    /dev/sg2    Disk IBM:2107900
          lun = 0x401040a500000000    /dev/sg3    Disk IBM:2107900
     port = 0x500507630303c562
         lun = 0x401040a400000000     /dev/sg4    Disk IBM:2107900
          lun = 0x401040a500000000    /dev/sg5    Disk IBM:2107900
adapter = 0.0.593a
     port = 0x500507630307c562
          lun = 0x401040b000000000    /dev/sg6    Disk IBM:2107900
          lun = 0x401040b300000000    /dev/sg7    Disk IBM:2107900
          ...
```

The `(x)` in the output indicates that the device is encrypted.

lsqeth - List qeth based network devices

This command is used to gather information on qeth-based network devices from sysfs and display it in a summary format.

Before you start: To be able to use this command you must also have installed **qethconf** (see “qethconf - Configure qeth devices” on page 431). You install **qethconf** and **lsqeth** with the same RPM.

Format

lsqeth syntax

```
►►─lsqeth──┬──────┬──┬──────────────┬──────────────────────────────────►◄
           └─ -p─┘  └─ <interface>─┘
```

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 on the specified interface only.

-h or **--help**
displays a short help text, then exits.

Examples

- The following command lists information on interface eth0 in the default format:

```
# lsqeth eth0
Device name                    : eth0
---------------------------------------------
        card_type              : OSD_100
        cdev0                  : 0.0.f5a2
        cdev1                  : 0.0.f5a3
        cdev2                  : 0.0.f5a4
        chpid                  : B5
        online                 : 1
        portname               : OSAPORT
        portno                 : 0
        route4                 : no
        route6                 : no
        checksumming           : sw checksumming
        state                  : UP (LAN ONLINE)
        priority_queueing      : always queue 2
        fake_broadcast         : 0
        buffer_count           : 16
        layer2                 : 0
        large_send             : no
        isolation              : none
        sniffer                : 0
```

- The following command lists information on all qeth-based interfaces in the former `/proc/qeth` format:

```
# lsqeth -p
devices                    CHPID interface  cardtype       port chksum prio-q'ing rtr4 rtr6 fsz   cnt
-------------------------- ----- ---------- -------------- ---- ------ ---------- ---- ---- ----- -----
0.0.833f/0.0.8340/0.0.8341 xFE   hsi0       HiperSockets   0    sw     always_q_2 no   no   n/a   16
0.0.f5a2/0.0.f5a3/0.0.f5a4 xB5   eth0       OSD_100        0    sw     always_q_2 no   no   n/a   16
0.0.fba2/0.0.fba3/0.0.fba4 xB0   eth1       OSD_100        0    sw     always_q_2 no   no   n/a   16
```

lsreipl - List IPL and re-IPL settings

Use this command to see from which device your system will boot after you issue the reboot command. Further you can query the system for information about the current boot device.

Format

lsreipl syntax

```
►►─lsreipl─┬─────┬──────────────────────────────────────────►◄
           └─ -i─┘
```

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.

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
```

lsshut - List the configuration for system states

Use this command to see how the system is configured to behave in the following system states: halt, panic, power off, and reboot.

Format

lsshut syntax

```
►►─lsshut──┬──────┬──────────────────────────────────────────────────►◄
           ├─ -h─┤
           └─ -v─┘
```

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
Panic      stop
Power off  vmcmd (LOGOFF)
Reboot     reipl
```

lstape - List tape devices

This command is used to gather information on CCW-attached tape devices and tape devices attached to the SCSI bus from sysfs (see “Displaying tape information” on page 82) and display it in a summary format.

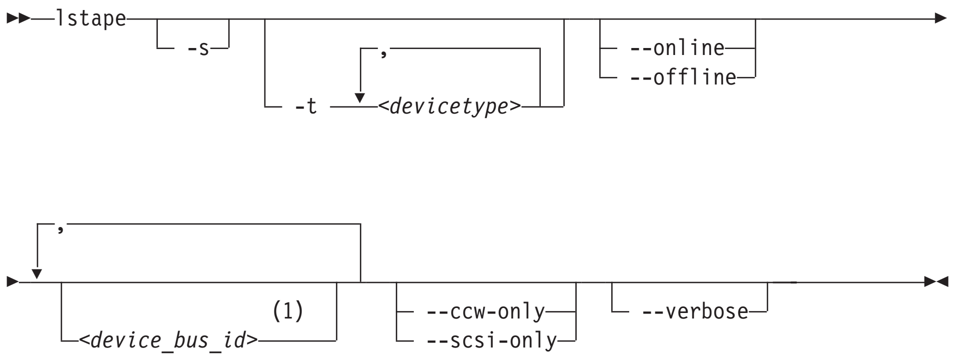
For information about SCSI tape devices, the command uses the following sources for the information displayed:

- The IBMtape or the open source lin_tape driver.
- The sg_inq command from the scsi/sg3_utils package.
- The st (SCSI tape) device driver in the Linux kernel.

If you use the IBMtape or lin_tape driver, the sg_inq utility is required. If sg_inq is missing, certain information about the IBMtape or lin_tape driver cannot be displayed.

Format

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 on the specified type or types of CCW-attached devices only.

--ccw-only
limits the output to information on CCW-attached devices only.

--scsi-only
limits the output to information on tape devices attached to the SCSI bus.

--online | --offline
limits the output to information on online or offline CCW-attached tape devices only.

<device_bus_id>
limits the output to information on the specified tape device or devices only.

--verbose
For tape devices attached to the SCSI bus only. Prints the serial of the tape as well as information about the FCP connection as an additional text line below each SCSI tape in the list.

-h or **--help**
displays a short help text.

--version
displays the version of the command.

Output attributes

The attributes in the output provide this data:

Table 49. Output for lstape

Attribute	Description
Generic	SCSI generic device file for the tape drive (for example /dev/sg0). This attribute is empty if the **sg_inq** command is not available.
Device	Main device file for accessing the tape drive, for example: • /dev/st0 for a tape drive attached through the Linux st device driver • /dev/sch0 for a medium changer device attached through the Linux changer device driver • /dev/IBMchanger0 for a medium changer attached through the IBMtape or lin_tape device driver • /dev/IBMtape0 for a tape drive 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 driver or "changer" for a medium changer.
State	The state of the SCSI device in Linux. This is an internal state of the Linux kernel, any state other than "running" can indicate problems.
HBA	The FCP adapter to which the tape drive is attached.
WWPN	The WWPN (World Wide Port Name) of the tape drive in the SAN.
Serial	The serial number field from the tape drive.

Examples

- This command displays information on all tapes found, here one CCW-attached tape and one tape and changer device 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 generic tape driver (st) and the generic changer driver (ch) are loaded, the output will list 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 on 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. Note that 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
```

lszcrypt - Display zcrypt devices

Use the **lszcrypt** command to display information about cryptographic adapters managed by zcrypt and zcrypt's AP bus attributes. To set the attributes, use “chzcrypt - Modify the zcrypt configuration” on page 370. The following information can be displayed for each cryptographic adapter:

- The card type
- The online status
- The hardware card type
- The hardware queue depth
- The request count

The following AP bus attributes can be displayed:

- The AP domain
- The configuration timer
- The poll thread status

Before you start:

- The sysfs file system must be mounted.

Format

lszcrypt syntax

```
►►─lszcrypt──┬─┬─ -V ─┬─┬──────────────────────────────────────────────┬──────────►◄
             │ │ -VV ─┘ │                                              │
             │ └─ -b ───┬──────────────────────────────────────────────┬─┘
             │          │                ┌──────────────────────────┐  │
             │          │                ▼                          │  │
             │          └─────────────── <device ID> ───────────────┘  │
             └────────────────────────────────────────────────────────┘
```

Where:

-V, -VV or **--verbose**
: increases the verbose level for cryptographic adapter information. The maximum verbose level is two (-VV). At verbose level one (-V) card type and online status are displayed. At verbose level two card type, online status, hardware card type, hardware queue depth, and request count are displayed.

<device ID>
: specifies the cryptographic adapter which will be displayed. A cryptographic adapter can be specified either in decimal notation or hexadecimal notation using a '0x' prefix. If no adapters are specified information about all available adapters will be displayed.

-b or **--bus**
: displays the AP bus attributes.

-h or **--help**
: displays short information on command usage.

-v or **--version**
: displays version information.

Examples

This section illustrates common uses for **lszcrypt**.

- To display informartion about all available cryptographic adapters:

```
# lszcrypt
```

  This displays, for example:

```
card00: CEX2A
card01: CEX2A
card02: CEX2C
card03: CEX2C
card04: CEX2C
card05: CEX2C
card06: CEX3C
card07: CEX3C
card08: CEX3C
card09: CEX3A
card0a: CEX3C
card0b: CEX3A
```

- To display card type and online status of all available cryptographic adapters:

```
lszcrypt -V
```

  This displays, for example:

```
card00: CEX2A online
card01: CEX2A online
card02: CEX2C online
card03: CEX2C online
card04: CEX2C online
card05: CEX2C online
card06: CEX3C online
card07: CEX3C online
card08: CEX3C online
card09: CEX3A online
card0a: CEX3C online
card0b: CEX3A online
```

- To display card type, online status, hardware card type, hardware queue depth, and request count for cryptographic adapters 0, 1, 10, and 12 (in decimal notation):

```
lszcrypt -VV 0 1 10 12
```

  This displays, for example:

```
card00: CEX2A online hwtype=6 depth=8 request_count=0
card01: CEX2A online hwtype=6 depth=8 request_count=0
card0a: CEX3C online hwtype=9 depth=8 request_count=0
card0c: CEX3A online hwtype=9 depth=8 request_count=0
```

- To display AP bus information:

```
lszcrypt -b
```

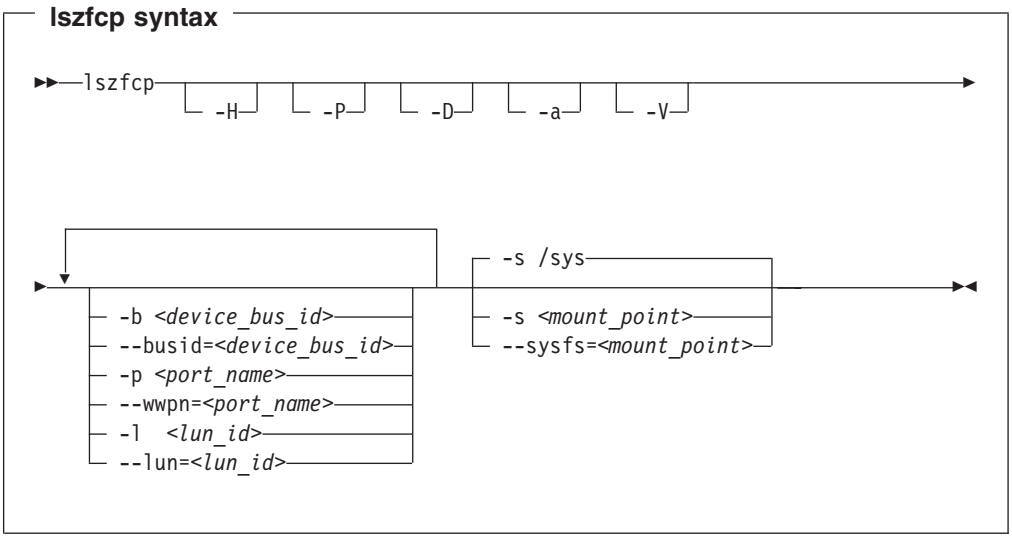
  This displays, for example:

```
ap_domain=8
ap_interrupts are enabled
config_time=30 (seconds)
poll_thread is disabled
poll_timeout=250000 (nanoseconds)
```

lszfcp - List zfcp devices

This command is used to gather information on zfcp adapters, ports, units, and their associated class devices from sysfs and to display it in a summary format.

Format



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.

-b or **--busid** *<device_bus_id>*
limits the output to information on the specified device.

-p or **--wwpn** *<port_name>*
limits the output to information on the specified port name.

-l or **--lun** *<lun_id>*
limits the output to information on the specified LUN.

-s 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 on 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 0:0:0:0
0.0.3d0c/0x500507630300c562/0x4010403300000000 0:0:0:1
0.0.3d0c/0x50050763030bc562/0x4010403200000000 0:0:1:0
0.0.3d0c/0x500507630303c562/0x4010403200000000 0:0:2:0
0.0.500c/0x50050763030bc562/0x4010403200000000 1:0:0:0
...
0.0.3c0c/0x500507630303c562/0x4010403200000000 5:0:2:0
```

- This command limits the output to the SCSI device with device bus-ID 0.0.3d0c:

```
# lszfcp -D -b 0.0.3d0c
0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:0
0.0.3d0c/0x500507630300c562/0x4010403300000000 0:0:0:1
0.0.3d0c/0x50050763030bc562/0x4010403200000000 0:0:1:0
0.0.3d0c/0x500507630303c562/0x4010403200000000 0:0:2:0
```

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 a Linux guest and 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 a service script `/etc/init.d/mon_statd` or call it manually. When it is called with the service utility, it reads the configuration file `/etc/sysconfig/mon_statd`.

Before you start:

- Install the monwriter device driver and set up z/VM to start the collection of monitor sample data. See Chapter 14, “Writing application APPLDATA records,” on page 181 for information on the setup for and usage of the monwriter device driver.
- Customize the configuration file `/etc/sysconfig/mon_statd` if you plan to call it with the service utility.

Format

You can run the **mon_fsstatd** command in two ways:

- Calling mon_statd with the service utility. This method will read the configuration file `/etc/sysconfig/mon_statd`. The mon_statd service script also controls other daemons, such as mon_procd.
- Calling mon_fsstatd from a command line.

Service utility syntax

mon_statd service utility syntax

```
►►──┬─service mon_statd──────┬──┬─start──┬──────────────────────────►◄
    └─/etc/init_d/ mon_statd─┘  ├─stop───┤
                                ├─status─┤
                                └─restart┘
```

Where:

start enables monitoring of guest file system size, using the configuration in /etc/sysconfig/mon_statd.

stop disable monitoring of guest file system size.

status show current status of guest file system size monitoring.

restart
stops and restarts monitoring. Useful to re-read the configuration file when it was changed.

Configuration file keywords:

FSSTAT_INTERVAL="<*n*>"
Specifies the desired sampling interval in seconds.

FSSTAT="yes | no"
Specifies whether to enable the mon_fsstatd daemon. Set to "yes" to enable the daemon. Anything other than "yes" will be interpreted as "no".

Command-line syntax

mon_fsstatd command-line syntax

```
                  ┌─ -i 60──────────┐
►►─mon_fsstatd────┼─────────────────┼──┬──────┬──────────────────────────►◄
                  └─ -i <seconds>───┘  └─ -a──┘
```

Where:

-i or --interval *<seconds>*
specifies the desired 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

Examples of service utility use

Example configuration file for mon_statd (/etc/sysconfig/mon_statd).

- This example sets the sampling interval to 30 seconds and enables the mon_fsstatd daemon:

```
FSSTAT_INTERVAL="30"
FSSTAT="yes"
```

Example of mon_statd use (note that your output may look different and include messages for other daemons, such as mon_procd):

- To enable guest file system size monitoring:

```
> service mon_statd start
...
Starting mon_fsstatd:[ OK ]
...
```

- To display the status:

```
> service mon_statd status
...
mon_fsstatd (pid 1075, interval: 30) is running.
...
```

- To disable guest file system size monitoring:

```
> service mon_statd stop
...
Stopping mon_fsstatd:[ OK ]
...
```

- To display the status again and check that monitoring is now disabled:

```
> service mon_statd status
...
mon_fsstatd is not running
...
```

- To restart the daemon and re-read the configuration file:

```
> service mon_statd restart
...
stopping mon_fsstatd:[ OK ]
starting mon_fsstatd:[ OK ]
...
```

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
```

Usage

Processing monitor data

The feature writes physical file system size data for a Linux guest 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 mounted at the time of the sample interval. The monitor data in each record contains a header (a time stamp, the length of the data, and an offset) followed by the file system data (as obtained from statvfs). The file system data fields begin with "fs_".

Table 50. 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 following the header.
__u16	data_offset	Offset from start of the header to start of file system data (that is, to the fields beginning with fs_).
__u16	fs_name_len	Length of the file system name. If the file system name was too long to fit in the monitor record, this is the length of the portion of the name that is contained in the monitor record.

Table 50. File system size data format (continued)

Type	Name	Description
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. If the mount directory name was too long to fit in the monitor record, this is the length of 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. If the mount type is too long to fit in the monitor record, this is the length of 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

As described in the monwriter documentation, 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 ppppppffnvvrrmm, where for records written by mon_fsstatd, these values will be:

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 will vary, 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. See Chapter 15, “Reading z/VM monitor records” for more information about monreader.

Further information

- Refer to *z/VM Saved Segments Planning and Administration*, SC24-6229 for general information on DCSSs.
- Refer to *z/VM CP Programming Services*, SC24-6179 for information on the DIAG x'DC' instruction.
- Refer to *z/VM CP Commands and Utilities Reference*, SC24-6175 for information on the CP commands.
- Refer to *z/VM Performance*, SC24-6208 for information on monitor APPLDATA.

mon_procd – Monitor Linux guest

The **mon_procd** command is a user space daemon that writes system summary information and information of each process for up to 100 concurrent processes that are managed by a Linux guest to the z/VM monitor stream using the monwriter character device driver. You can start the daemon with a service script `/etc/init.d/mon_statd` or call it manually. When it is called with the service utility, it reads the configuration file `/etc/sysconfig/mon_statd`.

Before you start:

- Install the monwriter device driver and set up z/VM to start the collection of monitor sample data. See Chapter 14, “Writing application APPLDATA records,” on page 181 for information on the setup for and usage of the monwriter device driver.
- Customize the configuration file `/etc/sysconfig/mon_statd` if you plan to call it with the service utility.
- The z/VM virtual machine in which the Linux guest running this daemon resides must have the OPTION APPLMON statement in its CP directory entry.

Format

You can run the **mon_procd** command in two ways:

- Calling **mon_procd** with the service utility. Use this method when you have the mon_statd service script installed in /etc/init.d. This method will read the configuration file /etc/sysconfig/mon_statd. The mon_statd service script also controls other daemons, such as mon_fsstatd.
- Calling **mon_procd** manually from a command line.

Service utility syntax

mon_statd service utility syntax

```
►►──┬─service mon_statd──────┬──┬─start───┬──────────────────────────────►◄
    └─/etc/init_d/ mon_statd─┘  ├─stop────┤
                                ├─status──┤
                                └─restart─┘
```

Where:

start enables monitoring of guest process data, using the configuration in /etc/sysconfig/mon_statd.

stop disables monitoring of guest process data.

status shows current status of guest process data monitoring.

restart
stops and restarts guest process data monitoring. Useful in order to re-read the configuration file when it has changed.

Configuration file keywords:

PROC_INTERVAL="<*n*>"
Specifies the desired 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" will be interpreted as "no".

Command-line syntax

mon_procd command-line syntax

```
                 ┌─ -i 60 ──────┐
►►─mon_procd─────┼──────────────┼──┬─────┬──────────────────────────────►◄
                 └─ -i <seconds>─┘  └─ -a─┘
```

Where:

-i or --interval *<seconds>*
specifies the desired 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

Examples of service utility use

Example configuration file for mon_statd (/etc/sysconfig/mon_statd).

- This example sets the sampling interval to 30 seconds and enables the mon_procd:

```
PROC_INTERVAL="30"
PROC="yes"
```

Example of mon_statd use (note that your output might look different and include messages for other daemons, such as mon_fsstatd):

- To enable guest process data monitoring:

```
> service mon_statd start
...
Starting mon_procd:[ OK ]
...
```

- To display the status:

```
> service mon_statd status
...
mon_procd (pid 1075, interval: 30) is running.
...
```

- To disable guest process data monitoring:

```
> service mon_statd stop
...
Stopping mon_procd:[ OK ]
...
```

- To display the status again and check that monitoring is now disabled:

```
> service mon_statd status
...
mon_procd is not running
...
```

- To restart the daemon and re-read the configuration file:

```
> service mon_statd restart
...
stopping mon_procd:[ OK ]
starting mon_procd:[ OK ]
...
```

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
```

Usage

Processing monitor data

The mon_procd daemon writes system summary information and information of each process for up to 100 processes currently being managed by a Linux guest 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 guest. If more than 100 processes exist in a Linux guest system at a given time, processes are sorted by the sum of CPU and memory usage percentage values and 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 427 for details. The following is the format of system summary data passed to the z/VM monitor stream.

Table 51. System summary 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.
__u64	uptime	Uptime of the Linux guest system.
__u32	users	Number of users on the Linux guest system.
char[6]	loadavg_1	Load average over the last one minute.
char[6]	loadavg_5	Load average over the last five minutes.
char[6]	loadavg_15	Load average over the last 15 minutes.
__u32	task_total	total number of tasks on the Linux guest system.
__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 representing (100 * percentage of total CPU time used for normal processes executing in user mode).
__u16	pnice	A number representing (100 * percentage of total CPU time used for niced processes executing in user mode).
__u16	psystem	A number representing (100 * percentage of total CPU time used for processes executing in kernel mode).
__u16	pidle	A number representing (100 * percentage of total CPU idle time).
__u16	piowait	A number representing (100 * percentage of total CPU time used for I/O wait).
__u16	pirq	A number representing (100 * percentage of total CPU time used for interrupts).
__u16	psoftirq	A number representing (100 * percentage of total CPU time used for softirqs).
__u16	psteal	A number representing (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 read from disk in KB.
__u64	mem_pgpgout	Data 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.

Table 51. System summary data format (continued)

Type	Name	Description
__u64	swap_cached	Cached swap memory in KB.
__u64	swap_pswpin	Pages swapped in.
__u64	swap_pswpout	Pages swapped out.

The following is the format of a process information data passed to the z/VM monitor stream.

Table 52. 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 representing (100 * percentage of the elapsed cpu time used by the process since last sampling).
__u16	pmem	A number representing (100 * percentage of physical memory used by the process).
__u64	total_time	Total cpu time the process has used.
__u64	ctotal_time	Total cpu time the process and its dead children has used.
__u64	size	Total virtual memory used by the task in KB.
__u64	swap	Swapped out portion of the virtual memory in KB.
__u64	resident	Non-swapped physical memory used by the task in KB.
__u64	trs	Physical memory devoted to executable code in KB.
__u64	drs	Physical memory devoted to other than executable code in KB.
__u64	share	Shared memory 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.

Table 52. Process data format (continued)

Type	Name	Description
__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 used to start the process and should not be larger than 64.
char[cmd_len]	cmd	Command or program name 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 used to start the process and should not be larger than 1024.
char [cmd_line_len]	cmd_line	Command line 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

As described in the monwriter documentation, 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 ppppppffnvvrrmm, where for records written by mon_procd, these values will be:

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 will vary, 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. Since a maximum number of buffers is set when a monwriter module is loaded, the maximum number of buffers must not be less than the sum of buffer records 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. See Chapter 15, “Reading z/VM monitor records” for more information about monreader.

Further information

- Refer to *z/VM Saved Segments Planning and Administration*, SC24-6229 for general information on DCSSs.
- Refer to *z/VM CP Commands and Utilities Reference*, SC24-6175 for information on the CP commands.
- Refer to *z/VM Performance*, SC24-6208 for information on monitor APPLDATA.

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. You cannot use this command in conjunction with the layer2 option. For z/VM guest LAN and VSWITCH interfaces in non-layer2 mode, note that only the `--query` option is supported.

Format

qetharp parameters

```
►►─qetharp──┬─┬─┬─┬─ -q <interface>──────────────────────────────────────────►◄
            │ │ │ │
            │ │ ├─ -n─┤
            │ │ └─ -c─┘
            │ ├─ -a <interface>── -i <ip_address>── -m <mac_address>──┤
            │ ├─ -d <interface>── -i <ip_address>──────────────────────┤
            │ └─ -p <interface>──────────────────────────────────────────┘
```

The meanings of the parameters of this command are as follows:

-q or **--query**
shows the address resolution protocol (ARP) information found in the ARP cache of the OSA or HiperSockets, which depends on *interface*. If it is an OSA device, it shows the ARP entries stored in the OSA feature's ARP cache, otherwise, the ones from the HiperSockets ARP cache. If the IP address is an IPv4 address, qetharp tries to determine the symbolic host name. If it fails, the IP address will be shown. In case of IPv6, there is currently no attempt to determine host names, so that the IP address will be shown directly.

-n or **--numeric**
shows numeric addresses instead of trying to determine symbolic host names. This option can only be used in conjunction with the -q option.

-c or **--compact**
limits the output to numeric addresses only. This option can only be used in conjunction 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 card.

<ip_address>
specifies the IP address to be added to the OSA adapter card.

-d or **--delete**
deletes a static ARP entry from the OSA adapter card.

<mac_address>
specifies the MAC address to be added to the OSA adapter card.

-p or **--purge**
flushes the ARP cache of the OSA, causing the hardware to regenerate the addresses. This option works only with OSA devices. qetharp returns immediately.

-v or **--version**
shows version information and exits

-h or **--help**
shows usage information and exits

Examples

- Show all ARP entries of the OSA defined as eth0:

```
# qetharp -q eth0
```

- Show all ARP entries of the OSA defined as eth0, without resolving host names:

```
# qetharp -nq eth0
```

- Flush the OSA's ARP cache for eth0:

```
# qetharp -p eth0
```

- Add a static entry for eth0 and IP address 1.2.3.4 to the OSA's ARP cache, using MAC address aa:bb:cc:dd:ee:ff:

```
# qetharp -a eth0 -i 1.2.3.4 -m aa:bb:cc:dd:ee:ff
```

- Delete the static entry for eth0 and IP address 1.2.3.4 from the OSA's ARP cache, using MAC address aa:bb:cc:dd:ee:ff:

```
# qetharp -d eth0 -i 1.2.3.4
```

qethconf - Configure qeth devices

The qethconf configuration tool is a bash shell script that simplifies configuring qeth devices (see Chapter 8, “qeth device driver for OSA-Express (QDIO) and HiperSockets,” on page 93) for:

- IP address takeover
- VIPA (virtual IP address)
- Proxy ARP

You cannot use this command in conjunction with the layer2 option.

From the arguments that are specified, **qethconf** assembles the corresponding function command and redirects it to the respective sysfs attributes. You can also use **qethconf** to list the already defined entries.

Format

qethconf syntax

```
►►─qethconf──┬─ ipa──┬─┬─┬─ add──┬─ <ip_addr>/<mask_bits>──┬─ <interface>──┬──────►◄
             │       │ │ │ └─ del─┘                         │               │
             │       │ │ └─ inv4──────────────────────────┘               │
             │       │ │   └─ inv6─┘                                      │
             │       │ └─ list──────────────────────────────────────────┘
             ├─ vipa─┬─┬─ add──┬─ <ip_addr>─ <interface>──┬──────────────────┤
             │ └─ parp─┘ │ └─ del─┘                        │                  │
             │           └─ list──────────────────────────┘                  │
             └─ list_all──────────────────────────────────────────────────────┘
```

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 makes the list of IP addresses that has been specified with `qethconf add` and `qethconf del` an exclusion list.

inv6
inverts the selection of address ranges for IPv6 address takeover. This makes the list of IP addresses that has been 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. Allows 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 associated with the specified address or address range.

-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 eth0
```

- Assume responsibility for packages destined for 1.2.3.5:

```
# qethconf parp add 1.2.3.5 eth0
qethconf: Added 1.2.3.5 to /sys/class/net/eth0/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 eth0
parp add 1.2.3.5 eth0
```

- Configure eth0 for IP address takeover for all addresses that start with 192.168.10:

```
# qethconf ipa add 192.168.10.0/24 eth0
qethconf: Added 192.168.10.0/24 to /sys/class/net/eth0/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 eth0
```

- Configure VIPA for eth1:

```
# qethconf vipa add 10.99.3.3 eth1
qethconf: Added 10.99.3.3 to /sys/class/net/eth1/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 eth1
```

- List all existing IPA, VIPA, and proxy ARP definitions.

```
# qethconf list_all
parp add 1.2.3.4 eth0
parp add 1.2.3.5 eth0
ipa add 192.168.10.0/24 eth0
vipa add 10.99.3.3 eth1
```

scsi_logging_level - Set and get the SCSI logging level

This command is used 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 describing 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.

Format

scsi_logging_level syntax

```
►►─scsi_logging_level─┬─┬─ -a── <level>──────────┬─┬─ -s ──────────►◄
                      │ │─ -E── <level>──────────│ │─ -g ─┬─┬─┘
                      │ │─ -T── <level>──────────│ │─ -c ─┘ │
                      │ │─ -S── <level>──────────│ │        │
                      │ │─ -M── <level>──────────│ │        │
                      │ │─ --mlqueue── <level>───│ │        │
                      │ │─ --mlcomplete── <level>│ │        │
                      │ │─ -L── <level>──────────│ │        │
                      │ │─ --llqueue── <level>───│ │        │
                      │ │─ --llcomplete── <level>│ │        │
                      │ │─ -H── <level>──────────│ │        │
                      │ │─ --hlqueue── <level>───│ │        │
                      │ │─ --hlcomplete── <level>│ │        │
                      │ └─ -I── <level>──────────┘ │        │
```

Where:

-a or **--all** *<level>*
species value for all SCSI_LOG fields.

-E or **--error** *<level>*
specifies SCSI_LOG_ERROR.

-T or **--timeout** *<level>*
specifies SCSI_LOG_TIMEOUT.

-S or **--scan** *<level>*
specifies SCSI_LOG_SCAN.

-M or **--midlevel** *<level>*
specifies SCSI_LOG_MLQUEUE and SCSI_LOG_MLCOMPLETE.

--mlqueue *<level>*
specifies SCSI_LOG_MLQUEUE.

--mlcomplete *<level>*
specifies SCSI_LOG_MLCOMPLETE.

-L 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 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 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
```

tape390_crypt - manage tape encryption

Use this command to enable and disable tape encryption for a channel attached tape device, as well as to specify key encrypting keys (KEK) by means of 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 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. The advantage of having two EEDKs is that one EEDK 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, which 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 has been encrypted, it cannot be used for IPL.

Prerequisites

To use tape encryption you need:

- A 3592 crypto-enabled tape device and control unit configured as system-managed encryption.
- A crypto-enabled 3590 channel-attached tape device driver. See Chapter 6, “Channel-attached tape device driver,” on page 77.
- A key manager. See *Encryption Key Manager Component for the Java(TM) Platform Introduction, Planning, and User's Guide*, GA76-0418 for more information.

Format

tape390_crypt syntax

```
►►─tape390_crypt─┬─ -q ──────────────┬─ <node> ──────────────────────►◄
                 │                   │
                 ├─ -e ─┬─ on ─┬─────┤
                 │      └─ off ─┘     │
                 │                   │
                 └─┤ Keys ├──────────┘
```

Keys:

```
  (1)                  ┌─<char>label─┐   ┌─ -d : ──────┐
├──────────▼─ -k <value>─┤             ├───┤             ├──┬─────┬──────────┤
                       └─<char>hash──┘   └─ -d <char>──┘  └─ -f ─┘
```

Notes:

1 The -k or --key operand can be specified maximally twice.

where:

-q or **--query**
displays information on 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 separating 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

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
```

Usage scenarios

The following scenarios illustrate the most common use of tape encryption. In all examples `/dev/ntibm0` is used as the tape device.

Using default keys for encryption:

1. Load the cartridge. If the cartridge is already loaded:
   - Switch encryption off:
     `tape390_crypt -e off /dev/ntibm`
   - 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 encryption off:
   ```
   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

This command is used to display messages on a physical tape device's display unit, optionally in conjunction with loading a tape.

Format

tape390_display syntax

```
►►─tape390_display─┬─────┬─┬─────┬──────────────────────────────────────►
                   └─ -l─┘ └─ -q─┘

   ┌─ -t standard──────┐
►──┼───────────────────┼──┬──┬──────────────────────────────────────────
   │                   │  │  │                  <message1>──────────────
   │                   │  │  └─ -b─┐
   │  -t──┬─ load──┐   │  │        │
   │      ├─ unload─┼   │  └─ <message1>── <message2>──┐
   │      └─ noop───┘   │                              │
   └─ -t── reload── <message1>── <message2>────────────┼──────────────
                                                       │
                                                       └─ <node>──────►◄
```

where:

-l or **--load**
: instructs the tape unit to load the next indexed tape from the automatic tape loader (if installed); ignored if there is no loader 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.

Notes:

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 created by udev:

- Alternately display “BACKUP” and “COMPLETE” at two 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 DASD performance

Use **tunedasd** to:

- Display and reset DASD performance statistics
- Query and set a DASD's cache mode
- Reserve and release DASD
- Breaking the lock of a known DASD (for accessing a boxed DASD while booting Linux see “Accessing DASD by force” on page 40)

Before you start: For the performance statistics, data gathering must have been switched on by writing “on” to `/proc/dasd/statistics`.

Format

tunedasd syntax

```
►►─tunedasd─┬─ -h ──────────────────────────────────────────────┬──────────►◄
            │                                                   │
            │                                   ┌──────────────┐│
            │                                   ▼              │
            │─ -g ──────────────────────────────────────────────┤─ <node> ─┘
            │─ -c <mode> ─┬──────────────────┬──────────────────│
            │             └─ -n <cylinders> ─┘                  │
            │─ -S ──────────────────────────────────────────────│
            │─ -L ──────────────────────────────────────────────│
            │─ -O ──────────────────────────────────────────────│
            │─ -R ──────────────────────────────────────────────│
            └─ -P ──┬──────────────────┬──────────────────────────┘
                    └─ -I <row> ───────┘
```

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, refer to *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.

-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 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 till 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

-v or **--version**
: displays version information.

-h or **--help**
: displays help information.

Examples

- This example first queries the current setting for the cache mode of a DASD with device node `/dev/dasdzzz` and then sets it to 1 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       0     235     150     297   18683     241       3       4       4
      0       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|
```

vmcp - Send CP commands to the z/VM hypervisor

vmcp is used to:

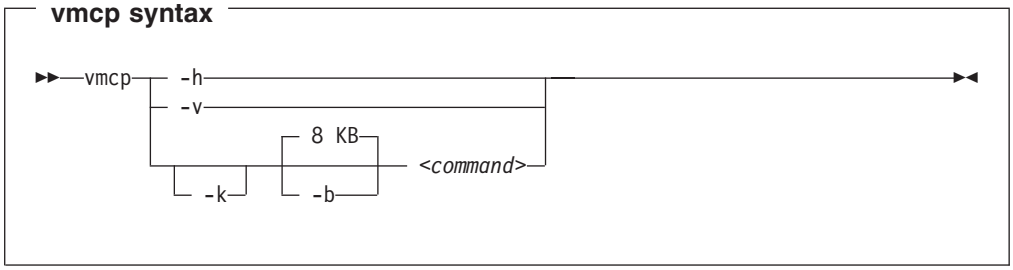
- Send control program (CP) commands to the z/VM hypervisor.
- Display the response from z/VM.

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 21, “z/VM CP interface device driver,” on page 219) or with the user space tool vmcp. In both cases, you must load the vmcp module.

Before you start: Ensure that vmcp is loaded by issuing: `modprobe vmcp`.

Format



Where:

-h or **--help**
: displays help information.

-v or **--version**
: displays version information.

-k or **--keepcase**
: converts the first word of the command to uppercase. Without this option, the complete command line is replaced by uppercase characters.

-b *<size>* or **--buffer** *<size>*
: specifies the buffer size in bytes for the response. 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.

<command>
: specifies the command you want to send to CP.

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 executed, 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 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 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

The **vmur** command provides all functions required to work with z/VM spool file queues:

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 reading the file.

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 with 0x00.

List
Display detailed information about one or all files on the specified spool file queue.

Purge Remove one or all files on the specified spool file queue.

Order Position a file at the top of the specified spool file queue.

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 in order to prevent unpredictable results or destructive conflicts.

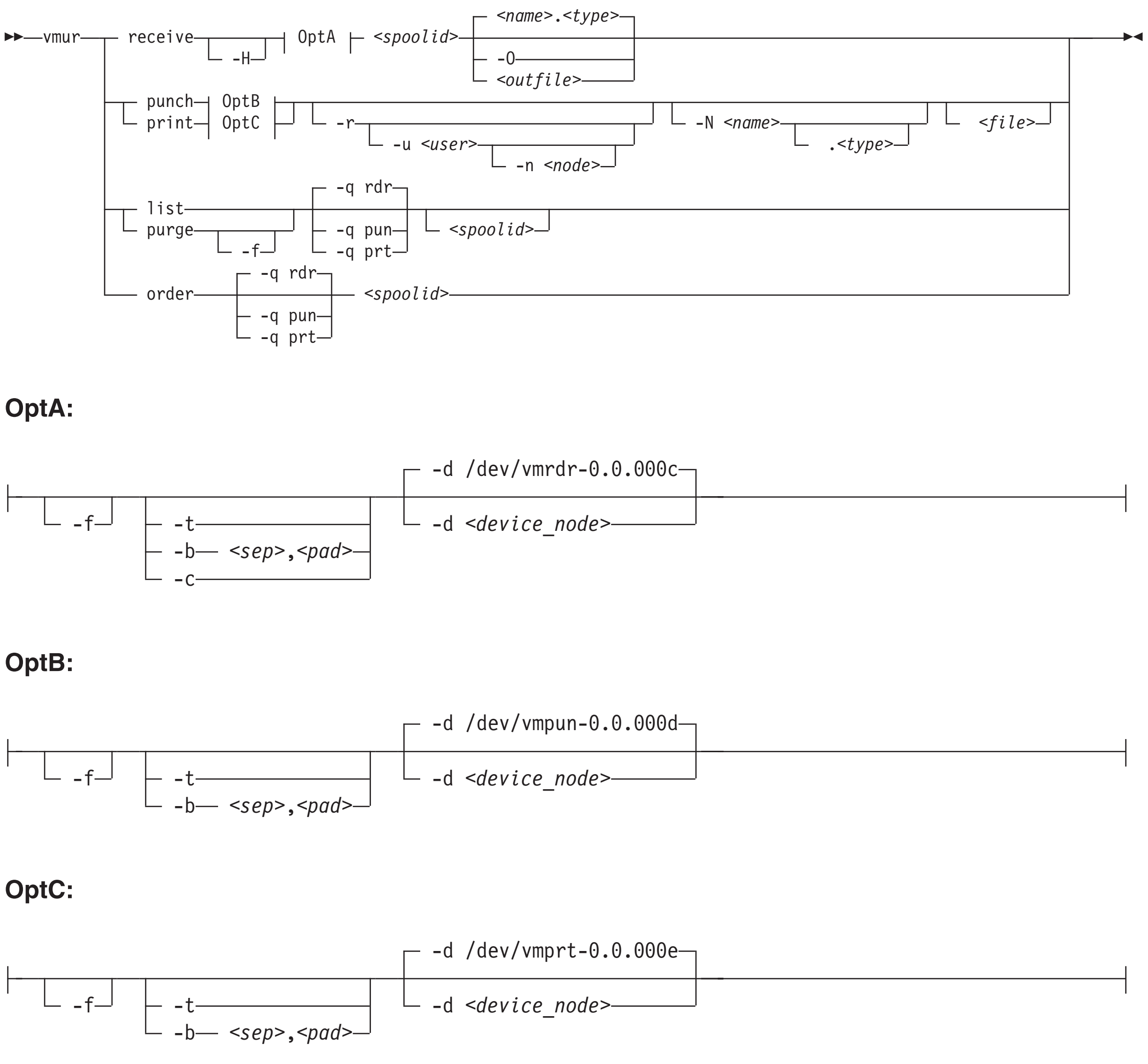
For example, this serialization prevents a process from issuing `vmur purge -f` while another process is executing `vmur receive 1234`. However, **vmur** is not serialized against concurrent CP commands issued through **vmcp**: if one process is executing `vmur receive 1234` and another process issues `vmcp purge rdr 1234`, then the received file might be incomplete. To avoid such unwanted effects use **vmur** exclusively when working with z/VM spool file queues.

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.

Before you start:
- Ensure that vmcp module is loaded by issuing: `modprobe vmcp`
- To use the receive, punch, and print functions, the vmur device driver must be loaded and the respective unit record devices must be set online.

Format

vmur syntax



Where:

re or **receive**
specifies that a file on the z/VM reader queue is to be received.

pun or **punch**
specifies that a file is to be written to the z/VM punch queue.

li or **list**
specifies that information on one or all files on a z/VM spool file queue is to be listed.

pur or **purge**
specifies that one or all files on a z/VM spool file queue is to be purged.

or or **order**
specifies that a file on a z/VM spool file queue is to be ordered, that is to be placed on top of the queue.

Note: The short forms given for receive, punch, print, list, purge, and order are the shortest forms possible. As is common in z/VM, you can use any form of these keywords that contain the minimum form. For example, `vmur re`, `vmur rec`, or `vmur rece` are all equivalent.

-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.

-q or **--queue**
specifies the z/VM spool file queue to be listed, purged or ordered. If omitted, the reader file queue is assumed.

-t or **--text**
specifies a text file requiring EBCDIC-to-ASCII conversion (or vice versa) according to character sets IBM037 and ISO-8859-1.

- For the receive function: specifies to receive the reader file as text file, that is, perform EBCDIC-to-ASCII conversion and insert an ASCII line feed character (0x0a) for each input record read from the z/VM reader. Trailing EBCDIC blanks (0x40) in the input records are stripped.
- For the punch or print function: specifies to punch the input file as text file, that is, perform ASCII-to-EBCDIC conversion and pad 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 or 132 for punch or print, respectively, an error message is issued.

The `--text` and the `--blocked` attributes are mutually exclusive.

-b *<sep, pad>*or **--blocked** *<sep, pad>*
specifies that the file has to be received or written using the blocked mode. As parameter for the `-b` option, specify the hex codes of the separator and the padding character. Example:

```
--blocked 0xSS,0xPP
```

Use this option if you need 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 read from the virtual reader and the separator character is inserted afterwards. The receive function's output can be piped to **iconv** 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 less 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 the VMDUMP spool file into a format appropriate for further analysis with crash.

-r or **--rdr**
specifies that the punch or print file is to be transferred to a reader.

-u *<user>* or **--user** *<user>*
specifies the z/VM user ID to whose reader the data is to be transferred. If user is omitted, the data is transferred to your own machine's reader. The user option is only valid if the `-r` option has been specified.

-n *<node>* or **--node** *<node>*
specifies the z/VM node ID 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 ID must be defined in the RSCS machine's configuration file. If node is omitted, the data is transferred to the specified user at your local z/VM system. The node option is only valid, if the `-u` option has been specified.

-f or **--force**
suppresses confirmation messages.

- For the receive function: specifies that *<outfile>* is to be overwritten without displaying any confirmation message.
- For the purge function: specifies that the spool files specified are to be purged without displaying any confirmation message.
- For the punch or print option: convert Linux input file name to valid spool file name automatically without any error message.

-O or **--stdout**
specifies that the reader file's contents are written to standard output.

-N or **--name**
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 to z/VM file name rules (that is, must be one to eight characters long).

If omitted, the Linux input file name (if any) is used instead. Use the --force option to enforce valid spool file names and types.

-H or **--hold**
specifies that the spool file to be received remains in the reader queue. If omitted, the spool file is purged.

<spoolid>
denotes the spool ID that identifies a file belonging to z/VM's reader, punch or printer queue. The spool ID must be a decimal number in the range 0-9999. If the spool ID is omitted in the list or purge function, all files on the respective queue are listed or purged.

<outfile>
specifies the name of the output file to receive the reader spool file's data. If both *<outfile>* and `--stdout` are omitted, name and type of the spool file to be received (see the NAME and TYPE columns in **vmur list** output) are

taken to build the output file *<name>.<type>*. If the spool file to be received is an unnamed file, an error message is issued.

<file> specifies the file data to be punched or printed. If file is omitted, the data is read from standard input.

-h or **--help**
displays short information on command usage. To view the man page, issue `man vmur`.

-v or **--version**
displays version information.

Examples

This section illustrates 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.

Besides the vmur device driver and the **vmur** command these scenarios require that:

- The vmcp module must be loaded.
- The **vmcp** and **vmconvert** commands from the s390utils package must be available.

Create and read a guest memory dump

1. Produce a 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: You can 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 using FTP:

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

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
```

Prepare the z/VM reader as an IPL device for Linux

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:

```
# chreipl ccw 0.0.000c
```

6. Boot Linux from the z/VM reader:

```
# reboot
```

Send a file to different z/VM guest virtual machines

This scenario describes how to send a file called `lnxprofile.exec` from the file system of a Linux instance that runs as a z/VM guest operating system to other 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.

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.

4. Repeat steps 2 and 3 for t6360025 at boet6360.

Send a file to a z/VSE instance

To send `lserv.job` to user ID vseuser at node vse01sys, issue:

```
vmur pun lserv.job -t -r -u vseuser -n vse01sys -N LSERV
```

znetconf - List and configure 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.

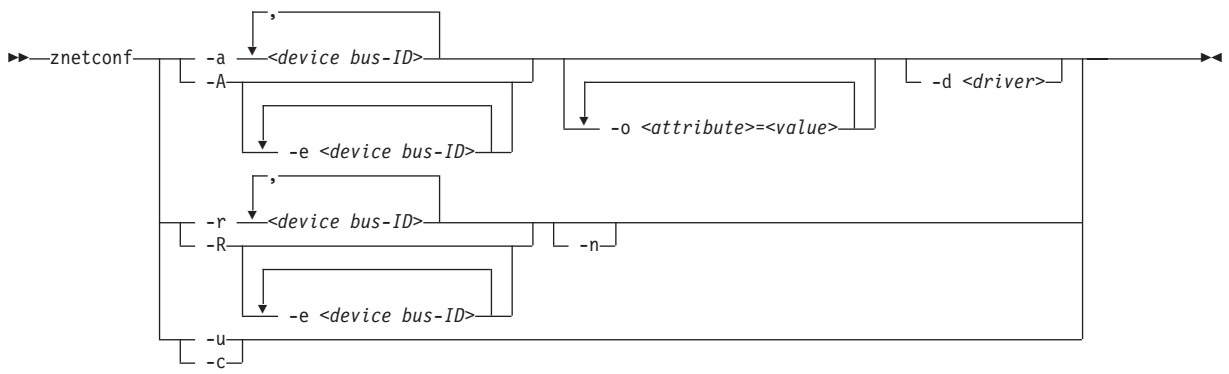
During automatic removal, znetconf sets the device offline and removes it.

Attention: Removing all network devices might lead to complete loss of network connectivity. You might require the HMC or a 3270 terminal session to restore the connectivity.

Before you start: The qeth, ctcm or lcs device drivers must be loaded. If needed, the **znetconf** command attempts to load the particular device driver.

Format

znetconf syntax



Where:

-a or **--add**
: configures the network device with the specified device bus-ID. You can enter a list of device bus-IDs 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 constituting the network device. If a device bus-ID begins with "0.0.", you can abbreviate it to the final four hexadecimal digits. For example, you can abbreviate 0.0.f503 to f503.

-A or **--add-all**
: configures all potential network devices. After running **znetconf -A**, enter

znetconf -c to see which devices have been configured. You can also enter **znetconf -u** to display devices that have not been configured.

-e or **--except**
omits the specified devices when configuring all potential network devices or removing all configured network devices.

-o or **--option** *<attribute>=<value>*
configures devices using the specified sysfs option.

-d or **--driver** *<driver name>*
configures devices using 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 separated by a comma. You can only remove configured devices as listed by **znetconf -c**.

-R or **--remove-all**
removes all configured network devices. After successfully running this command, all devices listed by **znetconf -c** become potential devices 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 and the portname option with the value myname:

```
znetconf -a f500 -o layer2=0 -o portname=myname
```

- 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 GuestLAN QDIO 00    qeth eth2 online
0.0.f503,0.0.f504,0.0.f505 1731/01 GuestLAN QDIO 01    qeth eth1 online
0.0.f5f0,0.0.f5f1,0.0.f5f2 1731/01 OSD_1000      76    qeth eth0 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
```

Chapter 42. Selected kernel parameters

The kernel parameters in this section affect Linux in general and are beyond the scope of an individual device driver or feature. Device driver-specific kernel parameters are described in the setting up section of the respective device driver chapter.

See Chapter 3, “Kernel and module parameters,” on page 17 for information about how to specify kernel parameters.

cio_ignore - List devices to be ignored

Usage

When a Linux on System z instance 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. The following applies to ignored devices:

- Ignored devices are not sensed and analyzed. The device cannot be used unless it has been 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.
- cio_ignore might hide essential devices such as the console when Linux is running as a z/VM guest operating system. The console is typically device number 0.0.0009.

See also “Changing the exclusion list” on page 461.

Format

cio_ignore syntax

```
►►─cio_ignore=─┬─all──────────────────┬──┬──────────────────────────────────────────────┬──►◄
               └─┤ <device_spec> ├────┘  │   ┌──────────────────────────────────────┐   │
                                         │   │                                      ▼   │
                                         └─,─┬─┬─────┬─┤ <device_spec> ├──────────┴───┘
                                               └─!───┘
```

<device_spec>:

```
├─┬─<device_bus_id>──────────────────────────┬─────────────────────────────────────────┤
  └─<from_device_bus_id>-<to_device_bus_id>──┘
```

Where:

all states that all devices are to be ignored.

<device_bus_id>
is a device bus ID of the form “0.n.dddd”, where n is the subchannel set ID, and dddd 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.

! 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 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 those 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 as well as 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

When a Linux on System z instance 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 has already been 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.

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.

Note: 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.

In these commands, *<device_list>* follows this syntax:

<device_list>:

```
├──┬─all──────────────┬──┬──────────────────────────────────────┬──┤
   └─┤ <device_spec> ├─┘  │   ┌─,──────────────────────────┐      │
                          └─,─┬──┬─┤ <device_spec> ├──┘
                              └─!─┘
```

<device_spec>:

```
├──┬─<device_bus_id>──────────────────────────┬──────────────────────────┤
   └─<from_device_bus_id>-<to_device_bus_id>─┘
```

Where the keywords and variables have the same meaning as in “Format” on page 460.

Note: 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 has been 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

Usage

Reduces hypervisor paging I/O overhead.

You can use Collaborative Memory Management Assist (CMMA, or "cmm2") on the z9 and later IBM processors. This support allows the CP and its guests to communicate attributes for specific 4K-byte blocks of guest memory. This exchange of information can allow both the z/VM host and its guests to optimize their use and management of memory.

Format

cmma syntax

```
                 ┌─cmma=──┬─no──┬─┐
                 │        └─off─┘ │
►►───────────────┼────────────────┼──────────────────────────────────────►◄
                 │ ┌─yes─┐        │
                 └─cmma=─┴─on──┘  ┘
```

Examples

This example switches the CMMA support on:

```
cmma=on
```

This is equivalent to:

```
cmma=yes
```

maxcpus - Restrict the number of CPUs Linux can use at IPL

Usage

Restricts the number of CPUs that Linux can use at IPL. For example, if there are four CPUs then specifying `maxcpus=2` will cause the kernel to use only two CPUs. See also “possible_cpus - Limit the number of CPUs Linux can use” on page 467.

Format

maxcpus syntax

```
►►──maxcpus=<number>──────────────────────────────────────────►◄
```

Examples

```
maxcpus=2
```

mem - Restrict memory usage

Usage

Restricts memory usage to the size specified. You can use the K, M, or G suffix to specify the value in kilobyte, megabyte, or gigabyte.

Format

mem syntax

```
►►──mem=<size>──┬─K─┬──────────────────────────────────────────►◄
                ├─M─┤
                └─G─┘
```

Examples

```
mem=64M
```

Restricts the memory Linux can use to 64 MB.

```
mem=123456K
```

Restricts the memory Linux can use to 123456 KB.

possible_cpus - Limit the number of CPUs Linux can use

Usage

Specifies the number of maximum possible and usable CPUs that Linux can add to the system. See also “maxcpus - Restrict the number of CPUs Linux can use at IPL” on page 465.

Format

possible_cpus syntax

```
►►──possible_cpus=<number>──────────────────────────────────────►◄
```

Examples

```
possible_cpus=8
```

ramdisk_size - Specify the ramdisk size

Usage

Specifies the size of the ramdisk in kilobytes.

Format

ramdisk_size syntax

```
►►──ramdisk_size=<size>──────────────────────────────────────────►◄
```

Examples

```
ramdisk_size=32000
```

ro - Mount the root file system read-only

Usage

Mounts the root file system read-only.

Format

ro syntax

```
►►──ro──────────────────────────────────────────────────────────►◄
```

root - Specify the root device

Usage

Tells Linux what to use as the root when mounting the root file system.

Format

root syntax

```
►►──root=<rootdevice>──────────────────────────────────────────────►◄
```

Examples

This example makes Linux use `/dev/dasda1` when mounting the root file system:

```
root=/dev/dasda1
```

user_mode - Set address mode for user space processes

Usage

Sets the address mode for user space processes.

Format

user_mode syntax

```
                  ┌─home────┐
►►─user_mode=─────┼─primary─┼──────────────────────────────────────►◄
                  └─secondary┘
```

Use this parameter if you are running an application that requires an address mode other than the default mode. The default address mode for user space processes is `home`.

Address mode `primary` can degrade performance on mainframe systems earlier than System z9.

Address mode `secondary` enables the data execution protection if your kernel has been built with the data execution protection feature. This has a negative performance impact on mainframe systems earlier than System z9. See Chapter 31, “Data execution protection for user processes,” on page 265 for more information about data execution protection.

Note: The `noexec` kernel parameter also sets the address mode (see “Enabling the data execution protection feature” on page 265). If you specify both `user_mode` and `noexec`, the address mode is set according to the parameter specified last.

vdso - Optimize system call performance

Usage

The kernel virtual dynamic shared object (vdso) support optimizes performance of the `gettimeofday`, `clock_getres`, and `clock_gettime` system calls. The 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 provided in the library.

The vdso support is included in the Linux on System z kernel.

Format

vdso syntax

```
                 ┌─vdso=──┬─1──┬─┐
                 │        └─on─┘ │
►►───────────────┼───────────────┼──────────────────────────────────────►◄
                 │ ┌─vdso=──┬─0──┬─┐
                 └─┼        └─off─┘ │
                   └───────────────┘
```

As 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 switch the support off.

Examples

This example switches the vdso support off:

```
vdso=0
```

vmhalt - Specify CP command to run after a system halt

Usage

Specifies a command to be issued to CP after a system halt. This command is only applicable if the Linux instance runs as a z/VM guest operating system.

Format

vmhalt syntax

```
►►──vmhalt=<COMMAND>──────────────────────────────────────────►◄
```

Examples

This example specifies that an initial program load of CMS should follow the Linux `"halt"` command:

```
vmhalt="I CMS"
```

Note: The command must be entered in uppercase.

vmpanic - Specify CP command to run after a kernel panic

Usage

Specifies a command to be issued to CP after a kernel panic. This command is only applicable if the Linux instance runs as a z/VM guest operating system.

Format

vmpanic syntax

```
►►──vmpanic=<COMMAND>──────────────────────────────────────────────►◄
```

Examples

This example specifies that a VMDUMP should follow a kernel panic:

```
vmpanic="VMDUMP"
```

Note: The command must be entered in uppercase.

vmpoff - Specify CP command to run after a power off

Usage

Specifies a command to be issued to CP after a system power off. This command is only applicable if the Linux instance runs as a z/VM guest operating system.

Format

vmpoff syntax

```
►►──vmpoff=<COMMAND>──────────────────────────────────────────►◄
```

Examples

This example specifies that CP should clear the guest 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

Usage

Specifies a command to be issued to CP on reboot. This command is only applicable if the Linux instance runs as a z/VM guest operating system.

Format

vmreboot syntax

```
►►──vmreboot=<COMMAND>──────────────────────────────────────────────►◄
```

Examples

This example specifies that a message to guest MASTER should be sent in case of a reboot:

```
vmreboot="MSG MASTER Reboot system"
```

Note: The command must be entered in uppercase.

Chapter 43. Linux diagnose code use

Red Hat Enterprise Linux 6 for System z issues several diagnose instructions to the hypervisor (LPAR or z/VM). Table 53 lists all diagnoses which are used by the Linux kernel or a kernel module.

Linux can fail if you change the privilege class of the diagnoses marked as **required** using the `MODIFY diag` command in z/VM.

Table 53. Linux diagnoses

Number	Description	Linux use	Required/ Optional
0x008	z/VM CP command console interface	• The **vmcp** command • The 3215 and 3270 console drivers • The z/VM recording device driver (vmlogrdr) • smsgiucv	Required
0x010	Release pages	CMM	Required
0x014	Input spool file manipulation	The vmur device driver	Required
0x044	Voluntary time-slice end	In the kernel for spinlock and udelay	Required
0x064	Allows Linux to attach a DCSS	The DCSS block device driver (dcssblk), xip, and the MONITOR record device driver (monreader).	Required
0x09c	Voluntary time slice yield	Spinlock.	Optional
0x0dc	Monitor stream	The APPLDATA monitor record and the MONITOR stream application support (monwriter).	Required
0x204	LPAR Hypervisor data	The hypervisor file system (hypfs).	Required
0x210	Retrieve device information	• The common I/O layer • The DASD driver DIAG access method • The vmur device driver	Required
0x224	CPU type name table	The hypervisor file system (hypfs).	Required
0x250	Block I/O	The DASD driver DIAG access method.	Required
0x258	Page-reference services	In the kernel, for pfault.	Optional
0x288	Virtual machine time bomb	The watchdog device driver.	Required
0x2fc	Hypervisor cpu and memory accounting data	The hypervisor file system (hypfs).	Required
0x308	Re-ipl	Re-ipl and dump code.	Required

Required means that a function is not available without the diagnose; optional means that the function is available but there might be a performance impact.

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 System z 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, use the Reader Comment Form in the back of this publication, send an email to eservdoc@de.ibm.com, or write to:

IBM Deutschland Research & Development GmbH
Information Development
Department 3248
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`

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 about 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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the

names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

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`

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Bibliography

The publications listed in this chapter are considered useful for a more detailed study of the topics contained in this book.

Linux on System z publications

Current versions of all Linux on System z publications can be found at:

`www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html`

- *Device Drivers, Features, and Commands on Red Hat Enterprise Linux 6*, SC34-2597
- *Using the Dump Tools on Red Hat Enterprise Linux 6*, SC34-2607
- *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413
- *How to Improve Performance with PAV*, SC33-8414
- *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596
- *libica Programmer's Reference*, SC34-2602

Red Hat Enterprise Linux 6 publications

The documentation for Red Hat Enterprise Linux 6 can be found at:

`http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/`

- *Red Hat Enterprise Linux 6 Deployment Guide*
- *Red Hat Enterprise Linux 6 Installation Guide*

z/VM publications

The publication numbers listed are for z/VM version 6. For the complete library including other versions, see:

`www.ibm.com/vm/library/`

- *z/VM Connectivity*, SC24-6174
- *z/VM CP Commands and Utilities Reference*, SC24-6175
- *z/VM CP Planning and Administration*, SC24-6178
- *z/VM CP Programming Services*, SC24-6179
- *z/VM Getting Started with Linux on System z*, SC24-6194
- *z/VM Performance*, SC24-6208
- *z/VM Saved Segments Planning and Administration*, SC24-6229
- *z/VM Systems Management Application Programming*, SC24-6234
- *z/VM TCP/IP Planning and Customization*, SC24-6238
- *z/VM Virtual Machine Operation*, SC24-6241
- *REXX/VM Reference*, SC24-6221
- *REXX/VM User's Guide*, SC24-6222

IBM Redbooks publications

You can search for, view, or download Redbooks publications, Redpapers, Hints and Tips, draft publications and additional materials, as well as order hardcopy Redbooks or CD-ROMs, at:

`www.ibm.com/redbooks`

- *Building Linux Systems under IBM VM*, REDP-0120

- *FICON CTC Implementation*, REDP-0158
- *Networking Overview for Linux on zSeries*, REDP-3901
- *Security on z/VM*, SG24-7471
- *IBM Communication Controller Migration Guide*, SG24-6298
- *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
- *Problem Determination for Linux on System z*, SG24-7599

Other System z publications

- *zSeries Application Programming Interfaces*, SB10-7030
- *IBM TotalStorage Enterprise Storage Server System/390 Command Reference 2105 Models E10, E20, F10, and F20*, SC26-7295
- *Processor Resource/Systems Manager Planning Guide*, SB10-7041
- *z/Architecture Principles of Operation*, SA22-7832

Networking publications

- *HiperSockets Implementation Guide*, SG24-6816
- *OSA-Express Customer's Guide and Reference*, SA22-7935
- *OSA-Express Implementation Guide*, SG25-5848

Security related publications

- *zSeries Crypto Guide Update*, SG24-6870
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294

ibm.com® resources

- 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 on System z information, see:
  `www.ibm.com/systems/z/connectivity/`
- 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`

Finding IBM books

For the referenced IBM books, links have been omitted to avoid pointing to a particular edition of a book. You can locate the latest versions of the referenced IBM books through the IBM Publications Center at:
`www.ibm.com/shop/publications/order`

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.

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

CCL.

The Communication Controller for Linux on System z (CCL) replaces the 3745/6 Communication Controller so that the Network Control Program (NCP) software can continue to provide business critical functions like SNI, XRF, BNN, INN, and SSCP takeover. This allows you to leverage your existing NCP functions on a "virtualized" communication controller within the Linux on System z environment.

cdl. compatible disk layout. A disk structure for Linux on System z which allows access from other System z operating systems. This replaces the older **ldl**.

CEC. (Central Electronics Complex). A synonym for *CPC*.

channel subsystem. The programmable input/output processors of the System z, 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.

Console. (1) In Linux, an output device for kernel messages. (2) In the context of IBM mainframes, a device that gives a system programmer control over the hardware resources, for example the LPARs.

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 System z devices using device numbers.

D

DASD. direct access storage device. A mass storage medium on which a computer stores data.

device driver. (1) A file that contains the code needed to use an attached device. (2) 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. (3) A collection of subroutines that control the interface between I/O device adapters and the processor.

DIAGNOSE. (1) In z/VM, a set of instructions that programs running on z/VM guest virtual machines can call to request CP services. (2) In an LPAR, a set of instructions that programs running in the LPAR can call to request hypervisor services.

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. A type of DASD emulated by z/VM.

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).

HMC. hardware management console. A console used to monitor and control hardware such as the System 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) 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). (1) The initialization procedure that causes an operating system to commence operation. (2) The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction. (3) 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.

ldl. Linux disk layout. A basic disk structure for Linux on System z. Now replaced by `cdl`.

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.linuxdoc.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 on System z. the port of Linux to the IBM System z architecture.

LPAR. logical partition of System z.

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). (1) A collection of objects that can be accessed by means of a network management protocol. (2) 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

OSA-Express. Abbreviation for Open Systems Adapter-Express networking features. These include 10 Gigabit Ethernet, Gigabit Ethernet, and Fast Ethernet.

OSM. OSA-Express for Unified Resource Manager. A CHPID type that provides connectivity to the intranode management network (INMN) from z196 to Unified Resource Manager functions. Uses OSA-Express3 1000BASE-T Ethernet exclusively operating at 1 Gbps.

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 to zBX. Uses OSA-Express3 10 Gigabit Ethernet exclusively.

P

POR. power-on reset

POSIX. Portable Operating System Interface for Computer Environments. An IEEE operating system standard closely related to the UNIX system.

R

router. A device or process which allows messages to pass between different networks.

S

SE. support element. (1) An internal control element of a processor that assists in many of the processor operational functions. (2) 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 on 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

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.

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

z196. IBM zEnterprise 196

zBX. IBM zEnterprise BladeCenter Extension

zEnterprise. IBM zEnterprise System. A heterogeneous hardware infrastructure that can consist of a zEnterprise 196 (z196) and an attached IBM zEnterprise BladeCenter Extension (zBX) Model 002, 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.

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

Readers' Comments — We'd Like to Hear from You

Linux on System z
Device Drivers, Features, and Commands
on Red Hat Enterprise Linux 6

Publication No. SC34-2597-00

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:
- Send your comments to the address on the reverse side of this form.
- Send your comments via email to: eservdoc@de.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	Email address

Readers' Comments — We'd Like to Hear from You
SC34-2597-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Research & Development GmbH
Information Development
Department 3248
Schoenaicher Strasse 220
71032 Boeblingen
Germany

Fold and Tape **Please do not staple** Fold and Tape

SC34-2597-00

Cut or Fold
Along Line

SC34-2597-00