Linux on System z

IBM

Device Drivers, Features, and Commands November, 2006

Linux Kernel 2.6 - October 2005 stream

Linux on System z

Device Drivers, Features, and Commands November, 2006

Linux Kernel 2.6 - October 2005 stream

Third Edition – (November, 2006)

This edition applies to the Linux on System z kernel 2.6 (October 2005 stream) and to all subsequent releases and modifications until otherwise indicated in new editions. This edition replaces SC33-8289-01.

SC33-8289 is the Linux on System z kernel 2.6 (October 2005 stream) equivalent to SC33-8281, which applies to Linux on zSeries kernel 2.6, April 2004 stream.

Contents

Summary of changes

This revision reflects changes to the October 2005 stream until November, 2006.

Edition 3 changes

New Information

- A new device driver described in Chapter 14, "Monitor stream application support," on page 181 allows applications to easily write application-specific monitor data to the z/VM monitor stream.
- A new file system described in Chapter 3, "S/390 hypervisor file system" supplies access to LPAR hypervisor CPU accounting data.
- A new sysfs attribute described in "Channel path measurement" on page 14 supplies access to channel path measurement data.
- Linux on System z use of diagnose codes is now described in Appendix B, "Linux diagnose code use," on page 381.
- Some common problems and their solutions have been collected in Appendix C, "Avoiding common pitfalls," on page 383.

Changed Information

- The DASD device driver now supports extended error reporting.
- The tape device driver now supports 3592 tape subsystems.
- The TSO value for the large_send attribute is ignored when layer2 is specified.
- The changed record layout of the APPLDATA_OS record is described in "APPLDATA monitor record layout" on page 176
- New features have been added to the cryptographic device driver described in Chapter 20, "Generic cryptographic device driver," on page 219.

Deleted Information

- In Chapter 22, "Control program identification," on page 235, the parameter sysplex_name has been removed.
- In Chapter 4, "DASD device driver," on page 25, the external programming interface (**ioctl**) has been removed.

This revision also includes maintenance and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Edition 2 changes

This revision reflects changes to the October 2005 stream until March 2006.

New Information

- Multiple subchannel set support. On machines that provide more than 64K device numbers, devices are grouped into subchannel sets of 64K. On these machines, the device bus ID of subchannel-attached devices is of the form 0.n.dddd, where n is the subchannel set ID and dddd is the device ID. For machines with 64K device numbers only, the subchannel set ID is always 0.
- The qeth device driver supports using a Linux guest as a guest LAN sniffer (see "Setting up a Linux guest as a z/VM guest LAN sniffer" on page 143).

- There are new cryptographic kernel configuration options (see "General architecture-specific options" on page 251).
- A new command, **lsfcp**, allows you to list information on zfcp adapters, ports, units, and their associated class devices in a summary format (see "lszfcp - List zfcp devices" on page 299).
- A new command, **san_disc**, allows you to discover ports and LUNs in SANs (see "san_disc - Discover ports and LUNs in Fibre Channel SANs" on page 308)
- A new command, **scsi_logging_level**, allows you to create, set, or get the SCSI logging level for troubleshooting purposes (see "scsi_logging_level - Set and get the SCSI logging level" on page 312).

Changed Information
- The zfcp device driver now supports Fibre Channel transport class attributes (see Chapter 5, "SCSI-over-Fibre Channel device driver," on page 49).
- The 3590 tape device driver is now available as open source (see Chapter 6, "Channel-attached tape device driver," on page 75).
- Supported IOCTLs have been added to Chapter 16, "Watchdog device driver," on page 193.
- The chccwdev command now supports generic attributes (see "chccwdev - Set a CCW device online" on page 269).
- Long command options have been added to "osasnmpd – Start OSA-Express SNMP subagent" on page 301.

This revision also includes maintenance and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Deleted Information
- The CLAW, CTC, and NETIUCV device drivers are deprecated and their descriptions have been removed. For information about the device drivers, see earlier versions of this book. The device drivers are still available for backward compatibility.

  Instead of the deprecated drivers, use the qeth device driver for OSA-Express (QDIO) and HiperSockets as follows:
  – As an alternative for CTC, use an OSA feature that is configured for QDIO mode.
  – As an alternative for virtual CTC and NETIUCV, use guest LAN Hipersockets or guest LAN type QDIO.
  – As an alternative for CTC inside a CEC use Hipersockets.

Updates for the October 2005 stream

This edition contains changes related to the October 14th, 2005 software drop. This book is the equivalent to SC33-8281, which applies to Linux® on zSeries® kernel 2.6, April 2004 stream.

Changes compared to SC33-8281-00 are as follows:

New information
- A device driver and tool is available for sending control program commands to the VM hypervisor (see "vmcp - Send CP commands to the VM hypervisor" on page 328 and Chapter 17, "z/VM CP interface device driver," on page 199

- Cooperative memory management support is available (see Chapter 18, "Cooperative memory management," on page 201)
- The IPL device can be derived (see Chapter 24, "IPL parameter interface," on page 245)

Changed Information

- The qeth device driver now supports Open Systems Adapter for NCP (ESCON/CDLC bridge) (see Chapter 8, "qeth device driver for OSA-Express (QDIO) and HiperSockets," on page 95)
- The generic cryptographic device driver now supports Crypto Express2 Accelerator (CEX2A) (see Chapter 20, "Generic cryptographic device driver")
- The DIAG250 interface supports 64-bit addressing mode from z/VM 5.2 or later (see "Enabling DIAG calls to access DASDs" on page 43)
- The SCSI-over-Fibre Channel device driver has been updated to support N-port ID virtualization on System z9 (seeChapter 5, "SCSI-over-Fibre Channel device driver," on page 49)
- Information about the impact of buffersize has been added.
- The **fdasd** command has been changed slightly to better fit other Linux commands.

Deleted Information

- A device-bus ID restriction has been deleted from Chapter 10, "CTCMPC device driver."
- Hotplug support has been removed from the generic cryptographic device driver.

About this document

This document describes the device drivers available to Linux for the control of IBM® System z9™, and eServer™ zSeries devices and attachments with the kernel 2.6 (October 2005 stream). It also provides information on commands and parameters relevant to configuring Linux for IBM mainframes.

Unless stated otherwise, the device drivers, features, and commands described in this book are available for the System z 64-bit and 31-bit architectures with version 2.6 of the Linux kernel.

Unless stated otherwise, all z/VM® related information in this book is based on the assumption that z/VM 4.4 or later is used.

In this book, System z is taken to include System z9 and zSeries in 64- and 31-bit mode.

The drivers described herein have been developed with version 2.6 of the Linux kernel. If you are using a later version of the kernel, the kernel parameters may be different from those described in this document.

For more specific information about the device driver structure, see the documents in the kernel source tree at `...linux/Documentation/s390`.

When you have installed Linux including the kernel sources, this path will be on your machine. Typically: `/usr/src/linux/Documentation/s390`.

Note: For tools related to taking and analyzing system dumps, see *Linux on System z9 and zSeries Using the Dump Tools*.

You can find the latest version of this document and of *Linux on System z9 and zSeries Using the Dump Tools* on the developerWorks® Web site at:

`ibm.com/developerworks/linux/linux390/october2005_documentation.html`

How this document is organized

The first part of this document contains general and overview information for the Linux on System z device drivers.

Part two consists of chapters specific to individual storage device drivers.

Part three consists of chapters specific to individual network device drivers.

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

Part five consists of chapters that describe Linux on System z features that are beyond the scope of an individual device driver.

Part six contains information on the commands and parameters used in configuring Linux on System z.

The Appendix provides a description of how you can boot Linux on System z.

Who should read this document

Most of the information in this document is intended for system administrators who want to configure a Linux on System z system.

Some sections are of interest primarily to kernel builders who want to build their own Linux kernel. These sections are marked with the same icon on the left margin as this paragraph.

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

Assumptions

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.

Distribution specific information

This book does not provide information that is specific to a particular Linux distribution. The device drivers, features, options, and commands it describes are either provided by the October 2005 stream downloads on developerWorks or are commonly available tools.

Your Linux distribution might provide additional utilities for working with System z devices that are not described in this book. For example, the examples in this book use the **ifconfig** command to activate interfaces. If your distribution provides it, you can also use IP tools instead of **ifconfig**. Refer to the documentation that is provided with your distribution to find out what additional utilities you can use.

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 365.

sysfs

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

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)

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
=	Equals 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:

`http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?`

Part 1. General concepts

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

- Chapter 1, "How devices are accessed by Linux"
- Chapter 2, "Devices in sysfs"
- Chapter 3, "S/390 hypervisor file system"

Chapter 1. How devices are accessed by Linux

User space programs access devices through:

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

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 30). Each device name is associated with a minor number.
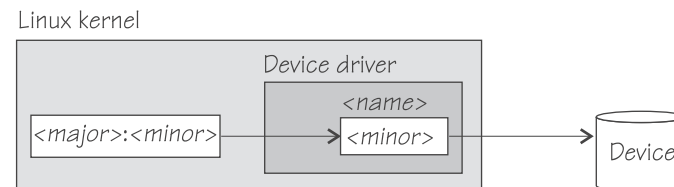
Figure 1. Major and minor numbers

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.

Your distribution might create these device nodes for you or provide udev to create them (see "Device nodes provided by udev" on page 4). If no devices nodes are provided, you need to create them yourself.

Creating device nodes

You can create a device node with an **mknod** command of the form:

```
# mknod <node> <mode> <major> <minor>
```

where:

<node>
    specifies the path to the node. You can use any path. To comply with Linux conventions, the path should begin with /dev/.

<mode>
    is "c" for character devices and "b" for block devices. For each minor number
    you can define a character device and a block device.

<major>
    is the major number that identifies the required device driver to the kernel.

<minor>
    is the minor number that maps to a device name used by the device driver.
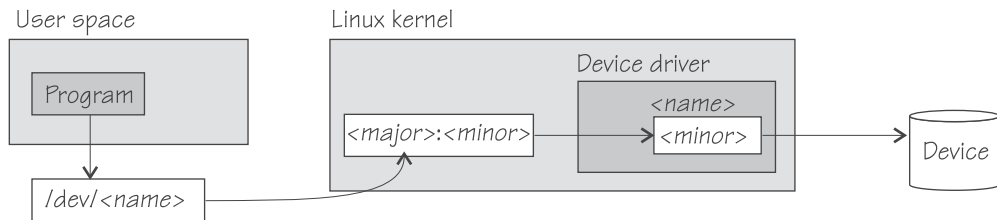
Figure 2. Device nodes

Figure 2 shows a standard device node that matches the device name used by the
device driver. You need not use device nodes like this. Which device a device node
maps to is determined by the major and minor number associated with it. You can
have multiple device nodes that all map to the same device.

For example, the following commands all create device nodes for the same device:

```
# mknod /dev/dasda b 94 0
# mknod /dev/firstdasd b 94 0
# mknod /dev/as/you/please b 94 0
```

For some device drivers, the assignment of minor numbers and names can change
between kernel boots, when devices are added or removed in a VM environment,
or even if devices are set offline and back online. The same file name, therefore,
can lead to a completely different device.

Device nodes provided by udev

If your distribution provides udev, you can use udev to create device nodes for you.
udev is a utility program that can use the device information in sysfs (see
Chapter 2, "Devices in sysfs," on page 9) to create device nodes.

Apart from creating device nodes that are based on the device names, udev can
create additional device nodes that are based on characteristics of the physical
devices, for example, on device bus-IDs or VOLSERs. Unless you change these
characteristics of your devices, the device nodes that are based on them remain the
same and map to the same device, even if the device name of a device has
changed (for example, after rebooting). udev keeps track of the mapping of the
device name and the actual devices for you and so helps you ensure that you are
addressing the device you intend to.

The format of the nodes that udev creates for you depends on distribution-specific
configuration files that reside in /etc/udev/rules.d/. If you use udev, be sure that
you use the nodes according to your distribution. Refer to your distribution
documentation to find out which udev-created device nodes are available.

See "Examples for udev-created DASD device nodes" on page 33 and "Examples for udev-created tape device nodes" on page 78 for examples of what udev created device nodes might look like.

Refer to the udev man page for more details.

Network interfaces

The Linux kernel representation of a network device is an interface.
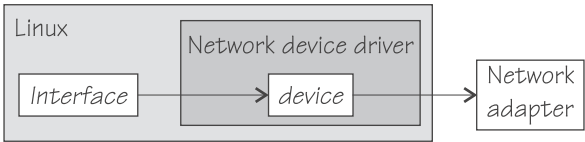
Figure 3. Interfaces

When a network device is defined, it is associated with a real or virtual network adapter. You can configure the adapter properties for a particular network device through the device representation in sysfs (see "Devices and device attributes" on page 11).

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 Linux on System z 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-2, OSA-Express2
tr	Token Ring	qeth, lcs	OSA-Express, OSA-2
ctc[1]	Channel-to-Channel	ctc	ESCON® channel card, FICON® channel card
mpc	Channel-to-Channel	mpc	ESCON channel card
claw[1]	CLAW	claw	ESCON channel card

[1]Note that the CTC and CLAW device drivers are deprecated. For information about them, see earlier versions of this book.

Table 1 summarizes the base names used for the Linux on System z 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[1]	virtual Channel-to-Channel	ctc	virtual CTC/A
mpc	virtual Channel-to-Channel	mpc	virtual CTC/A
iucv[1]	IUCV	netiucv	IUCV must be enabled for the VM guest

[1]Note that the CTC and NETIUCV device drivers are deprecated. For information about them, see earlier versions of this book.

Both the qeth device driver and the LCS device driver use the generic base name for Ethernet and Token Ring interfaces.

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. As an exception, IUCV devices do not need to be set online and the interface names are assigned when the device is created.

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. Your distribution might provide a way to track the mapping or to assign meaningful names to your interfaces.

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 295) to obtain a mapping.

After setting a device online (or creating an IUCV device), 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 (or created for IUCV).

For each IUCV network device and all other network devices that are 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.

Main steps for setting up a network interface

The following main steps apply to all Linux on System z network devices 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.

    This means creating directories that represent the device in sysfs.

- Configure the device through its attributes in sysfs (see "Device views in sysfs" on page 12).

    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 (skip this for IUCV network devices)

    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.

- 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 Linux on System z 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 and of two of the categories into which the Linux on System z device drivers and devices are grouped in sysfs.

Device categories

Figure 4 illustrates a part of the Linux on System z sysfs.
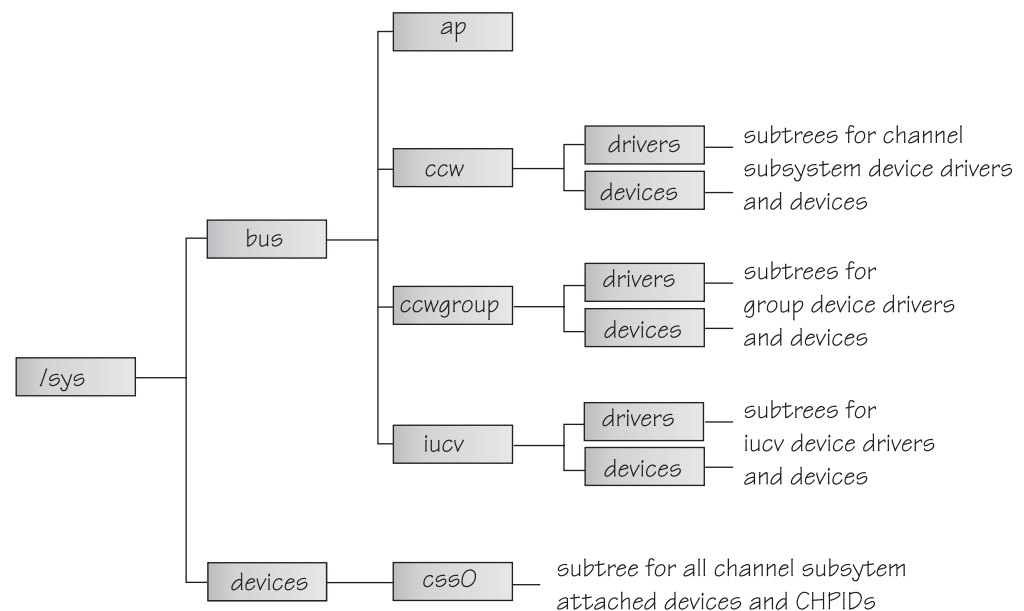
Figure 4. sysfs

`/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. Linux on System z has several categories of devices:

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 within an IBM mainframe. IUCV devices do not use the channel subsystem.

Table 3 lists the Linux on System z device drivers:

Table 3. Linux on System z device drivers

Device driver	Category	sysfs directories
3215 console	CCW	`/sys/bus/ccw/drivers/3215`
3270 console	CCW	`/sys/bus/ccw/drivers/3270`
Hardware console	n/a	n/a
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/pcica`
		`/sys/bus/ap/drivers/pcicc`
		`/sys/bus/ap/drivers/pcixcc`

DCSS	n/a	`/sys/devices/dcssblk`
z/VM monitor record reader	n/a	none
XPRAM	n/a	`/sys/devices/system/xpram`
z/VM recording device driver	IUCV	`/sys/bus/iucv/drivers/vmlogrdr`
OSA-Express, OSA-Express2 / HiperSockets (qeth)	CCW group	`/sys/bus/ccwgroup/drivers/qeth`
Watchdog device driver	n/a	none
LCS	CCW group	`/sys/bus/ccwgroup/drivers/lcs`
CTCMPC	CCW group	`/sys/bus/ccwgroup/drivers/ctcmpc`

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:

- The following are not categorized and do not have data under `/sys/bus`:
  – Hardware console device driver
  – DCSS device driver
  – z/VM monitor record reader
  – XPRAM device driver
  – Watchdog device driver
  – z/VM monitor record reader
- The IUCV device driver and the IUCV-dependent z/VM recording device driver have their own category, IUCV.

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 232 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 42).

> **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.

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. On machines that provide more than 64K device numbers, devices are grouped into subchannel sets of 64K. For machines with 64K device numbers only and for Linux systems running under z/VM, 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 10).

<device_bus_id>
       identifies an individual device (see "Devices and device attributes" on page 11).

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 11).

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>
> identifies device driver.

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

Examples:
- This example shows the path for a qeth device.
  ```
  /sys/devices/qeth/0.0.a100
  ```
- This example shows the path for a DCSS block device.
  ```
  /sys/devices/dcssblk/mydcss
  ```

Channel subsystem view

The channel subsystem view shows the devices in relation to their respective subchannel sets and subchannels. It 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 11).

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 0xb100 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.

The channel subsystem view also shows the channel-path identifiers (CHPIDs) see "CHPID information" on page 15.

Subchannel attributes

Apart from the bus ID of the attached device, the subchannel directories contain three attributes:

chpids
> is a list of the CHPIDs through with the device is connected.

detach_state
> is reserved for future use.

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 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 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
```

CHPID 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:

Note: Depending on your distribution, it might be necessary to reboot to set the device online.

```
# 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
```

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_id>/modalias` and is of the format
`ccw:t<cu_type>m<cu_model>` or
`ccw:t<cu_type>m<cu_model>dt<dev_tpe>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. S/390 hypervisor file system

The S/390 hypervisor file system provides a mechanism to access LPAR hypervisor CPU accounting data. Data is provided for each physical CPU of the machine and for all virtual CPUs of the different logical partitions.

Note that hypfs is not supported under z/VM. Therefore hypfs is not registered as a file system in a Linux VM guest.

Building a kernel with the S/390 hypervisor file system

This section is intended for those who want to build their own kernel. It describes the options you must select in the Linux configuration menu to include the S/390 hypervisor file system.

You need to select the kernel configuration option CONFIG_S390_HYPFS_FS to be able to access LPAR CPU data.

```
Base setup
   └ s390 hypervisor file system support   (CONFIG_S390_HYPFS_FS)
```

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 5 on page 20 illustrates the file system tree that is created.

```
                                    update ──── <cpu ID> ──┬── type       One subtree for
                                                           └── mgmtime   each CPU found

                                    cpus ───── <cpu ID> ──┬── type
                                                          └── mgmtime

    /sys/hypervisor/s390 ──┬─                <cpu ID> ──┬── type
                                                         └── mgmtime

                                    hyp ───── type

                           └─       systems ──┬── <LPAR name> ── cpus ──┬── <cpu ID> ──┬── type
                                              │                         │              ├── mgmtime
                                              │                         │              ├── cputime
                                              │                         │              └── onlinetime
                                              │                         │
                                              │                         └── <cpu ID>   One subtree for
                                              │                                        each CPU found
                                              │
                                              └── <LPAR name> ── cpus   Subtrees for
                                                                        each LPAR
```
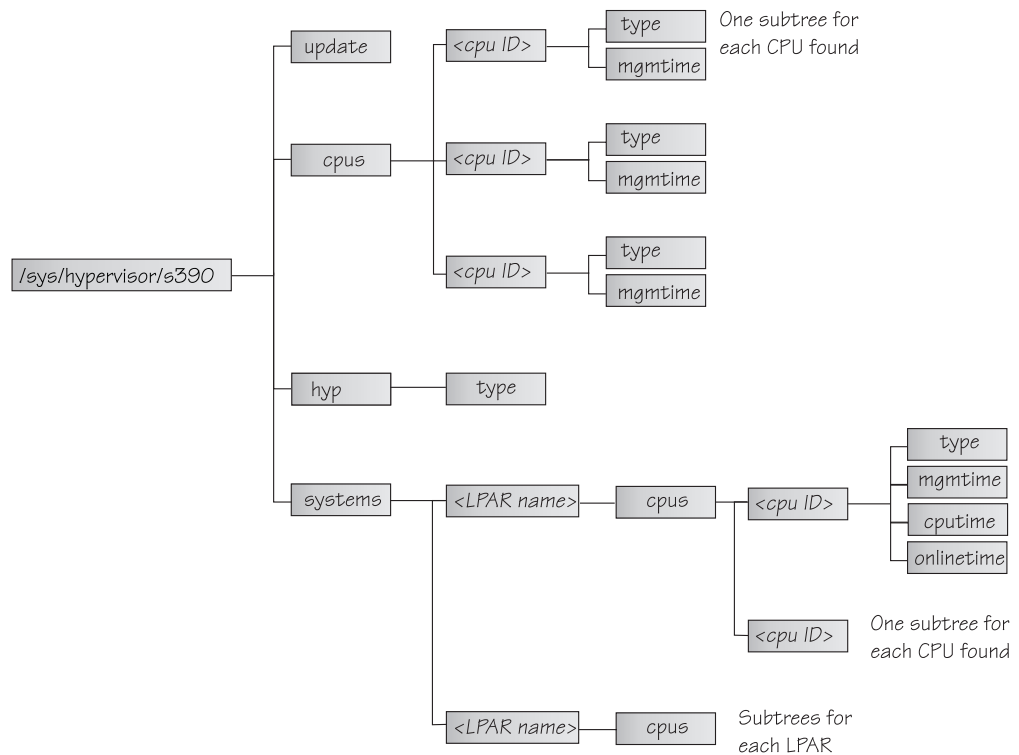
Figure 5. hypfs

The directories and attributes have the following meaning:

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 (currently only "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`.

Setting up the S/390 hypervisor file system

This section describes the parameters that you can use to configure 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
```

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 the CPU 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 (`-r-xr-x--`)

Updating the CPU 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.

Part 2. Storage device drivers

This part describes the following device drivers:

- Chapter 4, "DASD device driver"
- Chapter 5, "SCSI-over-Fibre Channel device driver"
- Chapter 6, "Channel-attached tape device driver"
- Chapter 7, "XPRAM device driver"

Note

For prerequisites and restrictions for these device drivers refer to the kernel 2.6 October 2005 stream pages on developerWorks at:

`ibm.com/developerworks/linux/linux390/october2005_recommended.html`

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 internal disks of the Multiprise® family, the volumes of
the RAMAC® virtual array, or the volumes of the IBM TotalStorage® Enterprise
Storage Server® (ESS). These emulated DASD are completely virtual and the
identity of the physical device is hidden.

SCSI disks attached through a System z9 or zSeries 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 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
FBA (Fixed Block Access)	6310	9336
	3880	3370

All models of the specified control units and device types are supported.

- The DASD device driver is also known to work with these devices:
  - Multiprise internal disks
  - RAMAC
  - RAMAC RVA
- Linux on System z 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) on storage
  devices that provide this feature.

What you should know about DASD

This section describes the available DASD layouts and the naming scheme Linux on System z uses 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®, OS/390®, z/VM, and VSE/ESA™) 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.

Linux on 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:

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:

- **Increase disk space efficiency.** You can use different block sizes for different partitions. A large block size can improve performance, but can also be wasteful of space. As a general rule, wastage amounts to half a block for each file, which can become significant for small files. It can be advantageous to store small files in a partition with a small block size and large files in a different partition with a larger block size.
- **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

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

Figure 6 illustrates a DASD with the compatible disk layout.

| IPL records | VOL1 | VTOC | /dev/dasd<x>1 | /dev/dasd<x>2 | /dev/dasd<x>3 |

/dev/dasd<x>

Figure 6. 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 30).

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 271) to format a disk with the compatible disk layout. You use the **fdasd** command (see "fdasd – Partition a DASD" on page 284) 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.

  Restriction: The VOLSER values `SCRTCH`, `PRIVAT`, `MIGRAT` or `Lnnnnn` (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

Linux on System z does not use the normal Linux partition table to keep an index of all partitions on a DASD. Like other System z operating systems, Linux on System z uses a Volume Table Of Contents (VTOC). The VTOC contains pointers to the location of every data set on the volume. In Linux on System z, these data sets form the Linux partitions.

The VTOC is located in the second track (cylinder 0, track 1). It contains a number of labels, each written in a separate block:

- One format 4 DSCB that describes the VTOC itself
- One format 5 DSCB

  The format 5 DSCB is required by other operating systems but is not used by Linux. **fdasd** sets it to zeroes.

- For volumes with more than 65636 tracks, one format 7 DSCB following the format 5 DSCB
- A format 1 DSCB for each partition

  The key of the format 1 DSCB contains the data set name, which identifies the partition to z/OS, OS/390, z/VM or VSE/ESA.

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/dasd/0.0.0193/
total 0
brw-------   1 root     root      94, 12 Jun  1  2001 disc -> ../../dasda
brw-------   1 root     root      94, 13 Jun  1  2001 part1 -> ../../dasda1
brw-------   1 root     root      94, 14 Jun  1  2001 part2 -> ../../dasda2
brw-------   1 root     root      94, 15 Jun  1  2001 part3 -> ../../dasda3
```

where the `disc` file and the `device` file represent the whole DASD and the `part#` files represent the individual partitions.

Linux disk layout

You can only format ECKD-type DASD with the Linux disk layout. Figure 7 illustrates a disk with the Linux disk layout.

Figure 7. 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 30). Linux can access the partition as /dev/dasd<x>1.

You use the **dasdfmt** command (see "dasdfmt - Format a DASD" on page 271) to format a disk with the Linux disk layout.

CMS disk layout

The CMS disk layout only applies to Linux as a 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 8 illustrates two variants of the CMS disk layout.

Figure 8. CMS disk layout

The variant in the upper part of Figure 8 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 8 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"). Linux can access the partition as `/dev/dasd<x>1`.

"Enabling DIAG calls to access DASDs" on page 43 describes how you can enable DIAG.

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 5 shows how the device names map to the available minor numbers.

Table 5. 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.

Creating device nodes

User space programs access DASD by device nodes. Your distribution might create the device nodes for you or provide udev to create them (see "Device nodes provided by udev" on page 4).

If no device nodes are created for you, you need to create them yourself, for example, with the **mknod** command. Refer to the **mknod** man page for further details.

Tip: Use the device names to construct your nodes (see "DASD naming scheme" on page 30).

Example:

The following nodes use the form /dev/*<device_name>* for the device nodes. The assignment of minor numbers is according to Table 5.

```
# mknod -m 660 /dev/dasda  b 94 0
# mknod -m 660 /dev/dasda1 b 94 1
# mknod -m 660 /dev/dasda2 b 94 2
# mknod -m 660 /dev/dasda3 b 94 3
# mknod -m 660 /dev/dasdb  b 94 4
# mknod -m 660 /dev/dasdb1 b 94 5
...
```

Assuring that a device node exists for extended error reporting

Before you start: This section applies only if you want to support applications that use the extended error reporting for ECKD-type DASD. See "Building a kernel with the DASD device driver" on page 35 for information on the kernel configuration option that enables extended error reporting.

Applications that use the extended error reporting facility require a misc character device to access the extended error data. This device node is typically called /dev/dasd_eer. If your distribution does not create the device node for you (for example, with udev), you need to create a node.

To check if there is already a node issue:

```
# find / -name dasd_eer
```

If your distribution provides the extended error reporting as a separate module, be sure to load the module before you check for the node.

If there is no device node, you need to create one. To find out the major and minor number for your monitor device read the dev attribute of the device's representation in sysfs:

```
# cat /sys/class/misc/dasd_eer/dev
```

The value of the dev attribute is of the form *<major>*:*<minor>*.

To create the device node issue a command of the form:

```
# mknod <node> c <major> <minor>
```

where *<node>* is your device node.

Example:
To create a device node /dev/dasd_eer:

```
# cat /sys/class/misc/dasd_eer/dev
10:61
# mknod /dev/dasd_eer c 10 61
```

In the example, the major number was 10 and the minor 61.

Examples for udev-created DASD device nodes

> **Note**
>
> The format of the nodes that udev creates for you depends on distribution-specific configuration files that reside in `/etc/udev/rules.d`. If you use udev, be sure that you use the nodes according to your distribution. The following examples use hypothetical nodes that are provided for illustration purposes only.

If your distribution provides udev, you can use udev to create DASD device nodes for you. udev is a utility program that can use the device information in sysfs (see Chapter 2, "Devices in sysfs," on page 9) to create device nodes.

Apart from creating device nodes that are based on the device names, udev can create additional device nodes that are based on, for example, on device bus-IDs or VOLSERs. Unless you change the VOLSERs or device numbers of your devices, device nodes that are based on a device bus-ID or VOLSER remain the same and map to the same device, even if the device name of a device has changed (for example, after rebooting). udev keeps track of the mapping of the device name and the actual devices for you and so helps you ensure that you are addressing the device you intend to.

For example, the configuration file might instruct udev to create three nodes for each device name. For a DASD with two partitions, a device bus-ID 0.0.b100 (device number 0xb100), and a VOLSER LNX001 it might create:

For the whole DASD:
- `/dev/dasdzzz` (standard device node according to the DASD naming scheme)
- `/dev/dasd/0.0.b100/disc`
- `/dev/dasd/LNX001/disc`

For the first partition:
- `/dev/dasdzzz1` (standard device node according to the DASD naming scheme)
- `/dev/dasd/0.0.b100/part1`
- `/dev/dasd/LNX001/part1`

For the second partition:
- `/dev/dasdzzz2` (standard device node according to the DASD naming scheme)
- `/dev/dasd/0.0.b100/part2`
- `/dev/dasd/LNX001/part2`

There is a program that you can use to read DASD VOLSERs from sysfs. You can use it to write your own udev rules for creating VOLSER based DASD device nodes. The following udev rule specification returns a VOLSER:

```
PROGRAM="/sbin/udev_volume_id -d -l
```

The sections that follow show how such nodes can be used to access a device by VOLSER or device bus-ID, regardless of its device name.

Accessing DASD by bus-ID

You can use device nodes that are based on your DASDs' device bus-IDs to be sure that you access a DASD with a particular bus-ID, regardless of the device name that is assigned to it.

Example

The examples in this section assume that udev provides device nodes as described in "Examples for udev-created DASD device nodes" on page 33. 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/dasd/0.0.b100/disc
```

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/dasd/0.0.b100/part1 /temp1 ext2 defaults 0 0
/dev/dasd/0.0.b100/part2 /temp2 ext2 defaults 0 0
```

Accessing DASD by VOLSER

If you want to use device nodes based on VOLSER, be sure that the VOLSERs in your environment are unique (see "Volume label" on page 27).

You can assign VOLSERs to ECKD-type devices with **dasdfmt** when formatting or later with **fdasd** when creating partitions. 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 needs to be forced, for example, by issuing:

```
# blockdev --rereadpt /dev/dasdzzz
```

Examples

The examples in this section assume that udev provides device nodes as described in "Examples for udev-created DASD device nodes" on page 33. To assure that you are addressing a device with VOLSER LNX001 you could make substitutions like the following.

Instead of issuing:

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

issue:

```
# fdasd /dev/dasd/LNX001/disc
```

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/dasd/LNX001/part1 /temp1 ext2 defaults 0 0
/dev/dasd/LNX001/part2 /temp2 ext2 defaults 0 0
```

Further information

For information on the IBM TotalStorage Enterprise Storage Server (ESS) and ECKD:

- Visit: `ibm.com/servers/storage/disk/ess/`
- Refer to *IBM TotalStorage Enterprise Storage Server User's Guide 2105 Models E10, E20, F10, and F20*, SC26-7295
- Refer to *IBM TotalStorage Enterprise Storage Server System/390® Command Reference 2105 Models E10, E20, F10, and F20*, SC26-7295

For information on DIAG refer to:

- *z/VM CP Programming Services*, SC24-5956

Building a kernel with the DASD device driver

This section is intended for those who want to build their own kernel. It describes the options you must select in the Linux configuration menu to include the DASD device driver.

The DASD device driver is provided as a base component with supplementary components for different device formats and optional functions. The driver can be compiled into the kernel or as a suite of separate modules that can be added and removed at run-time.

Figure 9 gives an overview of the available DASD kernel configuration options and the corresponding modules.

```
Block device drivers

  Support for DASD devices                        (CONFIG_DASD)
  ├─Profiling support for dasd devices            (CONFIG_DASD_PROFILE)
  ├─Support for ECKD Disks                        (CONFIG_DASD_ECKD)
  ├─Support for FBA Disks                         (CONFIG_DASD_FBA)
  ├─Support for DIAG access to Disks              (CONFIG_DASD_DIAG)
  └─Compatibility interface for DASD channel      (CONFIG_DASD_CMB)
    measurement blocks
```

Figure 9. DASD kernel configuration menu options

CONFIG_DASD

This option is required if you want to work with DASD devices and is a prerequisite for all other DASD options. It can be compiled into the kernel or as a separate module, dasd_mod.

This option depends on CONFIG_CCW.

CONFIG_DASD_PROFILE

This option makes the DASD device driver write profiling information to `/proc/dasd/statistics`.

CONFIG_DASD_ECKD

This option can be compiled into the kernel or as a separate module, dasd_eckd_mod.

CONFIG_DASD_FBA

This option can be compiled into the kernel or as a separate module, dasd_fba_mod.

CONFIG_DASD_DIAG

This option provides support for accessing disks under VM with the Diagnose250 command. It can be compiled into the kernel or as a separate module, dasd_diag_mod. It is available for 31-bit only. You must also enable the support for ECKD or FBA disks in order to get the device online.

CONFIG_DASD_EER

This option provides extended error reporting for ECKD disks. It can be compiled into the kernel or included in the separate module dasd_mod. Select this option if you want to use applications that require extended error reporting.

CONFIG_DASD_CMB

This option provides an additional interface to the channel measurement facility, which is normally accessed though sysfs. It is only needed to run applications written for the kernel 2.4 DASD channel measurement facility interface. It can be compiled into the kernel.

Setting up the DASD device driver

This section describes the parameters that you can use to configure the DASD device driver.

Kernel parameters

This section describes how to configure the DASD device driver if at least the base module has been compiled into the kernel. You configure the device driver by adding parameters to the kernel parameter line.

DASD kernel parameter syntax
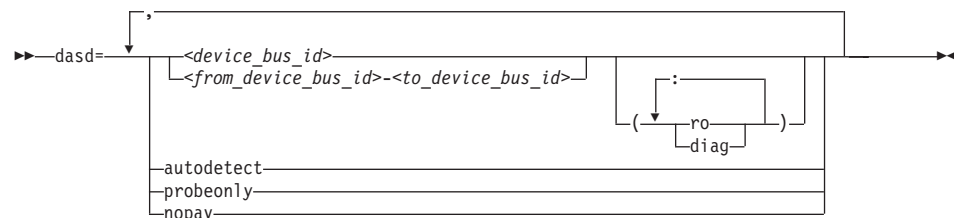
where:

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 30 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 guest operating system in VM 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) enablement for Linux instances that run in LPAR mode. The **nopav** keyword has no effect on Linux instances that run as VM guest operating systems.

<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.

If you supply a DASD kernel 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.

Example

The following kernel parameter specifies a range of DASD devices and two individual DASD devices.

```
dasd=0.0.7000-0.0.7002,0.0.7005(ro),0.0.7006
```

Table 6 shows the resulting allocation of device names and minor numbers:

Table 6. Example mapping of device names and minor numbers to devices

Minor	Name	To access
0	dasda	device 0.0.7000 as a whole
1	dasda1	the first partition on 0.0.7000
2	dasda2	the second partition on 0.0.7000
3	dasda3	the third partition on 0.0.7000

Table 6. Example mapping of device names and minor numbers to devices (continued)

Minor	Name	To access
4	dasdb	device 0.0.7001 as a whole
5	dasdb1	the first partition on 0.0.7001
6	dasdb2	the second partition on 0.0.7001
7	dasdb3	the third partition on 0.0.7001
8	dasdc	device 0.0.7002 as a whole
9	dasdc1	the first partition on 0.0.7002
10	dasdc2	the second partition on 0.0.7002
11	dasdc3	the third partition on 0.0.7002
12	dasdd	device 0.0.7005 as a whole
13	dasdd1	the first partition on 0.0.7005 (read-only)
14	dasdd2	the second partition on 0.0.7005 (read-only)
15	dasdd3	the third partition on 0.0.7005 (read-only)
16	dasde	device 0.0.7006 as a whole
17	dasde1	the first partition on 0.0.7006
18	dasde2	the second partition on 0.0.7006
19	dasde3	the third partition on 0.0.7006

Module parameters

This section describes how to load and configure those components of the DASD device driver that have been compiled as separate modules.

DASD module parameter syntax

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

device-spec:

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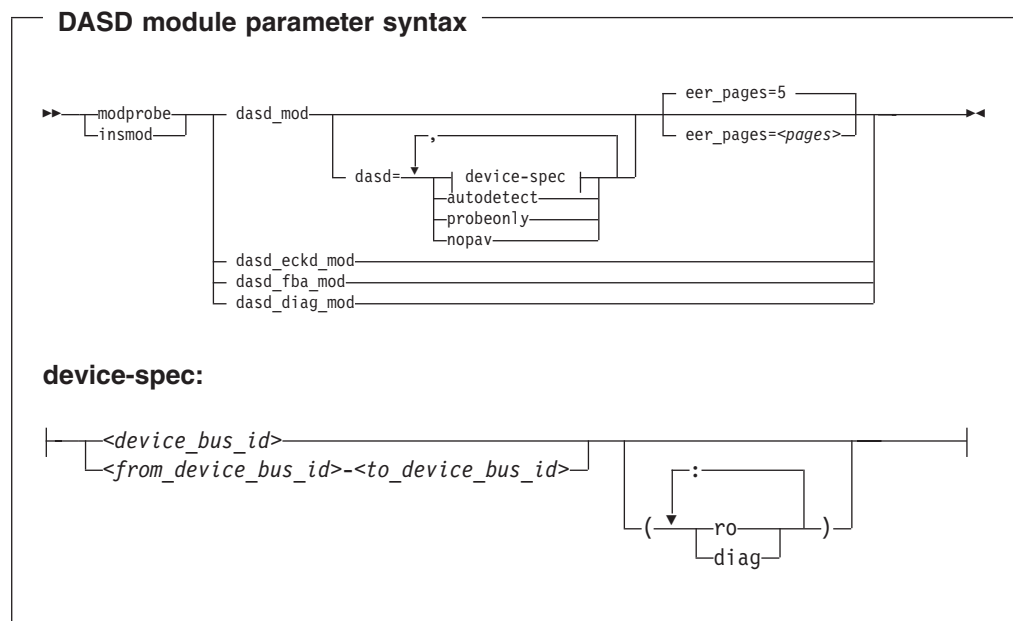
Where:

dasd_mod

loads the device driver base module.

When loading the base module you can specify the `dasd=` parameter. The variables and key words have the same meaning as in "Kernel parameters" on page 36.

When the extended error reporting feature is compiled into this module (see page 36), you can use the `eer_pages` parameter to determine the number of pages used for internal buffering of error records.

dasd_eckd_mod
loads the ECKD module.

dasd_fba_mod
loads the FBA module.

dasd_diag_mod
loads the DIAG module.

The DASD base component is required by the other modules. Be sure that it has been compiled into the kernel or that it is loaded first if it has been compiled as a separate module. **modprobe** takes care of this dependency for you and ensures that the base module is loaded automatically, if necessary.

For details on **insmod** and **modprobe** refer to the respective man pages.

Example

```
insmod dasd_mod dasd=0.0.7000-0.0.7002,0.0.7005(ro),0.0.7006
```

For the same mainframe setup, the resulting allocation of device nodes and minor numbers would be the same as in Table 6 on page 37.

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

This section describes the main steps for enabling an ECKD-type DASD for use by Linux on System z.

Before you can use an ECKD-type DASD as a Linux on System z disk, 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 base component and the ECKD component of the DASD device driver must have been compiled into the kernel or have been loaded as modules.
- 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. The DASD device nodes have the form /dev/dasd<x>, where <x> can be one to four lowercase alphabetic characters.

Perform these steps to prepare the DASD:
1. Assure that device nodes exist to address the DASD as a whole and for the partitions you intend to create.

   Example: To check if the device nodes for a DASD dasdzzz exist, change to /dev and issue:

```
# ls dasdzzz*
```

If necessary, create the device nodes. For example, issue:

```
# mknod -m 660 /dev/dasdzzz b 94 73108
# mknod -m 660 /dev/dasdzzz1 b 94 73109
# mknod -m 660 /dev/dasdzzz2 b 94 73110
# mknod -m 660 /dev/dasdzzz3 b 94 73111
```

See Table 5 on page 31 for the mapping of device names and minor numbers.

2. Format the device with the **dasdfmt** command (see "dasdfmt - Format a DASD" on page 271 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
   ```

3. 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 4. 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 284 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).

4. Depending on the intended use of each partition, create a file system on the partition or define it as a swap space.

   Either:

   > Create a file system of your choice. For example, use the Linux **mke2fs** command to create an ext3 file system (refer to the man page for details).

   > **Restriction:** You must 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:**

   ```
   # mke2fs -j -b 4096 /dev/dasdzzz1
   ```

> **Or:** Define the partition as a swap space with the **mkswap** command (refer to the man page for details).

5. Mount each file system to the mount point of your choice in Linux and enable your swap partitions.

   Example: To mount a file system in a partition `/dev/dasdzzz1` to a mount point `/mnt` and to enable a swap partition `/dev/dasdzzz2` issue:

   ```
   # mount /dev/dasdzzz1 /mnt
   # swapon /dev/dasdzzz2
   ```

   If a block device supports barrier requests, journaling file systems like ext3 or raiser-fs 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 Linux on System z.

Note: To access FBA devices, use the DIAG access method (see "Enabling DIAG calls to access DASDs" on page 43 for more information).

Before you start:
- The base component and the FBA component of the DASD device driver must have been compiled into the kernel or have been loaded as modules.
- 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. The DASD device nodes have the form `/dev/dasd<x>`, where *<x>* can be one to four lowercase alphabetic characters.

Perform these steps to prepare the DASD:

1. Assure that device nodes exist to address the DASD as a whole and the partition.

   Example: To check if the device nodes for a DASD dasdzzy exist, change to `/dev` and issue:

   ```
   # ls dasdzzy*
   ```

   If necessary, create the device nodes. For example, issue:

   ```
   # mknod -m 660 /dev/dasdzzy b 94 73104
   # mknod -m 660 /dev/dasdzzy1 b 94 73105
   ```

   See Table 5 on page 31 for the mapping of device names and minor numbers.

2. 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 **mke2fs** command to create an ext2 file system (refer to the man page for details).
>
>   **Example:** `mke2fs -b 4096 /dev/dasdzzy1`

Or: Define the partition as a swap space with the **mkswap** command (refer to the man page for details).

3. 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 12).

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 it 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 325.

Enabling DIAG calls to access DASDs

If the z/VM version is 5.2 or higher, 64-bit DIAG operations are supported. 31-bit is supported on all z/VM releases.

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 VM guest.
- The Linux instance has a 31-bit kernel or a 64-bit kernel (from z/VM 5.2 or later) that has been compiled with the CONFIG_DASD_DIAG option (see "Building a kernel with the DASD device driver" on page 35).
- The devices can be either in CMS format or in any other fixed block size format (for example LDL).
- The DIAG component (dasd_diag_mod) must be loaded or compiled into kernel.
- The component that corresponds to the DASD type (dasd_eckd_mod or dasd_fba_mod) must be loaded or compiled into kernel.
- The DASD is offline.

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 46).

Note: When switching between enabled and disabled DIAG calls on FBA-type DASD, first re-initialize the DASD. Otherwise data previously stored on the disk might no longer be accessible. You can re-initialize the DASD, for example, with CMS format or by overwriting any previous content.

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 (only applicable when compiled as module):

    ```
    # 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

Before you start: To use the extended error reporting for ECKD-type DASD you need:

*   A kernel that includes extended error reporting, either compiled into the kernel or as a separate module (see "Building a kernel with the DASD device driver" on page 35.
*   A misc character device (see "Assuring that a device node exists for extended error reporting" on page 32).

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
```

Displaying DASD information

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 7. DASD device attributes

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* on developerWorks at www.ibm.com/developerworks/linux/linux390/october2005_documentation.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.
online	"1" if the DASD is online or "0" if it is offline (see "Setting a DASD online or offline" on page 46).
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 44).
readonly	"1" if the DASD is read-only "0" if it can be written to.

uid	A device identifier of the form
	<vendor>.<serial>.<subsystem_id>.<unit_address> where
	
	<vendor>
	    is the specification from the vendor attribute.
	
	<serial>
	    is the serial number of the storage system.
	
	<subsystem_id>
	    is the subsystem ID used within the storage system to identify the DASD.
	
	<unit_address>
	    is the address used within the storage system to identify the DASD.
	This attribute is read-only.

use_diag	"1" if enable DIAG calls are enabled "0" if DIAG calls are not enabled (see "Enabling DIAG calls to access DASDs" on page 43).

Table 7. DASD device attributes (continued)

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 7 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
```

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 269) 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.

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 Linux on System z instance, for example, from VM.

When a DASD is attached, Linux attempts to initialize it according to the DASD device driver configuration (see "Kernel parameters" on page 36). 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 16).

Note

Detachment in VM of a device still open or mounted in Linux may trigger a limitation in the Linux kernel 2.6 common code and cause the system to hang or crash. Be sure that you unmount a device and set it offline before you detach it.

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 Linux on System z.

Throughout this chapter, the term *FCP channel* refers to a single virtual instance of a QDIO-based System z9 or zSeries SCSI-over-Fibre Channel adapter.

Both the Linux on System z 64-bit and 31-bit architectures are supported.

Features

The zfcp device driver supports the following devices and functions:

- Linux on System z can make use of all SAN-attached SCSI device types currently supported by Linux on other platforms. These include, for example, SCSI disks, tapes, CD-ROMs, and DVDs.
- SAN access through the following FCP adapters:
  - FICON
  - FICON Express
  - FICON Express2
- The zfcp device driver supports switched fabric and point-to-point topologies.
- The zfcp device driver provides an interface for SAN management clients (see "zfcp HBA API (FC-HBA) support" on page 51).

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.
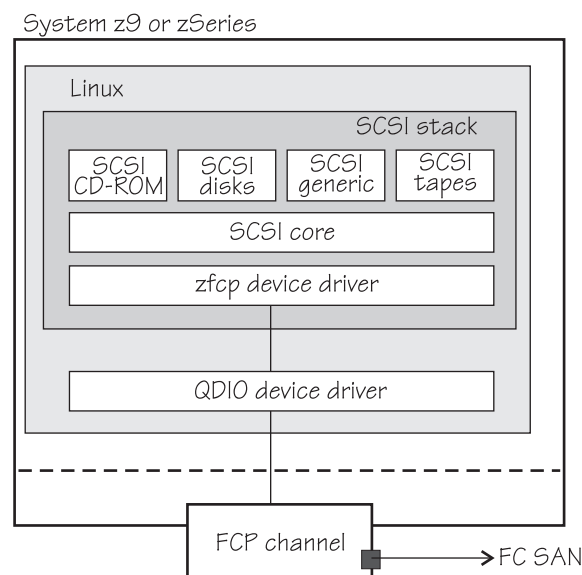
Figure 10. Device drivers supporting the Linux on System z FCP environment

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`

You can extend this structure by adding target ports to the FCP channel (see "Configuring and removing ports" on page 62). For each port you add you get 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 SCSI devices to the ports (see "Configuring SCSI devices" on page 65). For each SCSI device 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`
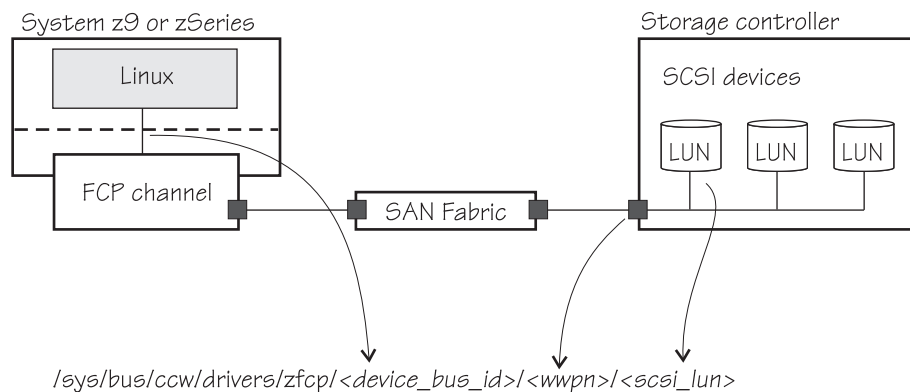
Figure 11. SCSI device in sysfs

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 299 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.

If you are using a distribution that provides udev, udev might create device nodes for you that allow you to identify the corresponding actual device. Refer to your distribution documentation to find out if udev is present and which device nodes it provides. See "Device nodes and major/minor numbers" on page 3 for more general information on udev.

You can create your own device nodes with **mknod** commands of the form:

```
# mknod /dev/<your_name> b <major> <minor>
```

See "Finding the major and minor numbers for a device" on page 70 if you need to create your own nodes.

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.

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**.

If your distribution provides udev, udev might create device nodes for your partition. Refer to your distribution documentation for details. If you need to create your own nodes for your partitions, see "Finding the major and minor numbers for a device" on page 70.

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 Linux on System z9 or zSeries.

As shown in Figure 12, the zfcp HBA API support includes a kernel module and a user space library.
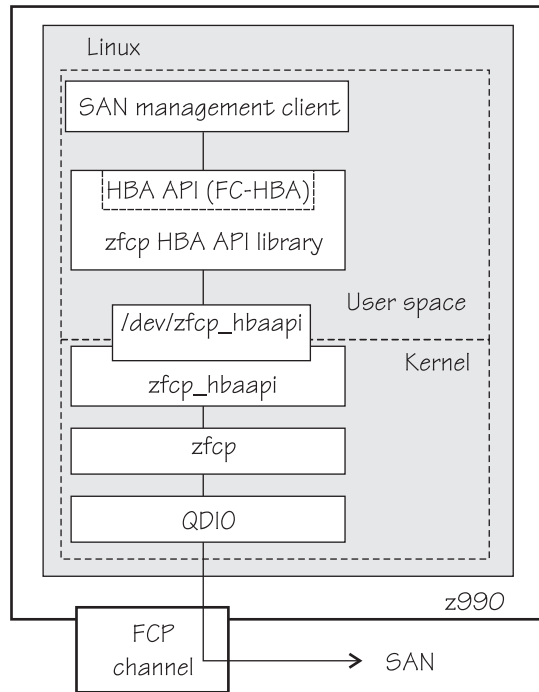
Figure 12. zfcp HBA API support modules

The library provides the zfcp HBA API to SAN management applications and uses a misc device file to communicate with the kernel module zfcp_hbaapi. The kernel module uses the zfcp device driver to communicate with the FCP adapter and the SAN.

Overview of zfcp HBA API support setup

Setting up the zfcp HBA API support includes:

1. Ensuring that the module is loaded if the support has been compiled as a module and, optionally, providing kernel or module parameters (see "zfcp HBA API kernel parameters" on page 55 and "zfcp HBA API module parameters" on page 56)

2. "Installing the zfcp HBA API library" on page 56

3. "Ensuring that the required device node exists for the HBA API support" on page 57

Discovering SAN ports and LUNs with the san_disc command

To discover ports and scan LUNs in a Fibre Channel SAN a command called **san_disc** is provided. The command can be helpful when detecting configuration problems with zfcp.

It is based on zfcp HBA API and is installed with newer versions of lib-zfcp-hbaapi. See "Installing the zfcp HBA API library" on page 56. See "san_disc - Discover ports and LUNs in Fibre Channel SANs" on page 308 for more details.

FCP LUN access control

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 Web site at:

`https://www.ibm.com/servers/resourcelink/hom03010.nsf/pages/fcpaccumain?opendocument`

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 59 for the adapter attributes.

For more details, refer to the connectivity page at

ibm.com`/servers/eserver/zseries/connectivity/fcp.html`

N_Port ID Virtualization is exclusive to IBM System z9.

Further information

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

`http://www.t10.org` and `http://www.t11.org`

Getting Started with zSeries Fibre Channel Protocol
Introduces the concepts of zSeries 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:

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

Linux for zSeries: Fibre Channel Protocol Implementation Guide
Includes an explanation of how FCP is configured using SUSE SLES9 under kernel 2.6.

`ibm.com/redbooks/abstracts/sg246344.html?Open`

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

`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.

`ibm.com/servers/eserver/zseries/connectivity/`

zfcp HBA API
See "API provided by the zfcp HBA API support" on page 72.

Building a kernel with the zfcp device driver

This section is intended for those who want to build their own kernel. It describes the options you must select in the Linux configuration menu to include the zfcp device driver.

Figure 13 summarizes the kernel configuration menu options that are relevant to the zfcp device driver:

```
SCSI Device Support --->
   SCSI low-level drivers --->
      FCP host bus adapter driver for IBM eServer zSeries  (CONFIG_ZFCP)
      └ FC HBA Support                                     (CONFIG_ZFCP_HBAAPI)
```

Figure 13. zfcp kernel configuration menu options

CONFIG_ZFCP
>   This option is required for zfcp support. Can be compiled into the kernel or as a separate module, zfcp.

CONFIG_ZFCP_HBAAPI
>   This option includes zfcp HBA API support. It can be compiled into the kernel or as a separate module, zfcp_hbaapi.

In addition, the following common code options are required:
- CONFIG_QDIO
- CONFIG_SCSI

zfcp needs the CONFIG_SCSI_FC_ATTRS option, which is automatically selected when you select CONFIG_ZFCP.

As for Linux on any platform, you need the common code options for specific devices and file systems you want to support. For example:
- SCSI disks support and PC-BIOS disk layout support

  Partitioning is only possible if PC-BIOS disk layout support is compiled into the kernel
- SCSI tapes support
- SCSI CD-ROM and ISO 9660 file system
- SCSI generic support

Setting up the zfcp device driver

This section provides information on how you can specify a SCSI boot device and how you can set up the zfcp HBA API support.

Device driver kernel parameters

This section describes how to configure the zfcp device driver if it has been compiled into the kernel. You configure the device driver by adding parameters to the kernel parameter line.

Use the zfcp.device kernel parameter to enable a SCSI device to be used as initial device.

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.

Example

The following parameter in the kernel parameter line allows you to boot from a SCSI device with LUN 0x4010403200000000, accessed through a target port with WWPN 0x500507630300c562 and connected through an FCP channel with device bus-ID 0.0.3d0c. Assuming that a device node `/dev/sda1` has been created for that SCSI device:

```
zfcp.device=0.0.3d0c,0x500507630300c562,0x4010403200000000 root=/dev/sda1
```

zfcp HBA API kernel parameters

If the HBA API support (see "zfcp HBA API (FC-HBA) support" on page 51) has been compiled into the kernel, you can optionally configure it by adding parameters to the kernel parameter line:

where:

<maxshared> is the maximum number of events in the shared event queue. The default is 20.

<maxpolled> is the maximum number of events in the polled event queue. The default is 20.

<minor number>

> is the minor number for the misc device that is registered. If no minor number is specified, it is allocated dynamically.

Example

The following parameters in the kernel parameter line limit the number of events in the zfcp HBA API shared event queue to 10 and forces the minor number for the character device to be 50.

```
zfcp_hbaapi.maxshared=10 zfcp_hbaapi.minor=50
```

zfcp HBA API module parameters

If your zfcp HBA API support has been compiled as a module, load the module with **modprobe** to assure that any other required modules are loaded. When you load the module, you can, optionally, provide parameters to configure the zfcp HBA API support.
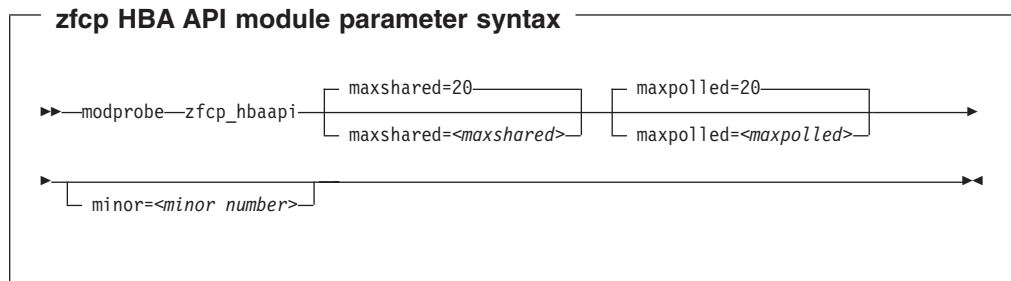
zfcp HBA API module parameter syntax

```
►►──modprobe──zfcp_hbaapi──┬─maxshared=20────────────┬──┬─maxpolled=20────────────┬──►
                           └─maxshared=<maxshared>─┘  └─maxpolled=<maxpolled>─┘

►──┬──────────────────────────────┬──────────────────────────────────────────────►◄
   └─minor=<minor number>─┘
```

where:

<maxshared>	is the maximum number of events in the shared event queue. The default is 20.
<maxpolled>	is the maximum number of events in the polled event queue. The default is 20.
<minor number>	is the minor number for the misc device that is registered. If no minor number is specified, it is allocated dynamically.

Installing the zfcp HBA API library

If you want to provide zfcp HBA API support, you need to install the zfcp HBA API library. You can find the library as a source package lib-zfcp-hbaapi-1.x.tar.gz at developerWorks:

```
ibm.com/developerworks/linux/linux390/index.shtml
```

On the left navigation bar, select Useful add-ons to find the HBA API library.

Perform the following steps to install the library:

1. Download the source package lib-zfcp-hbaapi-1.*<x>*.tar.gz from developerWorks. In the library name, *<x>* represents the newest available version.

2. Compile and install the package:

```
# tar xzf lib-zfcp-hbaapi-1.<x>.tar.gz
# cd lib-zfcp-hbaapi-1.<x>
# ./configure
# make
# make install
```

3. Optionally, build and install documentation. For this step you require the package doxygen.

```
# make dox
# make install
```

Result: You have installed:
- Shared and static versions of libzfcphbaapi at `/usr/local/lib`.
- The header file hbaapi.h at `/usr/local/include`.
- Optionally, the documentation package at `/usr/local/share/doc/zfcp-hbaapi-1.<x>`.
- A command, **san_disc**, for the discovery of ports and scanning of LUNs in a Fibre Channel SAN.

Note: The exact contents of the library depends on the version, see "API provided by the zfcp HBA API support" on page 72.

Ensuring that the required device node exists for the HBA API support

The zfcp HBA API support provides a misc device. This misc device is used for kernel-user-space communication. The major number for the device is 10. You can specify a minor number as a kernel or module parameter (see "zfcp HBA API kernel parameters" on page 55 and "zfcp HBA API module parameters" on page 56) or use a dynamic minor number.

Check for the device node, for example at /dev/zfcp_hbaapi. If your distribution does not create a device node for the zfcp HBA API support (for example, with udev), you need to create one.

To find out the major and minor number for your monitor device, read the dev attribute of the device's representation in sysfs:

```
# cat /sys/class/misc/zfcp_hbaapi/dev
```

The value of the dev attribute is of the form 10:*<minor>*.

To create, issue a command of the form:

```
# mknod <node> c 10:<minor>
```

where *<node>* is your device node.

Example:
To create a device node `/dev/zfcp_hbaapi`:

```
# cat /sys/class/misc/zfcp_hbaapi/dev
10:64
# mknod /dev/zfcp_hbaapi c 10 64
```

In the example, the major number was 10 and the minor 64.

Alternatively, you can create the device node, `/dev/zfcp_hbaapi`, using the following commands:

```
# minor='cat /proc/misc | awk "\$2==\"zfcp_hbaapi\" {print \$1}"'
# mknod /dev/zfcp_hbaapi c 10 $minor
```

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
  - Recovering a failed FCP channel
  - Configuring and removing ports
  - Finding out if NPIV is in use
- Working with target ports
  - Displaying port information
  - Recovering a failed port
  - Configuring SCSI devices
- Working with SCSI devices
  - Mapping the representations of a SCSI device in sysfs
  - Displaying information on SCSI devices
  - Finding the major and minor numbers for a device
  - Recovering a failed SCSI device
  - Removing SCSI devices

Displaying the device driver version

The read-only attribute "version" in sysfs provides the version of the zfcp device
driver:

```
/sys/bus/ccw/drivers/zfcp/version
```

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 269)
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.

When you set an FCP channel offline, the port and LUN subdirectories are
preserved but it is unregistered from the SCSI stack and its attribute values are no
longer valid.

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 8 summarizes the relevant attributes.

Table 8. Attributes with adapter information

lic_version	Hardware microcode level
in_recovery	Shows if adapter is in recovery (0 or 1)
failed	Shows if adapter is in failed state (0 or 1)
online	Shows if adapter is online (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

Table 9. Relevant transport class attributes, fc_host attributes

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 10. Relevant transport class attributes, fc_host statistics

reset_statistics	Writeable attribute to reset statistic counters.
seconds_since_last_reset	Seconds since last reset of statistic counters.
tx_frames	Transmitted FC frames.

Table 10. Relevant transport class attributes, fc_host statistics (continued)

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.
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 8 on page 59.

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 9 on page 59.

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 299) 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"
```

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/0.0.3d0c/host0/fc_host:host0/port_name
0x5005076401c08f98

# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/host0/fc_host:host0/permanent_port_name
0x5005076401c08f98
```

Alternatively you can use **lszfcp** (see "lszfcp - List zfcp devices" on page 299) to
display the above attributes:

```
# lszfcp -b 0.0.3d0c -a
0.0.3d0c host0
Bus = "ccw"
    availability       = "good"
    ...
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"
    ...
```

The example shows that permanent_port_name is the same as port_name, and the subchannel does not operate in NPIV mode.

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
```

Configuring and removing ports

Before you start: The FCP channel must be online.

To configure a port for an FCP channel write the port's WWPN to the FCP channel's port_add attribute. Issue a command of this form:

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

where:

<device_bus_id>

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

<wwpn>   is the WWPN of the port to be added.

Adding a port creates a directory in `/sys/bus/ccw/drivers/zfcp/<device_bus_id>` with the WWPN as the directory name.

You cannot read from the port_add attribute. 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" on page 65) or if the port is in use, for example, by error recovery.

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 configured and then the port with WWPN 0x500507630303c562 is removed.

```
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
# echo 0x500507630300c562 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/port_add
# 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
```

Displaying port information

For each target port, there is a number of read-only attributes in sysfs that provide port information. Table 11 on page 64 summarizes the relevant attributes.

Table 11. Attributes with port information

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)
failed	Shows if port is in failed state (0 or 1)

Table 12. Transport class attributes with port information

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 11.

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 12.

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 and removing ports" on page 62.

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
```

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.

<device_bus_id>

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

<wwpn>   is the WWPN of the target port.

Adding a SCSI device creates a directory in `/sys/bus/ccw/drivers/zfcp/`*<device_bus_id>*`/`*<wwpn>* with the LUN as the directory name.

You cannot read from the unit_add attribute. List the contents of `/sys/bus/ccw/drivers/zfcp/`*<device_bus_id>*`/`*<wwpn>* to find out which SCSI devices are currently configured for the port.

Adding a SCSI device also registers the device with the SCSI stack and creates a sysfs entry in the SCSI branch (see "Mapping the representations of a SCSI device in sysfs").

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.

Figure 14 on page 67 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.
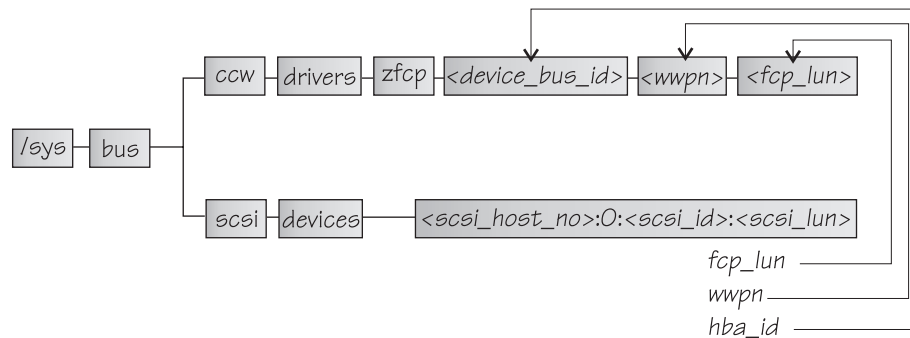
Figure 14. SCSI devices in sysfs

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 299).

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 on 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 13 summarizes the relevant attributes.

Table 13. Attributes with device access information

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).

Table 13. Attributes with device access information (continued)

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)
failed	Shows if unit is in failed state (0 or 1)

Table 14. SCSI device class attributes

device_blocked	Flag that indicates if device is in blocked state (0 or 1).
iocounterbits	The number of bits used for I/O counters.
iodone_cnt	The number of completed or rejected scsi commands.
ioerr_cnt	The number of SCSI commands that completed with an error.
iorequest_cnt	The number of SCSI commands.
queue_depth	The depth of the queue for this SCSI device.
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.
state	Shows the state of SCSI devices (writeable attribute). The value is one of the following: • created • running • cancel • quiesce • offline • blocked
timeout	Shows the timeout for SCSI commands sent on this device (writeable attribute).
delete	Triggers deletion of a SCSI device (write-only attribute).
rescan	Triggers a re-scan of a SCSI device (write-only attribute).

Issue a command of this form to read an attribute:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<scsi_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.
<scsi_lun>	is the FCP LUN of the SCSI device.
<attribute>	is one of the attributes in Table 13 on page 67.

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 14 on page 68.

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"
```

Finding the major and minor numbers for a device

You can find the major and minor numbers of a SCSI device and of SCSI partitions from the device representation in the sysfs SCSI branch (see "Mapping the representations of a SCSI device in sysfs" on page 66).

```
/sys/bus/scsi/devices/<scsi_device>/block:sd<x>/dev
```


Figure 15. Major/minor numbers for SCSI devices in sysfs

In Figure 15, *<scsi_device>* is the directory that represents a SCSI device (compare Figure 14 on page 67). If the disk is partitioned, the block directory that follows contains directories of the form *<name><n>* that represent the partitions. *<name>* is a standard name that the SCSI stack has assigned to the SCSI device and *<n>* is a positive integer that identifies the partition.

Both the block directory and the directories that represent the partitions contain an attribute dev. Read the dev attribute to find out the major and minor numbers for the entire device or for an individual partition. The value of the dev attributes is of the form *<major>*:*<minor>*.

Example
The following command shows a major of 8 and a minor of 0 for the SCSI device 0:0:1:1:

```
# cat /sys/bus/scsi/devices/0:0:1:1/block:sda/dev
8:0
```

Assuming that the device has three partitions sda1, sda2, and sda3, the following commands show the respective major and minor numbers:

```
# cat /sys/bus/scsi/devices/0:0:1:1/block:sda/sda1/dev
8:1
# cat /sys/bus/scsi/devices/0:0:1:1/block:sda/sda2/dev
8:2
# cat /sys/bus/scsi/devices/0:0:1:1/block:sda/sda3/dev
8:3
```

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
```

Removing SCSI devices

To remove a SCSI device from a target port you need to first unregister the device from the SCSI stack and then remove it from the target port.

You unregister the device 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
```

You can then 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.

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.

```
# echo 1 > /sys/bus/scsi/devices/0:0:1:1/delete
# echo 0x4010403200000000 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/unit_remove
```

Scenario

The following scenario describes the life-cycle of a SCSI device with LUN
0x4010403200000000. The device is attached through an FCP channel with device
bus-ID 0.0.3d0c and accessed through a target port 0x500507630300c562.

The FCP channel is set online, then port and device are configured.

```
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/online
# echo 0x500507630300c562 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/port_add
# echo 0x4010403200000000 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/unit_add
```

SCSI device and port are now to be removed. First the SCSI device must be
unregistered from the SCSI stack. Find out the SCSI device for the zfcp unit as
follows:

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

Delete SCSI device 0:0:0:0 and then remove the zfcp unit and port.

```
# echo 1 > /sys/bus/scsi/devices/0:0:0:0/delete
# echo 0x4010403200000000 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/unit_remove
# echo 0x500507630300c562 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/port_remove
```

API provided by the zfcp HBA API support

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

Table 15 gives an overview of available packages of zfcp HBA API and the binaries
that are installed with those versions.

Table 15. zfcp HBA API library versions

Package	Installed binaries
zfcp HBA API library 1.4	san_disc command
zfcp HBA API library 1.3	libzfcphbaapi.so, san_disc command
zfcp HBA API library 1.2 and earlier	libzfcphbaapi.so

Note: The zfcp-specific HBA API library libzfcphbaapi is a proprietary solution for
an HBA API library. It is not installed with new packages of lib-zfcp-hbaapi.
Instead the san_disc command is statically linked against libzfcphbaapi.

Functions provided

The zfcp HBA API (see "zfcp HBA API (FC-HBA) support" on page 51) is defined in
the Fibre Channel - HBA API (FC-HBA) specification (see http://www.t11.org).

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

- HBA_GetVersion()
- HBA_LoadLibrary()
- HBA_FreeLibrary()
- HBA_GetWrapperLibraryAttributes()
- HBA_GetVendorLibraryAttributes()
- HBA_GetNumberOfAdapters()
- HBA_GetAdapterName()
- HBA_OpenAdapter()
- HBA_CloseAdapter()
- HBA_RefreshInformation()
- HBA_RefreshAdapterConfiguration()
- HBA_GetAdapterAttributes()
- HBA_GetAdapterPortAttributes()
- HBA_GetDiscoveredPortAttributes()
- HBA_SendScsiInquiry()
- HBA_SendReadCapacity()
- HBA_SendReportLUNs()
- HBA_GetFcpTargetMapping()
- HBA_SendCTPassThru()
- HBA_GetRNIDMgmtInfo()
- HBA_GetEventBuffer()
- HBA_SendRNID()
- HBA_SendRLS()
- HBA_SendRPS()
- HBA_GetPortStatistics()
- HBA_ResetStatistics()

All other FC-HBA functions return status code
HBA_STATUS_ERROR_NOT_SUPPORTED where possible. The exception are the
following commands that are not implemented: HBA_GetSBTargetMapping(),
HBA_GetSBStatistics(), and HBA_SBDskCapacity().

Restriction: ZFCP HBA API for Linux 2.6 can access only adapters, ports and units
that are configured in the operating system. As an exception, ELS commands can
also be sent to ports that are not configured within zfcp.

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. 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.

LIB_ZFCP_HBAAPI_DEVICE_FILE
> specifies the name of the misc device file to be used. The default is
> `/dev/hba_api`.

Chapter 6. Channel-attached tape device driver

The Linux on System z tape device driver supports channel-attached tape devices.

SCSI tape devices attached through a System z9 or zSeries 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 79)
- Message display support ("tape390_display - display messages on tape devices and load tapes" on page 323)
- 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 Linux 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 seeks.

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

	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 297) 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, there are directories that provide information on the major/minor assignment. 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
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/ntibm0/dev
254:1
# cat /sys/class/tape390/ntibm1/dev
254:2
# cat /sys/class/tape390/ntibm1/dev
254:3
# cat /sys/class/tape390/ntibm2/dev
254:4
# cat /sys/class/tape390/ntibm2/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
254:1
# cat /sys/block/btibm2/dev
254:2
```

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

Creating device nodes

User space programs access tape devices by *device nodes*. Your distribution might create these device nodes for you or provide udev to create them (see "Device nodes provided by udev" on page 4).

If no device nodes are created for you, you need to create them yourself, for example, with the **mknod** command. Refer to the **mknod** man page for further details.

Tip: Use the device names to construct your nodes (see "Tape naming scheme" on page 76).

Example: Defining standard tape nodes

In this example, the tape major number is assumed to be 254 for both the character and block devices. The nodes use the standard form /dev/*<device_name>* for the device nodes and the assignment of minor numbers is according to Table 16 on page 76.

```
# mknod /dev/ntibm0 c 254 0
# mknod /dev/rtibm0 c 254 1
# mknod /dev/btibm0 b 254 0
# mknod /dev/ntibm1 c 254 2
# mknod /dev/rtibm1 c 254 3
# mknod /dev/btibm1 b 254 2
# mknod /dev/ntibm2 c 254 4
# mknod /dev/rtibm2 c 254 5
# mknod /dev/btibm2 b 254 4
...
```

Examples for udev-created tape device nodes

> **Note**
>
> The format of the nodes that udev creates for you depends on distribution-specific configuration files that reside in /etc/udev/rules.d. If you use udev, be sure that you use the nodes according to your distribution. The following examples use hypothetical nodes that are provided for illustration purposes only.

If your distribution provides udev, you can use udev to create tape device nodes for you. udev is a utility program that can use the device information in sysfs (see Chapter 2, "Devices in sysfs," on page 9) to create device nodes.

Apart from creating device nodes that are based on the device names, udev can create additional device nodes that are based on, for example, on device bus-IDs. Unless you change the device bus-IDs of your devices, device nodes that are based on a device bus-ID remain the same and map to the same device, even if the device name of a tape device has changed (for example, after rebooting). udev keeps track of the mapping of the device name and the actual devices for you and so helps you ensure that you are addressing the device you intend to.

For example, the configuration file might instruct udev to create two nodes for each logical device, the standard node and a node that is based on the device bus-ID. For a tape device with a device bus-ID 0.0.01ac it might create:

For the non-rewinding character device:
- `/dev/ntibm0` (standard device node according to the tape naming scheme)
- `/dev/tape/0.0.01ac/non-rewinding`

For the rewinding character device:
- `/dev/rtibm0` (standard device node according to the tape naming scheme)
- `/dev/tape/0.0.01ac/rewinding`

For the block device:
- `/dev/btibm0` (standard device node according to the tape naming scheme)
- `/dev/tape/0.0.01ac/block`

The next section shows how such nodes can be used to access a tape device by device bus-ID, regardless of its device name.

Accessing tapes by bus-ID

You can use device nodes that are based on your tape devices' device bus-IDs to be sure that you access a tape device with a particular bus-ID, regardless of the device name that is assigned to it.

Example

The examples in this section assume that udev provides device nodes as described in "Examples for udev-created tape device nodes" on page 78. To assure that you are addressing a device with bus-ID 0.0.01ac you could make substitutions like the following:

Instead of issuing:

```
# mt -f /dev/ntibm0 unload
```

issue:

```
# mt -f /dev/tape/0.0.01ac/non-rewinding unload
```

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).

Building a kernel with the tape device driver

This section is intended for those who want to build their own kernel. It describes the options you must select in the Linux configuration menu to include the tape device driver.

The tape device driver is available as a base component with supplementary components for particular hardware and for the block device mode.

Figure 16 summarizes the kernel configuration menu options that are relevant to the tape device driver:

```
Character device drivers
   S/390 tape device support                (CONFIG_S390_TAPE)
   ├─Support for tape block devices         (CONFIG_S390_TAPE_BLOCK)
   ├─Support for 3480/3490 tape hardware    (CONFIG_S390_TAPE_34XX)
   └─Support for 3590 tape hardware         (CONFIG_S390_TAPE_3590)
```

Figure 16. Tape kernel configuration menu options

CONFIG_S390_TAPE
> This option is required if you want to work with channel-attached tape devices. It can be compiled into the kernel or as a separate module, tape.

CONFIG_S390_TAPE_BLOCK
> This base component option allows you to use channel-attached tapes as block devices.

CONFIG_S390_TAPE_34XX
> This option can be compiled into the kernel or as a separate module, tape_34xx.

CONFIG_S390_TAPE_3590
> This option can be compiled into the kernel or as a separate module, tape_3590.

Setting up the tape device driver

There are no kernel or module parameters for the tape device driver. This section describes how to load the tape modules, where applicable.

For information on device nodes see "Tape naming scheme" on page 76.

Loading the tape device driver

If the tape_34xx or tape_3590 device drivers have not been built into the kernel, you must load the kernel modules before you can work with the 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 ─┘                                         │
│                                                                        │
└────────────────────────────────────────────────────────────────────────┘
```
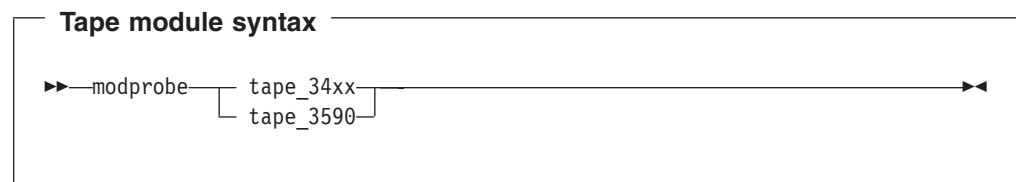
Refer to the **modprobe** man page for details on **modprobe**.

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 21, "Channel measurement facility," on page 231.

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 323.

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 269) 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 "Creating device nodes" on page 78) 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.

Your distribution might use udev to create alternative device nodes that distinguish devices by the physical device's bus ID instead of the index number. If you are using such device nodes you do not need to know the index number (see "Examples for udev-created tape device nodes" on page 78).

If you need to know the index number, issue a command of this form:

```
# lstape <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
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

Each physical tape device is represented in a sysfs directory of the form

```
/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 on the physical device. The attributes: blocksize, state, operation, and medium_state, might not show the current values if the device is offline.

Table 17. Tape device attributes

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 21, "Channel measurement facility," on page 231)
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 There are several other operation codes, for example, for rewind and seek.
medium_state	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 /bus/ccw/devices/<device_bus_id>/<attribute>
```

where *<attribute>* is one of the attributes of Table 17.

Tip: You can display a summary of this information by using the **lstape** command (see "lstape - List tape devices" on page 297).

Example

The following sequence of commands reads the attributes for a physical tape device with a device bus-ID 0.0.015f:

```
# cat /bus/ccw/devices/0.0.015f/online
1
# cat /bus/ccw/devices/0.0.015f/cmb_enable
0
# cat /bus/ccw/devices/0.0.015f/cutype
3480/01
# cat /bus/ccw/devices/0.0.015f/devtype
3480/04
# cat /bus/ccw/devices/0.0.015f/blocksize
0
# cat /bus/ccw/devices/0.0.015f/state
UNUSED
# cat /bus/ccw/devices/0.0.015f/operation
---
# cat /bus/ccw/devices/0.0.015f/medium_state
1
```

Issuing an **lstape** command for the same device yields:

```
# lstape 0.0.015f
TapeNo  BusID     CuType/Model DevType/Model  BlkSize State   Op    MedState
2       0.0.015f  3480/01      3480/04        auto    UNUSED  ---   LOADED
```

Enabling compression

You can use the **mt** command to control Improved Data Recording Capability (IDRC) compression.

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 323) 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 323). 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.
- Device nodes are available as defined in "Example: Defining standard tape nodes" on page 78.
- 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
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/ntibm0 rewind
# mt -f /dev/ntibm0 fsf 1
```

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

```
# mount -t iso9660 -o ro,block=2048 /dev/btibm0 /mnt
```

Chapter 7. XPRAM device driver

The zSeries architecture in 31-bit mode and the S/390 architecture support only 2 GB (gigabytes) of main storage (main memory). To overcome this limitation additional storage can be declared and accessed as expanded storage. For compatibility reasons, expanded storage can also be declared in the 64-bit mode of zSeries.

The XPRAM device driver is a block device driver that enables Linux on System z to 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 xpram*<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 xpram0. For partitioned expanded storage, the *<n>* in the device name denotes the (n+1)th partition. For example, the first partition is called xpram0, the second xpram1, and the 32nd partition is called xpram31.

Table 18. XPRAM device names, minor numbers, and partitions

Minor	Name	To access
0	xpram0	the first partition or the entire expanded storage if there are no partitions
1	xpram1	the second partition
2	xpram2	the third partition
... <n>	... xpram<n>	... the (<n>+1)th partition
...	...	...
31	xpram31	the 32nd partition

Creating device nodes

User space programs access XPRAM devices by *device nodes*. Your distribution might create these device nodes for you or provide udev to create them (see "Device nodes provided by udev" on page 4).

If no device nodes are created for you, you need to create them yourself, for example, with the **mknod** command. Refer to the **mknod** man page for further details.

Tip: Use the device names to construct your nodes (see "XPRAM partitions and device nodes" on page 87).

Example: Defining standard XPRAM nodes

The nodes use the standard form /dev/*<device_name>* for the device nodes and the assignment of minor numbers is according to Table 18 on page 87.

```
# mknod /dev/xpram0 b 35 0
# mknod /dev/xpram1 b 35 1
# mknod /dev/xpram2 b 35 2
...
# mknod /dev/xpram30 b 35 30
# mknod /dev/xpram31 b 35 31
```

XPRAM use for diagnosis

Issuing an IPL command to reboot Linux on System z 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 by an IML (power off/on).

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.

Building a kernel with the XPRAM device driver

This section is intended for those who want to build their own kernel.

To build a kernel with XPRAM support you need to select option CONFIG_BLK_DEV_XPRAM in the configuration menu:

```
Block devices
    XPRAM disk support                              (CONFIG_BLK_DEV_XPRAM)
```

Figure 17. XPRAM kernel configuration menu option

The XPRAM support is available as a module, xpram, or built-in.

Setting up the XPRAM device driver

This section describes the parameters that you can optionally use to split the available expanded storage into partitions. The syntax is different for the kernel parameters and the corresponding module parameters. By default the entire expanded storage is treated as a single partition.

See "Creating device nodes" on page 88 for information on the device nodes that you need to access the partitions.

Kernel parameters

This section describes how to configure the XPRAM device driver if it has been compiled into the kernel. You can optionally partition the available expanded storage by adding the xpram.parts kernel parameter to the kernel parameter line.

XPRAM kernel parameter syntax

▶▶──xpram.parts=*<number_of_partitions>*──────────────────────────────▶◀
                                          └─,─┬─────────────┬─┘
                                           ◄─,─*<partition_size>*─┘
---

where:

<number_of_partitions>
    is an integer in the range 1 to 16 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 may be blank, specified as a decimal value, or a hexadecimal value preceded by 0x, and may be qualified by a magnitude:
    • k or K for Kilo (1024) is the default
    • m or M for Mega (1024×1024)
    • g or G for Giga (1024×1024×1024)

    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 (hex 800M), partition 4 has 4 GB, 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.

  ```
  xpram.parts=4,0x800M,0,0,4g
  ```

- 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 storage.

  ```
  xpram.parts=3,,512
  ```

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

  ```
  xpram.parts=2
  ```

Module parameters

This section describes how to load and configure the XPRAM device driver if it has been compiled as a separate module. You can optionally partition the available expanded storage by using the devs and sizes module parameters when you load the xpram module.
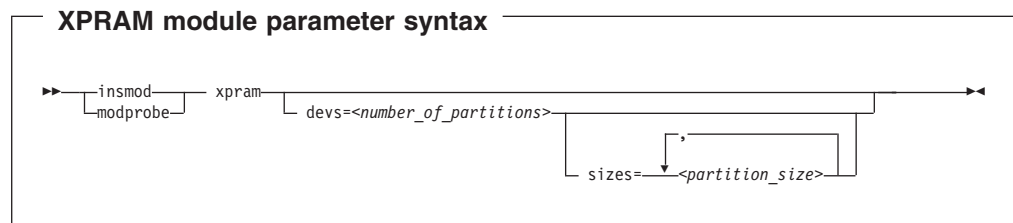
XPRAM module parameter syntax

where:

<number_of_partitions>
   is an integer in the range 1 to 16 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. Network device drivers

This part describes the following device drivers:

- Chapter 8, "qeth device driver for OSA-Express (QDIO) and HiperSockets"
- Chapter 9, "LAN channel station device driver"
- Chapter 10, "CTCMPC device driver"

The CLAW, CTC, and NETIUCV device drivers are deprecated and their descriptions have been removed. For information about the device drivers, see earlier versions of this book. The device drivers are still available for backward compatibility.

Instead of the deprecated drivers, use the qeth device driver for OSA-Express (QDIO) and HiperSockets as follows:

- As an alternative for CTC, use an OSA feature that is configured for QDIO mode.
- As an alternative for virtual CTC and NETIUCV, use guest LAN Hipersockets or guest LAN type QDIO.
- As an alternative for CTC inside a CEC use Hipersockets.

> **Note**
>
> For prerequisites and restrictions for these device drivers refer to the kernel 2.6 October 2005 stream pages on developerWorks at:
>
> `ibm.com/developerworks/linux/linux390/october2005_recommended.html`

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

The qeth network device driver supports System z OSA-Express and OSA-Express2 features in QDIO mode and HiperSockets as follows:

- OSA-Express:
  - Fast Ethernet
  - 1000Base-T Ethernet
  - Gigabit Ethernet
  - Token Ring (zSeries only)
  - ATM
- OSA-Express2:
  - Gigabit Ethernet
  - 10 Gigabit Ethernet
  - 1000Base-T Ethernet (System z9 only)
- HiperSockets:
  - Virtual network devices that provide virtual networks within a System z mainframe.

Most of the qeth device driver parameters are common to HiperSockets and to OSA-Express devices in QDIO mode.

OSA-Express is a LAN adapter that is used to connect a System z9, zSeries, or S/390® mainframe to a LAN. In addition it may serve as a Network Control Program (NCP) adapter for an internal ESCON/CDLC interface to another host operating system. This feature is exploited by the IBM Communication Controller for Linux on System z9 (CCL). 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*, SG24-6298-01

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

Features

The qeth device driver supports the following functions:

- (OSA-Express2 only) Up to 640 TCP/IP stacks or connections per dedicated CHPID or 640 total stacks across multiple LPARs using a shared or spanned CHPID.
- Virtual QDIO guest LAN environments
- HiperSockets
- Guest LANs using virtual HiperSockets
- OSA Network Control Program (ESCON/CDLC bridge) (System z9 only)
- Auto-detection of qeth subchannels (see "Overview of the steps for setting up a qeth group device" on page 97)
- Internet Protocol Version 4 (IPv4)
- Internet Protocol Version 6 (IPv6) for Ethernet interfaces (see "Support for IP Version 6 (IPv6)" on page 103)

- Routing (see "Setting up a Linux router" on page 109)
- Checksumming (see "Setting the checksumming method" on page 112)
- Priority queueing for OSA-Express CHPID in QDIO mode (see "Using priority queueing" on page 113)
- Broadcast (see "Setting the scope of Token Ring broadcasts" on page 115 and "Faking broadcast capability" on page 115)
- Query and purge of ARP data (see "qetharp - Query and purge OSA and HiperSockets ARP data" on page 303)
- SNMP via the OSA-Express feature (see "osasnmpd – Start OSA-Express SNMP subagent" on page 301and Chapter 23, "OSA-Express SNMP subagent support," on page 237)
- IP address takeover (see "Taking over IP addresses" on page 121)
- VLAN (see "Scenario: Virtual LAN (VLAN) support" on page 132)
- Virtual IP addresses for OSA-Express CHPID in QDIO mode (see "Scenario: VIPA – minimize outage due to adapter failure" on page 125)
- DHCP for OSA-Express CHPID in QDIO mode (see "Setting up for DHCP with IPv4" on page 141)
- MAC-based addressing for qeth devices (see "MAC address handling for IPv4 with the layer2 option" on page 102)
- Guest LAN sniffer (see "Setting up a Linux guest as a z/VM guest LAN sniffer" on page 143)

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 packets.

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.
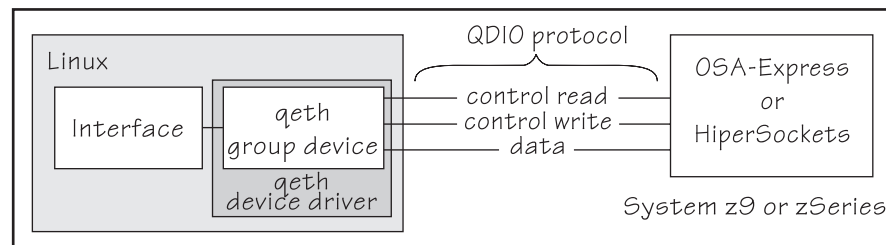

Figure 18. I/O subchannel interface

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   must be even.

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 CHPID 0xfc, then you can configure 0.0.fc00,0.0.fc01,0.0.fc02 and 0.0.fc04,0.0.fc05,0.0.fc06 with different attribute values, for example 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.

Hint: After booting Linux, each qeth device bus-ID is represented by a subdirectory in `/sys/bus/ccw/drivers/qeth/`. These subdirectories are the named with the bus IDs of the devices. For example, a qeth device with bus-ID 0.0.fc00 is represented as `/sys/bus/ccw/drivers/qeth/0.0.fc00`

There are several steps you need to perform until user space applications on your Linux instance can use a qeth group device:

- Create the qeth group device.
- Configure the device.
- Set the device online.
- Activate the device.

These tasks and the configuration options are described in detail in "Working with the qeth device driver" on page 105.

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 (including the OSA-Express ATM device when emulating Ethernet in QDIO mode).

hsi*<n>*
   for HiperSockets devices.

tr*<n>*   for Token Ring features.

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 and Token Ring 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 Token Ring feature is set online and there is already one LCS Token Ring feature online, the LCS feature is named "tr0" and the qeth feature is named "tr1". See also "LCS interface names" on page 145.

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 118 and "Finding out the bus ID of a qeth interface" on page 118 provide information on how to map device bus-IDs and interface names.

MAC address handling for IPv4

In LAN environments, data packets find their destination through Media Access Control (MAC) addresses in their Logical Link Control (LLC) header (see Figure 19).
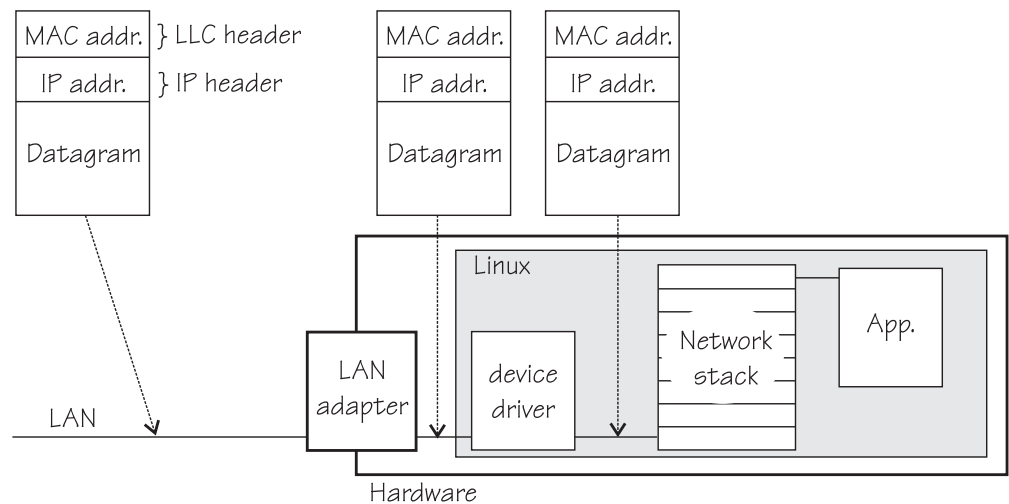
Figure 19. Standard IPv4 processing

MAC address handling as shown in Figure 19) applies to non-mainframe environments and a mainframe environment with an OSA-Express adapter where the layer2 option is enabled (see "MAC address handling for IPv4 with the layer2 option" on page 102).

For IPv6, both inbound and outbound packets are complete packets with LLC headers. The OSA-Express adapter in QDIO mode passes complete packets to the Linux image and the device driver lets the network stack compose packets with an LLC header.

For IPv4 without the layer2 option, the adapter removes the LLC header before passing the packet to the network stack of the recipient Linux image (see Figure 20 on page 99).
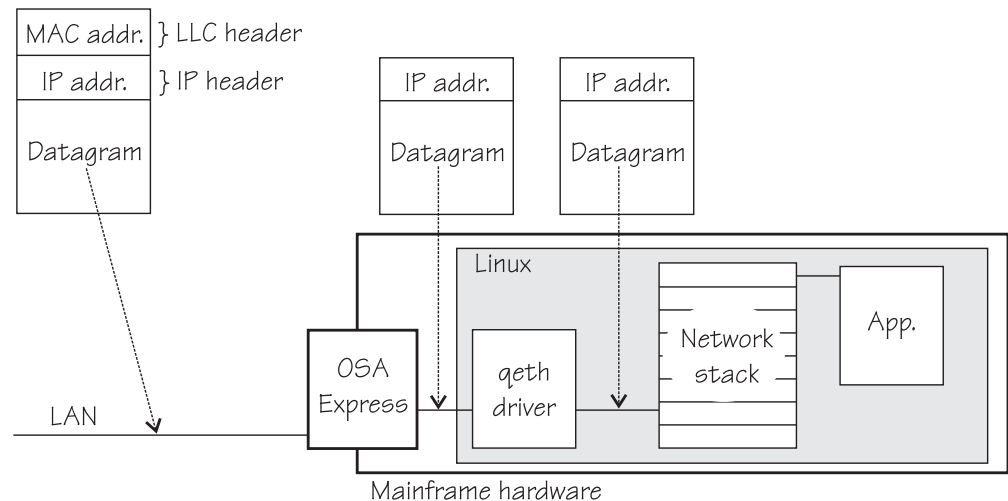
Figure 20. IPv4 processing by OSA-Express without layer2

For outbound packets, without the layer2 option, the adapter adds the LLC header to IPv4 packets with the destination MAC address.

Letting the OSA-Express hardware handle the LLC header allows multiple operating systems to share an OSA-Express adapter. Usually, LLC processing by the OSA-Express adapter also yields better performance than letting the Linux images that share the OSA-Express handle the LLC header themselves.

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 LLC headers
* ARP processing

LLC headers
Without the layer2 option, the OSA-Express adapter in QDIO mode removes the LLC 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 LLC headers for outgoing IPv4 packets and removes them from incoming IPv4 packets, the operating systems' network stacks only send and receive IPv4 packets without LLC headers.

This can be a problem for applications that expect LLC headers. For examples of how such problems can be resolved see:
* DHCP (see "Setting up for DHCP with IPv4" on page 141)
* tcpdump (see "Setting up for tcpdump with IPv4" on page 142)

ARP

The OSA-Express adapter in QDIO mode responds to Address Resolution Protocol (ARP) requests for all registered IP addresses.

ARP is a TCP/IP protocol that translates 32-bit IP addresses into the corresponding hardware addresses. For example, for an Ethernet, the hardware addresses are 48-bit Ethernet Media Access Control (MAC) addresses. The mapping of IP 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 IP address, the corresponding MAC address is copied into the LLC header and the packet is added to the appropriate interface's output queue. If the entry is not found, the ARP functions retain the IP 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.

This short overview is intended as background information for the sections that follow and is by no means an exhaustive description of the ARP protocol. Consult the TCP/IP literature for more details on ARP.

Faking LLC headers

Note: The information in this section is not applicable if you are using the layer2 option (see "MAC address handling for IPv4 with the layer2 option" on page 102). If you are using the layer2 option, the qeth device driver ignores your setting for faking LLC headers.

Before you start:
- This section applies to IPv4 only.
- The device must be offline while you enable faking LLC headers.
- If you are setting up an IPv4 interface for an OSA-Express CHPID in QDIO mode you must not enable the layer2 option (see "MAC address handling for IPv4 with the layer2 option" on page 102).

IPv4 packets within mainframe environments do not have LLC headers. HiperSockets do not use LLC headers and the OSA-Express QDIO microcode removes LLC headers from incoming packets (see "MAC address handling for IPv4" on page 98). This is a problem for network programs that require incoming packets with LLC headers.

The fake_ll attribute instructs qeth to insert a fake LLC header in all incoming packets. The packets are then passed to the Linux network stack and finally to the recipient programs or applications (Figure 21 on page 101).
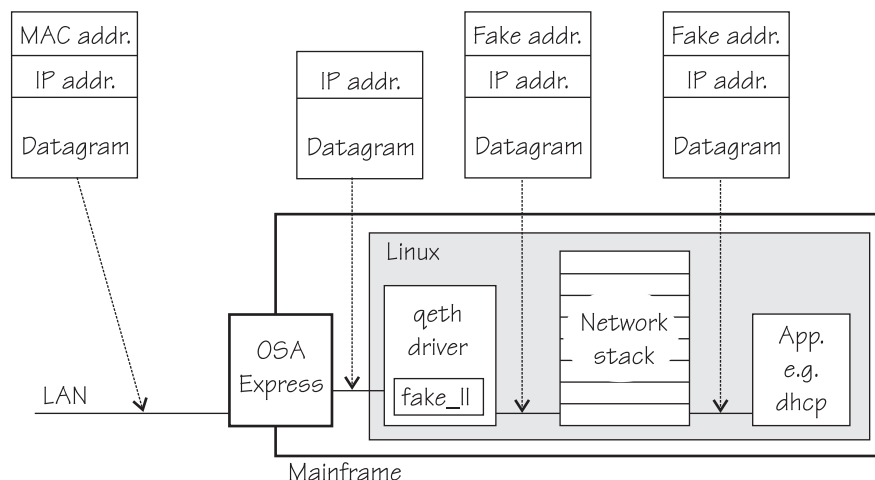
Figure 21. qeth with fake_ll option for incoming packets

Network programs that expect incoming packets to have a LLC header can then be used as usual, without any patches. Examples for programs that expect an LLC header are the DHCP server program *dhcp* and client program *dhcpcd* (see "Setting up for DHCP with IPv4" on page 141).

Shortcomings of fake_ll: An obvious disadvantage of fake_ll is, that it introduces additional processing and, thus, has an adverse effect on performance.

Because fake_ll is a qeth option, it also cannot supply fake LLC headers for programs that intercept outgoing packets before they have reached the qeth driver.

The OSA-Express adapter in QDIO mode suppresses the construction of LLC headers in the associated network stacks. Outgoing packets that originate from programs that build their own LLC headers and bypass the Linux network stack have LLC headers. Outgoing packets from all programs that work through the network stack do not have an LLC header until they reach the OSA-Express adapter.
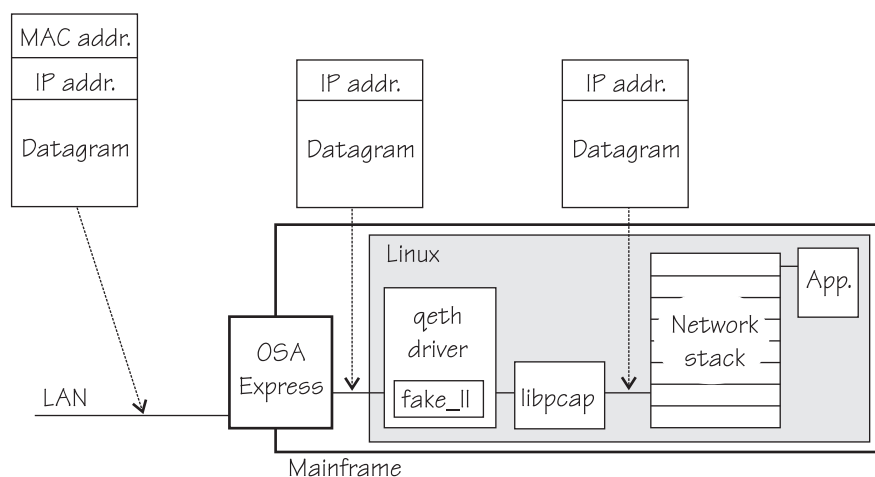

Figure 22. qeth with fake_ll option for outgoing packets

An example of a program that expects LLC headers in outgoing packets is *libpcap* a program that captures outgoing packets for *tcpdump*.

See "Setting up for tcpdump with IPv4" on page 142 for information on how to make tcpdump work correctly for IPv4.

Setting the fake_ll attribute: Set the value of the device's fake_ll attribute to "1" to insert fake LLC headers in incoming IPv4 packets and to "0" to suppress inserting fake LLC headers. By default, the qeth device driver does not insert fake LLC headers.

Example: To make a device 0xa110 insert fake LLC headers in incoming IPv4 packets issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a110/fake_ll
```

MAC address handling for IPv4 with the layer2 option

The layer2 option stops the OSA-Express adapter from stripping the MAC addresses from incoming packets. Incoming and outgoing packets are complete with an LLC header at all stages between the Linux network stack and the LAN as shown in Figure 19 on page 98. This layer 2 based forwarding requires unique MAC addresses for all concerned Linux instances.

Be aware that in conjunction with the *layer2* option, the following cannot be configured as described in the respective sections:
- Router definitions (see "Setting up a Linux router" on page 109)
- HiperSockets network concentrator (see "HiperSockets Network Concentrator" on page 135)
- IP address takeover (see "Taking over IP addresses" on page 121)
- Proxy ARP (see "Configuring a device for proxy ARP" on page 123)
- VIPA (see "Configuring a device for virtual IP address (VIPA)" on page 124)
- Stateless autoconfiguration in IPv6 (see "Support for IP Version 6 (IPv6)" on page 103)

Accordingly, you cannot use the following commands if you are using the *layer2* option:
- qetharp (see "qetharp - Query and purge OSA and HiperSockets ARP data" on page 303)
- qethconf (see "qethconf - Configure qeth devices" on page 305)

If your environment does not fulfill the requirements, or you want to use a feature that you cannot use in conjunction with the *layer2* option, consider using the *fake_ll* option instead (see "Faking LLC headers" on page 100).

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 need to assign the MAC addresses yourself. Consult your distribution documentation on how to assign a MAC address. Alternatively, you can change it 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.

Support for IP Version 6 (IPv6)

IPv6 applies only to the Ethernet interfaces of the OSA-Express adapter running in QDIO mode. IPv6 is not supported on HiperSockets nor on the OSA-Express Token Ring and ATM features.

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.

Without the layer2 option, (see "MAC address handling for IPv4 with the layer2 option" on page 102), stateless autoconfiguration generates unique IP addresses for all Linux instances, even if they share an OSA-Express adapter with other operating systems. With the layer2 option, each Linux instances is associated with a unique MAC addresses.

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).

Further information

For details on OSA-Express in QDIO mode refer to *OSA-Express Customer's Guide and Reference*, SA22-7935.

For information on guest LANs and virtual HiperSockets, refer to:
* `http://www.linuxvm.org/Info/HOWTOs/guestlan.html`
* *zSeries HiperSockets*, SG24-6816
* *z/VM CP Command and Utility Reference*, SC24-6008

For information on proxy ARP, visit: `http://www.sjdjweis.com/linux/proxyarp/`.

For information on layer2, refer to:
* *OSA-Express Implementation Guide*
* *Networking Overview for Linux on zSeries*
* *z/VM 5.1 Connectivity*

Building a kernel with the qeth device driver

This section is intended for those who want to build their own kernel. It describes the options you must select in the Linux configuration menu to include the qeth device driver.

Figure 23 on page 104 summarizes the kernel configuration menu options that are relevant to the qeth device driver:

```
Base setup
   QDIO support                                      (CONFIG_QDIO)
     └─Performance statistics in /proc      (CONFIG_QDIO_PERF_STATS)
Networking support
└─S/390 network device drivers
   └─Gigabit Ethernet device support                 (CONFIG_QETH)
      ├─IPv6 support for qeth                      (CONFIG_QETH_IPV6)
      ├─VLAN support for qeth                         (CONFIG_VLAN)
      └─Performance statistics in /proc      (CONFIG_QETH_PERF_STATS)
```

Figure 23. qeth kernel configuration menu options

CONFIG_QDIO
> This option provides the interface between the IBM mainframe and an OSA-Express CHPID in QDIO mode or a HiperSockets CHPID.
>
> It is required if you want to work with qeth devices. It depends on the common code option "TCP/IP networking" (CONFIG_INET).
>
> It can be compiled into the kernel or as a separate module, qdio.

CONFIG_QDIO_PERF_STATS
> This option gathers QDIO performance statistics in procfs.

CONFIG_QETH
> This option is required if you want to work with qeth devices. It can be compiled into the kernel or as a separate module, qeth. This option depends on the common code options, "Token Ring driver support" (CONFIG_TR), "Ethernet (10 or 100 Mbit)" (CONFIG_NET_ETHERNET), and "IP: multicasting"(CONFIG_IP_MULTICAST).

CONFIG_QETH_IPV6
> Provides IPv6 support for the qeth device driver.

CONFIG_VLAN
> Provides IEEE 802.1q VLAN support for the qeth device driver (see "Scenario: Virtual LAN (VLAN) support" on page 132).

CONFIG_QETH_PERF_STATS
> Gathers QDIO performance statistics in procfs.

If you want to configure qeth devices for VIPA (see "Configuring a device for virtual IP address (VIPA)" on page 124), you also need the common code option CONFIG_DUMMY. It can be compiled into the kernel or as a separate module, dummy.

Setting up the qeth device driver

There are no kernel or module parameters for the qeth device driver. qeth devices are set up using sysfs.

Loading the qeth device driver modules

If the qeth device driver has not been built into the kernel, you have to load it 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:

```
# modprobe qeth
```

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. Table 19 serves as both a task overview and a summary of the attributes and the possible values you can write to them. Underlined values are defaults.

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.

The layer2 option changes the way OSA-Express CHPIDs in QDIO mode handle MAC addresses (see "MAC address handling for IPv4" on page 98) and is applicable to IPv4 interfaces only. Layer2 is not compatible with some of the other attributes. Attributes that cannot be specified in conjunction with layer2 are highlighted in grey.

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

- if_name
- card_type
- buffer_count
- recover

Table 19. qeth tasks and attributes. If you specify the layer2 attribute, the attributes and values that are highlighted in grey are ignored and the defaults apply.

Task	Corresponding attributes	Possible attribute values
"Creating a qeth group device" on page 107	none	n/a
"Assigning a port name" on page 108	portname	any valid port name
"Setting up a Linux router" on page 109	route4 route6	primary_router secondary_router primary_connector secondary_connector multicast_router no_router
"Setting the checksumming method" on page 112	checksumming	hw_checksumming sw_checksumming no_checksumming
"Providing Large Send" on page 112	large_send	no EDDP
		TSO

Table 19. qeth tasks and attributes (continued). If you specify the layer2 attribute, the attributes and values that are highlighted in grey are ignored and the defaults apply.

Task	Corresponding attributes	Possible attribute values
"Using priority queueing" on page 113	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
"Setting the Token Ring MAC address format" on page 114	canonical_macaddr	0 or 1
"Setting the scope of Token Ring broadcasts" on page 115	broadcast_mode	local all_rings
"Faking broadcast capability" on page 115	fake_broadcast	0 or 1
"Faking LLC headers" on page 100	fake_ll	0 or 1
"Setting the layer2 attribute" on page 116	layer2	0 or 1
"Adding additional hardware-header space" on page 116	add_hhlen	integer in the range 0 to 1024, the default is 0
"Specifying the number of inbound buffers" on page 116	buffer_count	integer in the range 8 to 128, the default is 16
"Specifying the relative port number" on page 117	portno	integer, either 0 or 1, the default is 0
"Finding out the type of your network adapter" on page 117	card_type	n/a, read-only
"Setting a device online or offline" on page 118	online	0 or 1
"Finding out the interface name of a qeth group device" on page 118	if_name	n/a, read-only
"Finding out the bus ID of a qeth interface" on page 118	none	n/a
"Activating an interface" on page 119	none	n/a
"Deactivating an interface" on page 121	none	n/a
"Taking over IP addresses" on page 121	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 123	rxip/add4 rxip/add6 rxip/del4 rxip/del6	IPv4 or IPv6 IP address
"Configuring a device for virtual IP address (VIPA)" on page 124	vipa/add4 vipa/add6 vipa/del4 vipa/del6	IPv4 or IPv6 IP address
"Recovering a device" on page 125	recover	1

Task	Corresponding attributes	Possible attribute values

Tips:

- Instead of using the attributes for IPA, proxy ARP and VIPA directly, use the **qethconf** command.
- Your distribution might also provide a distribution-specific configuration tool. Refer to your distribution documentation for distribution-specific alternatives.

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.

Tips:

- Work through one of the paths that are based on the device bus-ID.
- Your distribution might provide a distribution-specific configuration file through which you can set the attributes. Refer to your distribution documentation for distribution-specific information.

The following sections describe the tasks in detail.

Creating a qeth group device

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.
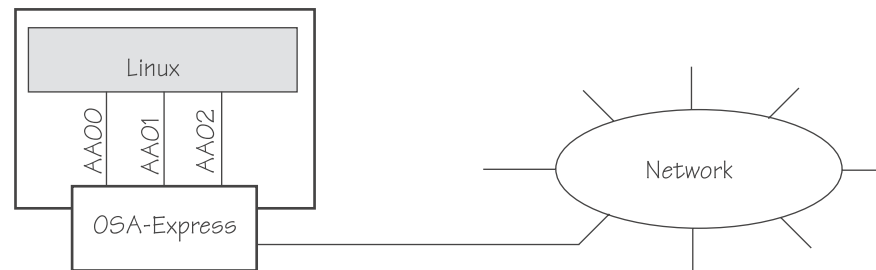
Note: If you have defined an OSA-Express CHPID in QDIO mode for a mainframe earlier than z990 you might need to set the portname attribute (see "Assigning a port name").

Example
In this example, a single OSA-Express CHPID in QDIO mode is used to connect a Linux instance to a network.

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.

Assigning a port name

Before you start:
- This section does not apply to:
  - HiperSockets and OSN CHPIDs
  - z9 mainframes
  - z990 mainframes
  - z900 and z800 mainframes with a microcode level of at least Driver 3G - EC stream J11204, MCL032 (OSA level 3.33).
  - z/VM guest LAN environments with APAR VM63308 applied.
- The device must be offline while you assign the port name.

For S/390 mainframes and z900 or z800 mainframes that are not exempted by the conditions listed under "Before you start," you must associate each OSA-Express

CHPID in QDIO mode with a port name. The port name identifies the port for sharing by other operating system instances. The port name can be 1 to 8 characters long and must be uppercase. All operating system instances that share the port must use the same port name.

To assign a port name set the portname device group attribute to the name of the port. Issue a command of the form:
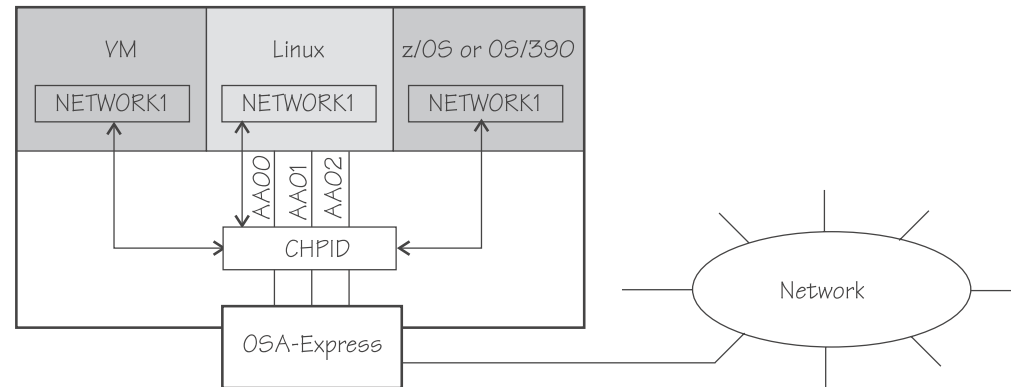
```
# echo <PORTNAME> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/portname
```

Example
In this example, two other mainframe operating systems share the OSA-Express CHPID in QDIO mode and use the port name "NETWORK1".

Mainframe configuration:

Linux configuration:

```
# echo NETWORK1 > /sys/bus/ccwgroup/drivers/qeth/0.0.aa00/portname
```

Setting up a Linux router

Note: If you enable the layer2 option (see "MAC address handling for IPv4 with the layer2 option" on page 102) the qeth device driver ignores the router settings described in this section. With the layer2 option enabled, set up a router as you would in a discrete server environment.

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.
- You must not enable the layer2 option if you are setting up an IPv4 interface for an OSA-Express CHPID in QDIO mode.

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:

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.

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).

A HiperSockets CHPID honors the following value, provided the microcode level supports this feature:

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.

Both types of CHPIDs honor:

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

Note: To configure Linux running as a VM guest 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
```

Depending on your distribution, you might be able to use distribution-specific configuration files. Refer to your distribution documentation for distribution-specific procedures.

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 VM guest to act as a router. This is usually done in procfs or in a configuration file; refer to your distribution manual for details.

Mainframe configuration:

System z9 or zSeries
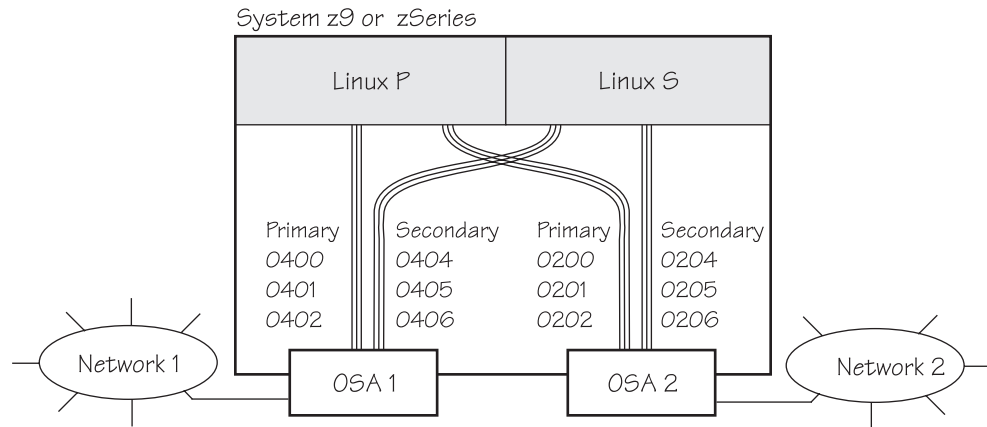
It is assumed that both Linux instances are configured as routers in their respective LPARs or in VM.

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 assign port names to the CHPIDs (For S/390 and certain microcode levels of z900 and z800 mainframes only, see "Assigning a port name" on page 108):

```
# echo NETWORK1 > /sys/bus/ccwgroup/drivers/qeth/0.0.0400/portname
# echo NETWORK2 > /sys/bus/ccwgroup/drivers/qeth/0.0.0200/portname
```

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 assign port names to the CHPIDs (For S/390 and certain microcode levels of z900 and z800 mainframes only, see "Assigning a port name" on page 108):

```
# echo NETWORK1 > /sys/bus/ccwgroup/drivers/qeth/0.0.0404/portname
# echo NETWORK2 > /sys/bus/ccwgroup/drivers/qeth/0.0.0204/portname
```

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
```

See "HiperSockets Network Concentrator" on page 135 for further examples.

Setting the checksumming method

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.

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
```

Providing Large Send

You can offload TCP segmentation from the Linux network stack to the OSA-Express2 features. Large Send can lead to enhanced performance and latency for interfaces with predominately large outgoing packets. 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 that connects to an interface though a real LAN.

You cannot use hardware TCP segmentation for HiperSockets or for connections between systems running on the same OSA-Express2 CHPID. 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 CHPID, will lead to unpredictable results for the receiving system.

> **Note:** If you are using the layer2 option (see "MAC address handling for IPv4 with the layer2 option" on page 102), the TSO value will be ignored.

EDDP
The qeth device driver provides the Large Send. There are no hardware related restrictions for device driver segmentation offload. It can be used with any qeth supported hardware.

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 20 shows how the qeth device driver maps service types to the available queues:

Table 20. IP service types and queue assignment for type of service queueing

Service type	Queue
Low latency	0
High throughput	1

Table 20. IP service types and queue assignment for type of service queueing (continued)

Service type	Queue
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
```

Setting the Token Ring MAC address format

Before you start:
- This section applies to OSA-Express CHPIDs in QDIO mode with the Token Ring feature only.
- The device must be offline while you set the Token Ring MAC address format.

The qeth group device can interpret MAC addresses in canonical or non-canonical form in a Token Ring. The default interpretation is the non-canonical form.

To set the MAC address format to canonical set the canonical_macaddr device group attribute to "1". To reset the MAC address format to non-canonical set the canonical_macaddr device group attribute to "0". Issue a command of the form:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/canonical_macaddr
```

Note: If you are using the layer2 option (see "MAC address handling for IPv4 with the layer2 option" on page 102), the qeth group device uses the default non-canonical MAC address format regardless of how you set the canonical_macaddr attribute.

Example
In this example, a device 0.0.a000 is instructed to interpret the Token Ring MAC addresses as canonical.

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

Setting the scope of Token Ring broadcasts

Before you start:

- This section applies to OSA-Express CHPIDs in QDIO mode with the Token Ring feature only.
- The device must be offline while you set the scope of Token Ring broadcasts.

To control the scope of Token Ring broadcasts set the broadcast_mode attribute to one of the following values:

local
    to restrict Token Ring broadcasts to the local LAN segment.

all_rings
    to allow Token Ring broadcasts to propagate to all rings that are connected via bridges. This is the default.

Issue a command of the form:

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

Note: If you are using the layer2 option (see "MAC address handling for IPv4 with the layer2 option" on page 102), the qeth group device uses the default Token Ring broadcast scope regardless of how you set the broadcast_mode attribute.

Example

In this example, the scope of broadcasts for a device 0.0.a000 is limited to the local LAN segment.

```
# echo local > /sys/bus/ccwgroup/drivers/qeth/0.0.a000/broadcast_mode
```

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.

There are processes, for example, the *gated* routing daemon, that 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
```

Note: If you are using the layer2 option (see "MAC address handling for IPv4 with the layer2 option" on page 102), the qeth group device does not fake broadcast capabilities, regardless of how you set the fake_broadcast attribute.

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
```

Setting the layer2 attribute

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.

Set the value of the device's layer2 attribute to "1" to make the OSA-Express adapter keep the MAC addresses in incoming IPv4 packets and to "0" to make the OSA-Express adapter remove LLC headers. By default, the OSA-Express adapter removes LLC headers from incoming IPv4 packets.

Example: To make the OSA-Express adapter keep MAC addresses in IPv4 packets for a device 0xa110:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a110/layer2
```

Switching on the layer2 option has far-reaching consequences. For more details, see "MAC address handling for IPv4 with the layer2 option" on page 102.

Adding additional hardware-header space

Before you start: The device must be offline while you add additional hardware-header space.

Some software makes use of free space in front of packets. For example, extra space can be beneficial for Linux virtual servers and IP tunneling.

To reserve additional hardware-header space in front of every packet in socket buffers set the add_hhlen attribute to an integer value up to 1024. The integer is the number of bytes to be added. Issue a command of the form:

```
# echo <integer> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/add_hhlen
```

Example

In this example, device 0.0.a000 is instructed to increase the hardware-header space by 8 bytes.

```
# echo 8 > /sys/bus/ccwgroup/drivers/qeth/0.0.a000/add_hhlen
```

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.

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

ATM adapters provide one physical port (port 0) and two logical ports (0 and 1) 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
```

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").

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. 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
```

Tip: you can also read the content of /proc/qeth to obtain a mapping for all qeth interfaces and devices.

Example

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

Finding out the bus ID of a qeth interface

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.

Tip: you can also use the **lsqeth** command (see "lsqeth - List qeth based network devices" on page 295) or read the content of /proc/qeth to obtain a mapping for all qeth interfaces and devices.

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" on page 118).
- You need to know the IP address you want to assign to the device.

The MTU range for OSA-Express CHPIDs in QDIO mode is 576 – 61440. However, depending on your medium and networking hardware settings, it might be restricted to 1492, 1500, 8992 or 9000. The recommended MTU size for OSA-Express CHPIDs in QDIO mode is 1492 (for Gigabit Ethernet and OSA-Express2 OSD 1000Base-T Ethernet: 8992 for jumbo frames). Choosing 1500 (or 9000 for Gigabit Ethernet or OSA-Express2 OSD 1000Base-T Ethernet jumbo frames) can cause performance degradation.

On 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

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 of the form "could not set IP address" or "duplicate IP address" in the kernel messages. You can display these messages with **dmesg**. For most distributions 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.

There are several setups that 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 121 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 123 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 124 for details.

You can use the **qethconf** command (see "qethconf - Configure qeth devices" on page 305) 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
```

Taking over IP addresses

This section describes how to configure for IP takeover if the layer2 option (see "MAC address handling for IPv4 with the layer2 option" on page 102) 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 addresses 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 zSeries operating system. IP takeover for HiperSockets CHPIDs is restricted to taking over addresses from other Linux instances in the same Central Electronics Complex (CEC).

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
The qeth group device that is to take over an IP address must be enabled for IP takeover. For HiperSockets, both the device that takes over the address and the device that surrenders the address must be enabled. 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 each qeth group device can take over. You use the **qethconf** command to display or change this 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 address to be activated or deactivated. See "qethconf - Configure qeth devices" on page 305 for more details on the **qethconf** command.

Example: In this example, there is only one range of IP address 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 address that can be taken over by device eth0.

```
# qethconf ipa add 192.168.10.1/24 eth0
qethconf: Added 192.168.10.1/24 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 192.168.10.1/24 eth0
```

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

```
# qethconf ipa del 192.168.10.1/24 eth0
qethconf: Deleted 192.168.10.1/24 from /sys/class/net/eth0/device/ipa_takeover/del4.
qethconf: Use "qethconf ipa list" to check for the result
```

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 121).

- 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 122).

To complete taking over a specific IP address and remove it from the CHPID or LPAR that previously held it, issue an **ifconfig** or equivalent command.

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

```
# ifconfig hsi0 192.168.10.22
```

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 to set it to any free IP address and then to set it to the address to be taken over.

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:

```
# ifconfig hsi0 0.0.0.0
# ifconfig hsi0 192.168.10.22
```

Be aware of the information in "Confirming that an IP address has been set" on page 120 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 address handling for IPv4 with the layer2 option" on page 102) 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. See `http://www.sjdjweis.com/linux/proxyarp/` for more information on proxy ARP.

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 305).

Be aware of the information in "Confirming that an IP address has been set" on page 120 when working with proxy ARP.

Example
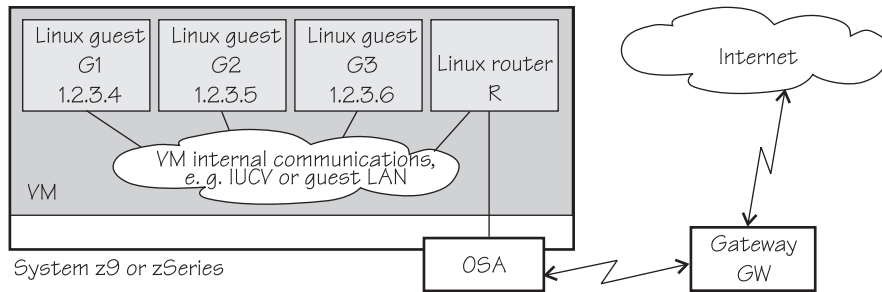Figure 24 on page 124 shows an environment where proxy ARP is used.

Figure 24. 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 address handling for IPv4 with the layer2 option" on page 102) 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.
- Virtual IP address (VIPA) can only be configured if the kernel has been compiled with the common code configuration option CONFIG_DUMMY.

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 305).

For an example of how to use VIPA, see "Scenario: VIPA – minimize outage due to adapter failure" on page 125.

Be aware of "Confirming that an IP address has been set" on page 120 when working with VIPAs.

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
```

Scenario: VIPA – minimize outage due to adapter failure

This chapter describes how to use
- Standard VIPA
- Source VIPA (versions 1.x)
- Source VIPA 2 (version 2.0.0)

VIPA allows you to 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 Server, 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. The VIPA functionality requires a kernel built with the CONFIG_DUMMY option.
2. See the information in "Confirming that an IP address has been set" on page 120 concerning possible failure when setting IP addresses for OSA-Express features in QDIO mode (qeth driver).
3. 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 25 on page 126).

- 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.

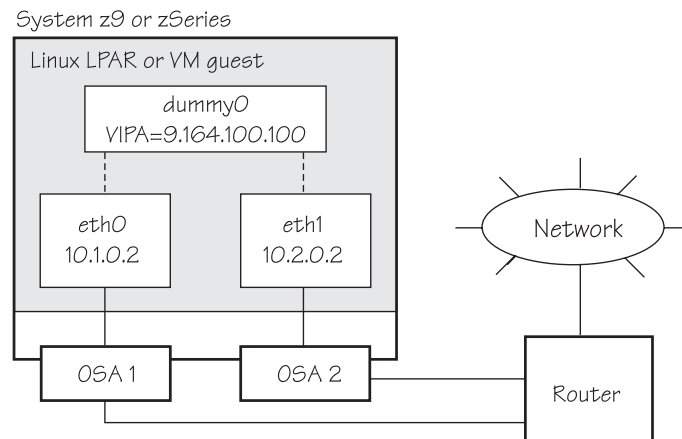Figure 25 shows the network adapter configuration used in the example.

Figure 25. Example of using Virtual IP Address (VIPA)

1. If the dummy component has not been compiled into the kernel, ensure that the dummy module has been loaded. If necessary, load it by issuing:

```
# modprobe dummy
```

2. Create a dummy interface with a virtual IP address, 9.164.100.100:

```
# ifconfig dummy0 9.164.100.100
```

3. Enable the network devices for this VIPA so that it accepts packets for this IP address.

```
 # 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
 # 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:

```
# qethconf vipa add 20020000000000000000000012345678 eth0
qethconf: Added 20020000000000000000000012355678 to /sys/class/net/eth0/device/vipa/add6.
qethconf: Use "qethconf vipa list" to check for the result
# qethconf vipa add 20020000000000000000000012355678 eth1
qethconf: Added 20020000000000000000000012355678 to /sys/class/net/eth1/device/vipa/add6.
qethconf: Use "qethconf vipa list" to check for the result
```

4. Ensure that the addresses have been set:

```
# qethconf vipa list
vipa add 9.164.100.100 eth0
vipa add 9.164.100.100 eth1
```

5. Ensure that your service (such as the Apache Web server) listens to the virtual IP address.

6. 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.

```
# 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.

```
# route delete -host 9.164.100.100
```

2. Create the alternative route to the virtual IP address.

```
# route add -host 9.164.100.100 gw 10.2.0.2
```

Source VIPA

This version of Source VIPA has been superseded in October 2003 by Source VIPA 2 (version 2.0.0). If you use Source VIPA 2, read "Source VIPA 2" on page 129.

Purpose

Source VIPA provides a functionality specially required in high-performance environments. It is a very flexible means for source address selection to arbitrary applications.

Normally, IP packets are tagged with the IP address of the adapter through which they leave the system. In case of an adapter failure, IP packets of outgoing sockets cannot be routed correctly and thus become ″lost″ in the network. With Source VIPA, they are tagged with the VIPA instead. This is done by dynamically linking the Source VIPA shared object file to the application to be Source-VIPA-enabled before linking the runtime library to it. This catches some socket calls and modifies them to yield a Source VIPA effect. The result is a per-application configuration of Source VIPA.

The Source VIPA solution does not affect kernel stability. Source VIPA is controlled by a configuration file (usually /etc/src_vipa.conf) containing flexible rules when to use Source VIPA, based on destination IP address ranges.

Note: This implementation of Source VIPA applies to IPv4 only.

An alternative but somewhat less flexible approach to Source VIPA is to use the `ip` tool to modify the routing table:

```
ip route your_route_statement src S1.S2.S3.S4
```

Usage

Installation:

```
make
make starter
make install
```

Paths can be changed in the Makefile. Defaults are:

```
SRC_VIPA_PATH=/lib
SRC_VIPA_STARTER_PATH=/usr/local/bin
```

The starter script should be in the execution path when you start the application.

Configuration: `/etc/src_vipa.conf`, or the file pointed to by environment variable SRC_VIPA_CONFIG_FILE, contains lines such as the following:

```
# comment
D1.D2.D3.D4/MASK S1.S2.S3.S4
.INADDR_ANY P1-P2 S1.S2.S3.S4
.INADDR_ANY P S1.S2.S3.S4
```

`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, src_vipa does a bind to S1.S2.S3.S4. Instead of IP addresses in dotted notation, host names can be used and will be resolved using DNS.

`.INADDR_ANY P1-P2 S1.S2.S3.S4`, or the same command with only one port P, will associate sockets with IP addresses causing 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 S1.S2.S3.S4 (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 `bind` to bind to a local port but with `INADDR_ANY` as a source address to use automatic source address selection.

The default behavior for all ports is that the kind of bind calls will not be modified.

Enabling an application: The command:

```
src_vipa.sh <application and parameters>
```

This enables the Source VIPA functionality for the application. The config file is read once at the start of the application. It is also possible to change the starter script and run multiple applications using different Source VIPA settings in separate files pointed to by a SRC_VIPA_CONFIG_FILE environment variable defined and exported prior to invoking the respective application.

Restrictions
LD_PRELOAD security prevents setuid executables to be run under src_vipa; programs of this kind can only be run when the real UID is 0.
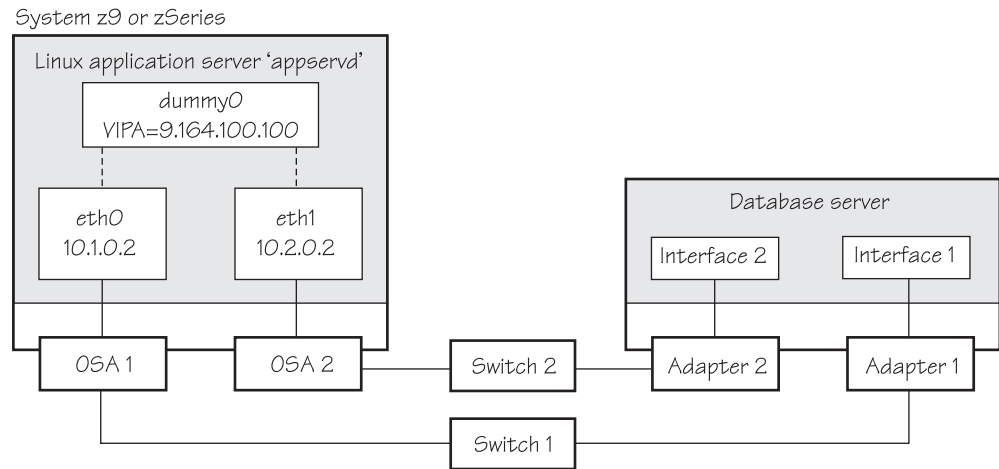
Example

Figure 26. Example of using source VIPA

The command:

```
src_vipa.sh appservd start
```

starts the application server with Source VIPA functionality. Packets leaving 'appservd' are tagged with the source address 9.164.100.100, regardless of the physical interface. In case of an outage, communication can still be maintained with the originating application as long as one of the physical interfaces is functioning. For example, if Switch 1 fails, Switch 2 can maintain the logical connection with 'appservd'.

Source VIPA 2

Purpose

Source VIPA 2 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 2 (/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:

```
make
make starter
make install
```

Paths can be changed in the Makefile. Defaults are:

```
SRC_VIPA_PATH=/lib
SRC_VIPA_STARTER_PATH=/usr/local/bin
```

The starter script should be in the execution path when you start the application.

Migration: If you migrate from an earlier version of Source VIPA and do not need multiple VIPAs, the *onevipa policy* followed by your VIPA is the recommended change (see "Policies"). Please check your syslog (usually in `/var/log/messages`) for problems the first time you use the new version.

Configuration: With Source VIPA 2 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 the *policies* section 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.

Restrictions

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.

The maximum number of VIPAs per destination is currently defined as 8.

Example
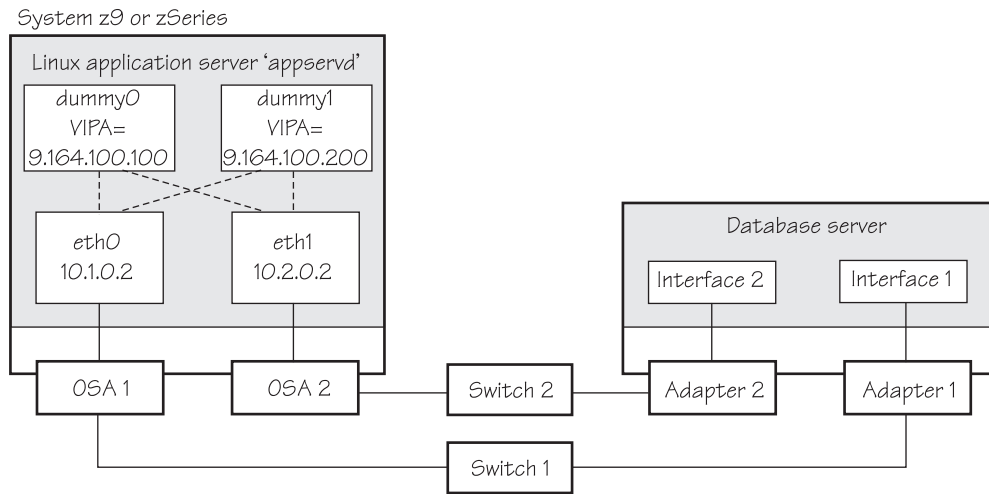
Figure 27. Example of using source VIPA 2

The entry in the Source VIPA configuration file:

```
9.0.0.0/8 lrr 9.164.100.100 9.164.100.200
```

sets up a Source VIPA 2 with a local round robin policy.

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.

Building a Linux kernel with VLAN and OSA-Express support is a prerequisite for using VLAN under Linux.

VLAN is not supported on Token Ring Interfaces of the OSA-Express adapter running in QDIO mode.

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.
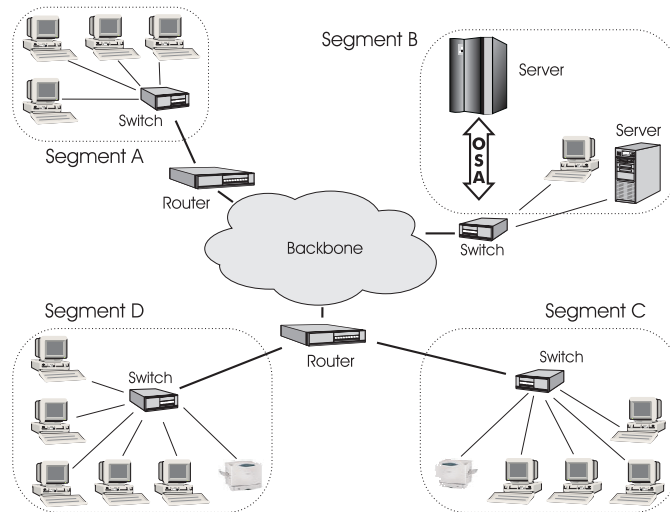
Figure 28. Conventional routed network

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.
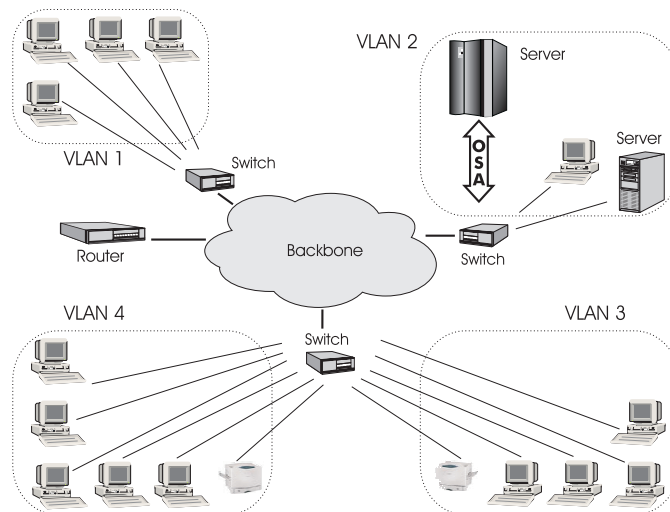
Figure 29. Switched VLAN network

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.
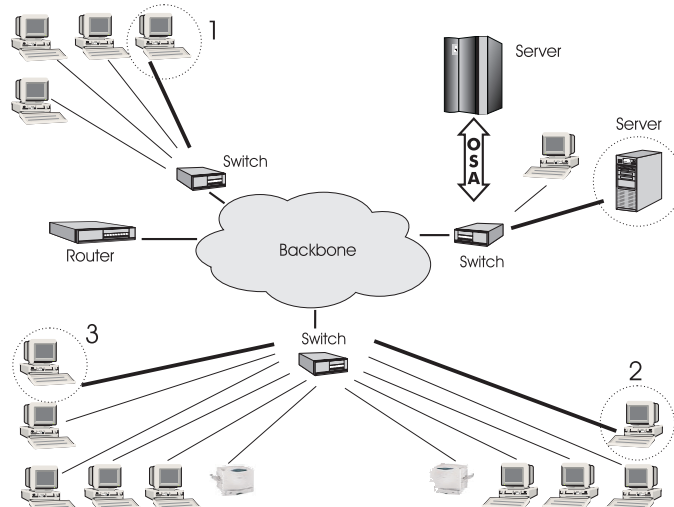
Figure 30. VLAN network organized for traffic flow

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, each connected to a physical Gigabit Ethernet LAN or 10 Gigabit Ethernet via 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: Intel)
    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
```

Further information

More information on VLAN for Linux is available at

```
http://scry.wanfear.com/~greear/vlan.html
```

HiperSockets Network Concentrator

This section describes how configure a HiperSockets Network Concentrator if the layer2 option (see "MAC address handling for IPv4 with the layer2 option" on page 102) is not enabled.

Before you start: This section applies to IPv4 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 zSeries Server environment, or systems connected by a different HiperSockets Network Concentrator into a zSeries 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 123) 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 on a zSeries Linux system 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 109).

  If only unicast packages are to be forwarded, there is also the possibility not to identify the OSA interface as multicast router: add the interface name to the `start_hsnc` script and only unicast packets will be forwarded.

- 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 either manually with the command

  ```
  sysctl -w net.ipv4.ip_forward=1
  ```

Alternatively, distribution-dependent configuration files can be used 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.

For further information about availability consult the general documentation of Linux on zSeries on availability.

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 most distributions 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.

Restrictions

- 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.
- For restrictions regarding multicast and broadcast forwarding, visit the IBM developerWorks Web site at:

  `ibm.com/developerworks/linux/linux390/perf/tuning_rec_networking.shtml`.
- To use HiperSockets Network Concentrator the kernel patches and s390-bit tools from the "June 2003 stream" on developerWorks as of 10/31/2003 are required.

Examples

Figure 31 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.
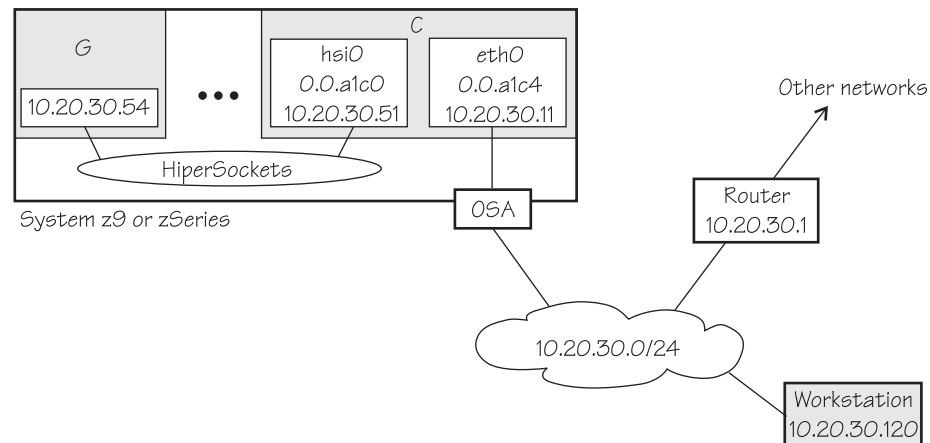
Figure 31. HiperSockets network concentrator setup

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
```

Tip: Refer to your distribution information on how to use configuration files to automatically enable IP forwarding when booting.

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 32 shows the example of Figure 31 on page 138 with an additional mainframe. On the second mainframe a Linux instance D acts as a HiperSockets network concentrator.
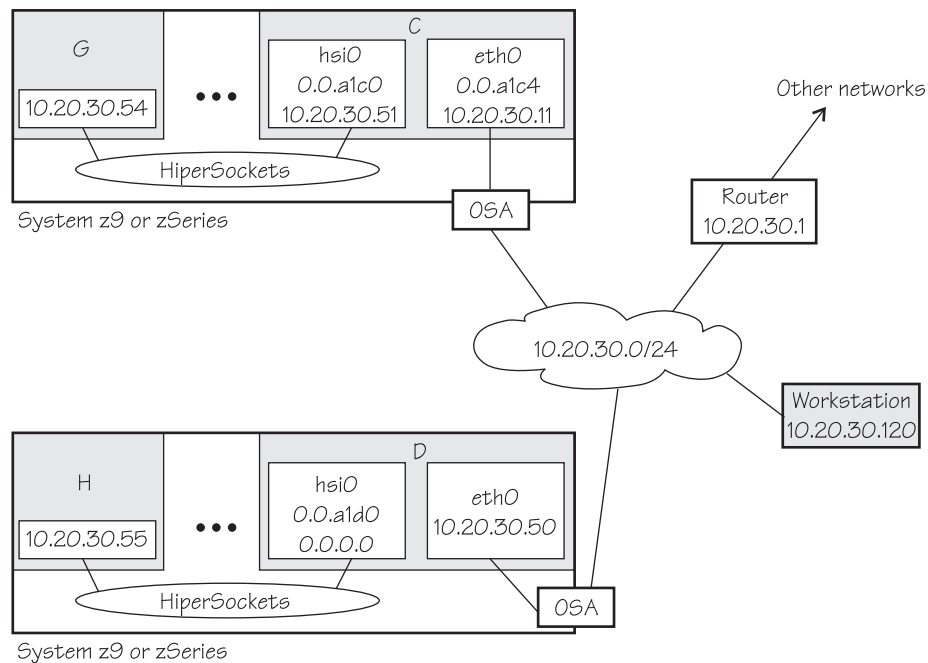
Figure 32. Expanded HiperSockets network concentrator setup

The configuration of C, G, and the workstation remain the same as for Figure 31 on page 138.

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: Refer to your distribution information on 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, LLC header, and MAC address handling (see "MAC address
handling for IPv4" on page 98). Because a HiperSockets connection does not go
out on a physical network, there are no ARP, LLC 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 on the Internet
at: `http://www.ietf.org/`.

Two types of DHCP environments have to be taken into account:
- DHCP via OSA-Express adapters in QDIO mode
- DHCP in a z/VM guest LAN

For information on setting up DHCP for Linux for zSeries 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: `ibm.com/redbooks/`.

As an example of DHCP client and server that you can use, this book discusses the
dhcpcd client and server *dhcp*; the distribution you use may provide different DHCP
client and server programs. There are three possibilities to get the DHCP client
dhcpcd and server *dhcp* working properly via OSA-Express adapters in QDIO
mode:
- Enabling the qeth *layer2* option (see "MAC address handling for IPv4 with the
  layer2 option" on page 102). This is the preferred method.
- Enabling the qeth *fake_ll* option (see "Faking LLC headers" on page 100) on both
  the client and the server side. This is the preferred method for environments that
  do not support the layer2 option.

Required options for using DHCP on Linux for System z

The following option are required if you are using `dhcpcd` on Linux for System z
without the layer2 option:
- You need to run `dhcpcd` with option **-B**.

  This option instructs the DHCP server to broadcast its response to the DHCP
  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 could
  not receive the response from the DHCP server without this option.

- You need to run `dhcpcd` with option **-I**.

   Specifies the client identifier string. On default `dhcpcd` uses the MAC address of the network interface as default. Hence, without this option, all Linux guests that share the same OSA-Express adapter in QDIO mode would also have the same client identifier.

There are no special options you need for using `dhcp` on Linux for System z.

Setting up for tcpdump with IPv4

For connections through an OSA-Express adapter in QDIO mode, the OSA-Express adapter off-loads ARP, LLC header, and MAC address handling (see "MAC address handling for IPv4" on page 98). Because a HiperSockets connection does not go out on a physical network, there are no ARP, LLC headers, and MAC addresses for packets in a HiperSockets LAN. The resulting problems for tcpdump are the same in both cases and the fixes for connections through the OSA-Express adapter also apply to HiperSockets.

tcpdump uses the packet capture library libpcap. The libpcap library provides a high level interface to packet capture systems. All packets on the network, even those destined for other hosts, are accessible through this mechanism.

libpcap requires an Ethernet LLC header for the packets it captures. To make it work properly in a Linux on System z environment you can do either:
- Enable the qeth *layer2* option (for OSA Express only, see "MAC address handling for IPv4 with the layer2 option" on page 102). This is the preferred method.
- Make some changes in the libpcap library.

You need to write your own patch that is suited to your environment. The following sample patch is for illustration purposes only and not intended for you to use directly. The sample patch for libpcap version 0.6.2. illustrates how you can do this:

```
┌─ libpcap 0.6.2. sample patch ─────────────────────────────────────────────┐
│                                                                            │
│   --- libpcap-0.6.2/pcap-linux.c Fri Jan 31 17:24:51 2003                   │
│   +++ libpcap-0.6.2/pcap-linux.c.s390qdio Fri Jan 31 17:23:31 2003          │
│   @@ -310,6 +310,35 @@                                                      │
│      return -1;                                                             │
│      }                                                                      │
│     }                                                                       │
│   + /* IBM OSA-Express modifications                                        │
│   +  */                                                                     │
│   +#define IBM_SRC_MAC   "IBMOSA"                                           │
│   +#define IBM_DST_MAC   "eWorld"                                           │
│   +        do {                                                             │
│   +         unsigned short enc_proto;                                       │
│   +         unsigned short proto = 0;                                       │
│   +         enc_proto = *((char*)handle->buffer +                           │
│   +                      sizeof(unsigned short));                           │
│   +         if ( (enc_proto == ETH_P_IP) ||                                 │
│   +             (enc_proto == ETH_P_IPV6) )                                 │
│   +           proto = ETH_P_8021Q;                                          │
│   +         else  if ( (*((char*)handle->buffer) >= 0x45) &&                │
│   +        (*((char*)handle->buffer) <= 0x4f))                              │
│   +            proto = ETH_P_IP;                                            │
│   +         else  if ( *((char*)handle->buffer) == 0x60 )                   │
│   +            proto = ETH_P_IPV6;                                          │
│   +   if (proto) {                                                          │
│   +         memmove( ((char*)handle->buffer+sizeof(struct ethhdr)),         │
│   +          handle->buffer,packet_len);                                    │
│   +         packet_len += 14;                                               │
│   +         struct ethhdr *hdr = (struct ethhdr *)handle->buffer;           │
│   +         memcpy(hdr->h_dest,IBM_DST_MAC,ETH_ALEN);                       │
│   +         memcpy(hdr->h_source,IBM_SRC_MAC,ETH_ALEN);                     │
│   +         hdr->h_proto = proto;                                           │
│   +    }                                                                    │
│   +        } while(0);                                                      │
│   +#undef IBM_SRC_MAC                                                       │
│   +#undef IBM_DST_MAC                                                       │
│                                                                            │
│    #ifdef HAVE_PF_PACKET_SOCKETS                                            │
│     /*                                                                      │
│   @@ -552,7 +581,8 @@                                                       │
│      /*                                                                     │
│       * We have a filter that'll work in the kernel.                        │
│       */                                                                    │
│   -   can_filter_in_kernel = 1;                                            │
│   +/*IBM QDIO device have to filter in the user land*/                      │
│   +   can_filter_in_kernel = 0;                                            │
│      break;                                                                 │
│     }                                                                       │
│    }                                                                        │
│                                                                            │
└────────────────────────────────────────────────────────────────────────────┘
```

The first part of the patch adds a fake LLC header to all network packets that do not have one. The second part prevents filtering of network packets in the kernel so that packets are filtered in user mode after a fake LLC header has been added. There is no impact on other network device types, like LCS devices.

Setting up a Linux guest as a z/VM guest LAN sniffer

This topic describes how you can set up a Linux instance to act as a guest LAN sniffer, for example, to make data on guest LAN traffic available to tools like TCPDUMP or ETHEREAL. The guest LAN sniffer can be used on a guest that is defined through a z/VM virtual switch. If the switch connects to a VLAN that includes nodes outside z/VM, these external nodes are beyond the scope of the sniffer.

For general information on VLAN and z/VM virtual switches, refer to Redpaper *Linux on IBM @server zSeries and S/390: VSWITCH and VLAN Features of z/VM 4.4*, REDP-3719 at: ibm.com/redbooks/.

Before you start:

- You need z/VM 5.2 or later.
- You need class B authorization on z/VM.
- The guest LAN must be defined through a virtual switch.
- The Linux guest to be set up as a guest LAN sniffer must run as a guest of the same z/VM as the guest LAN you want to investigate.

Linux setup:

Ensure that the qeth device driver has been compiled into the Linux kernel or that the qeth device driver has been loaded as a module.

z/VM setup:

Ensure that the Linux guest is authorized for the switch and for promiscuous mode.

Perform the following steps from a CMS session on your z/VM:

1. Check if the Linux guest already has the required authorizations; issue the following CP command:

   ```
   q vswitch <switchname> promisc
   ```

   where *<switchname>* is the name of the virtual switch. If the output lists the Linux guest as authorized for promiscuous mode, no further setup is required.

2. If the output from step 1 does not list the Linux guest, check if the guest is authorized for the virtual switch; issue the following CP command:

   ```
   q vswitch <switchname> acc
   ```

   where *<switchname>* is the name of the virtual switch.

   If the output lists the Linux guest as authorized, you must temporarily revoke the authorization for the switch before you can grant authorization for promiscuous mode. Issue a CP command of this form:

   ```
   set vswitch <switchname> revoke <userid>
   ```

   where *<switchname>* is the name of the virtual switch and *<userid>* is the z/VM user ID of the Linux guest.

3. Authorize the Linux guest for the switch and for promiscuous mode; issue a CP command of this form:

   ```
   set vswitch <switchname> grant <userid> promisc
   ```

   where *<switchname>* is the name of the virtual switch and *<userid>* is the z/VM user ID of the Linux guest.

For details on the CP commands used in this section, refer to *z/VM CP Commands and Utilities Reference*, SC24-6081 at the IBM Publications Center (see "Finding IBM books" on page xxi).

Chapter 9. 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:

- OSA-2 Ethernet/Token Ring
- OSA-Express
  - Fast Ethernet
  - 1000Base-T Ethernet (z890 and z990)
  - Token Ring
- OSA-Express2
  - 1000Base-T Ethernet (System z9 only)

Features

The LCS device driver supports the following devices and functions:

- Auto detects whether the CHPID is connected to Token Ring or Ethernet
- 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.
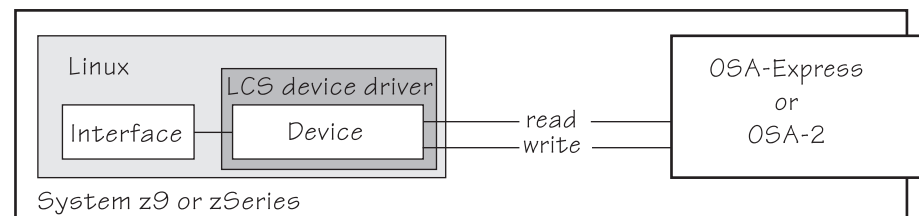
Figure 33. I/O subchannel interface

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. According to the feature used, the naming scheme uses two base names:

eth<n>   for Ethernet features

tr<n>   for Token Ring 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. When the first Token Ring feature is set online, it is assigned the interface name "tr0".

The LCS device driver shares the name space for Ethernet and Token Ring 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 97.

Building a kernel with the LCS device driver

This section is intended for those who want to build their own kernel. It describes the options you must select in the Linux configuration menu to include the LCS device driver.

You need to select the option CONFIG_LCS if you want to work with LCS devices.

```
Networking support
 └─S/390 network device drivers
    └─Lan Channel Station Interface                    (CONFIG_LCS)
```

Figure 34. LCS kernel configuration menu option

The CONFIG_LCS option can be compiled into the kernel or as a separate module, lcs.

Depending on the features you intend to support, you need to include at least one the common code options CONFIG_TR and CONFIG_NET_ETHERNET. For multicast support you also require the common code option CONFIG_IP_MULTICAST.

Setting up the LCS device driver

There are no kernel or module parameters for the LCS device driver.

If you have compiled the LCS component as a separate module, you need to load it 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
- Specifying a timeout for LCS LAN commands
- Setting a device online or offline
- Activating and deactivating an interface

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`

Specifying a timeout for LCS LAN commands

Before you start: The LCS group device must be offline while you specify the timeout.

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 tr0 without IPv6 support
 lcs: LCS device tr0 with Multicast support
...
```

The interface name that has been assigned to the LCS group device in the example is tr0. To confirm that this is the correct name for our group device issue:

```
# readlink /sys/class/net/tr0/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
```

Chapter 10. CTCMPC device driver

The CTCMPC device driver is required by Communications Server for Linux to provide Channel-to-Channel (CTC) Multi-Path Channel (MPC) connections. Through CTCMPC connections, Linux can be a communication peer for VTAM on traditional mainframe operating systems.

This section describes how to set up the CTCMPC device driver. Visit `ibm.com/software/network/commserver/linux/` for more information on Communications Server for Linux and on using CTCMPC connections.

Features

The CTCMPC device driver allows Communications Server for Linux to provide:
- ESCON CTC connections (standard CTC and basic CTC) between mainframes in basic mode, LPARs or VM guests.
- Virtual CTC/A connections between VM guests of the same VM system.
- Connections to VTAM on traditional mainframe operating systems.

What you should know about CTCMPC

This section provides information on CTCMPC interfaces.

CTCMPC group devices

The CTCMPC device driver requires two I/O subchannels for each interface, a read subchannel and a write subchannel. The device bus-IDs that correspond to the two subchannels must be configured for control unit type 3088.
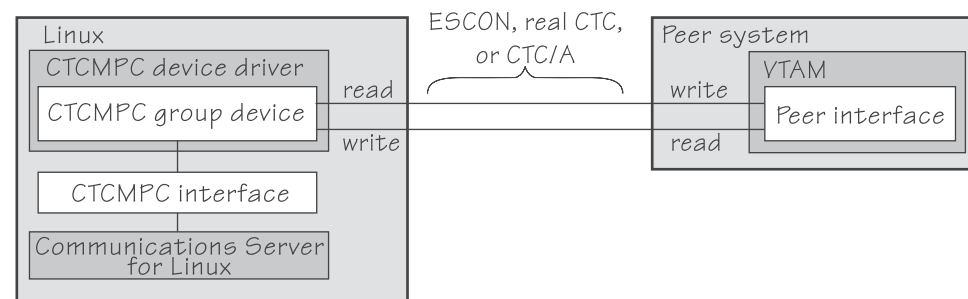
Figure 35. I/O subchannel interface

The device bus-IDs that correspond to the subchannel pair are grouped as one CTCMPC group device.

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.

CTCMPC interfaces

When a CTCMPC group device is set online, the CTCMPC device driver automatically assigns an interface name to it. The interface names are of the form mpc*<n>* where *<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.

Building a kernel with the CTCMPC device driver

This section is intended for those who want to build their own kernel.

You need to select the kernel configuration option CONFIG_MPC to be able to use CTCMPC connections.

```
Networking support
   S/390 network device support
   └─CTCMPC device support                            (CONFIG_MPC)
```

Figure 36. CTCMPC kernel configuration menu option

The CTCMPC device driver can be compiled into the kernel or as a separate module, ctcmpc.

Setting up the CTCMPC device driver

You do not need to specify kernel or module parameters for the CTCMPC device driver. If the CTCMPC device driver has been compiled as a separate module, load it with the **modprobe** command to ensure that any other required modules are loaded:

```
# modprobe ctcmpc
```

Working with the CTCMPC device driver

This section describes typical tasks that you need to perform when working with CTCMPC devices.
- Creating a CTCMPC group device
- Setting a device online or offline

Refer to the Communications Server for Linux documentation for information on how to configure and activate CTCMPC interfaces.

Creating a CTCMPC group device

Before you start: You need to know the device bus-IDs that correspond to the local read and write subchannel of your CTCMPC connection as defined in your IOCDS.

To define a CTCMPC group device, write the device bus-IDs of the subchannel pair to /sys/bus/ccwgroup/drivers/ctcmpc/group. Issue a command of this form:

```
# echo <read_device_bus_id>,<write_device_bus_id> > /sys/bus/ccwgroup/drivers/ctcmpc/group
```

Result: The CTCMPC device driver uses the device bus-ID of the read subchannel to create a directory for a group device:

/sys/bus/ccwgroup/drivers/ctcmpc/<read_device_bus_id>

This directory contains a number of attributes that determine the settings of the CTCMPC 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/ctcmpc/group
```

This command results in the creation of the following directories in sysfs:
* /sys/bus/ccwgroup/drivers/ctcmpc/0.0.2000
* /sys/bus/ccwgroup/devices/0.0.2000
* /sys/devices/ctcmpc/0.0.2000

Setting a device online or offline

To set a CTCMPC group device online, set the online device group attribute to "1".
To set a CTCMPC group device offline, set the online device group attribute to "0".
Issue a command of this form:

```
# echo <flag> > /sys/bus/ccwgroup/drivers/ctcmpc/<device_bus_id>/online
```

Setting a device online associates it with an interface name. Setting the device offline preserves the association with 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 access the CTCMPC 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 CTCMPC device with bus ID 0.0.2000 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcmpc/0.0.2000/online
# dmesg
...
mpc0: read: ch-0.0.2000, write: ch-0.0.2001, proto: 4
...
```

The interface name that has been assigned to the CTCMPC group device in the example is mpc0. To confirm that this is the correct name for our group device issue:

```
# readlink /sys/class/net/ctcmpc0/device
../../../devices/cu3088/0.0.2000
```

To set the same device offline issue:

```
# echo 0 > /sys/bus/ccwgroup/drivers/mpc/0.0.2000/online
```

Part 4. z/VM virtual server integration

This section describes device drivers and features that help to effectively run and manage a z/VM-based virtual Linux server farm.

- Chapter 11, "z/VM DCSS device driver"
- Chapter 12, "z/VM *MONITOR record reader device driver"
- Chapter 13, "Linux monitor stream support for z/VM"
- Chapter 14, "Monitor stream application support," on page 181
- Chapter 15, "z/VM recording device driver"
- Chapter 16, "Watchdog device driver"
- Chapter 17, "z/VM CP interface device driver"
- Chapter 18, "Cooperative memory management"

> **Note**
>
> For prerequisites and restrictions for these device drivers and features, refer to the kernel 2.6 October 2005 stream pages on developerWorks at:
>
> `ibm.com/developerworks/linux/linux390/october2005_recommended.html`

Chapter 11. 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 xip2 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.

What you should know about DCSS

This section provides information on 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

When the DCSS device driver is loaded, it dynamically allocates a major number to DCSS devices. A different major number might be used when the device driver is reloaded, for example when Linux is rebooted. Check the entry for "dcssblk" in `/proc/devices` to find out which major number is used for your DCSSs.

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 is removed, its device name and corresponding minor number are free and can be reassigned. A DCSS that is added always receives the lowest free minor number.

Creating device nodes

User space programs access DCSS devices by *device nodes*. Your distribution might create these device nodes for you or provide udev to create them (see "Device nodes provided by udev" on page 4).

If no device nodes are created for you, you need to create them yourself, for example, with the **mknod** command. Refer to the **mknod** man page for further details.

Tip: Use the device names to construct your nodes (see "DCSS naming scheme").

Example: Defining standard DCSS nodes

To create standard DCSS device nodes of the form `/dev/<device_name>` issue commands of this form:

```
# mknod /dev/dcssblk0 b <major> 0
# mknod /dev/dcssblk1 b <major> 1
# mknod /dev/dcssblk2 b <major> 2
...
```

Further information

- For information on DCSS see *z/VM Saved Segments Planning and Administration*, SC24-6056
- For related z/VM information see *CP Command and Utility Reference*, SC24-6008.
- For an example of how the xip2 file system and DCSS can be used see *How to use Execute-in-Place Technology with Linux on z/VM* on developerWorks at:

  ibm.com/developerworks/linux/linux390/october2005_documentation.html

Building a kernel with the DCSS device driver

This section is intended for those who want to build their own kernel.

To build a kernel with DCSS support you need to select option CONFIG_DCSSBLK in the configuration menu:

```
Block devices
    DCSSBLK support                              (CONFIG_DCSSBLK)
```

Figure 37. DCSS kernel configuration menu option

The DCSS support is available as a module, dcssblk, or built-in.

Setting up the DCSS device driver

Kernel parameters

This section describes how to configure the DCSS device driver if the DCSS block device support has been compiled into the kernel. You configure the device driver by adding parameters to the kernel parameter line.

Use the dcssblk.segments kernel parameter to load one or more DCSSs during the boot process (for example, for use as swap devices).
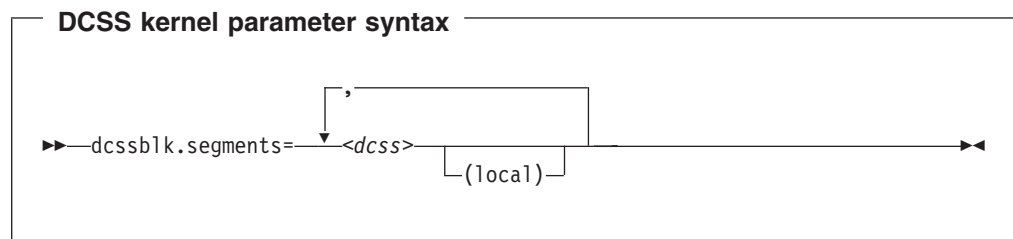
DCSS kernel parameter syntax

```
>>--dcssblk.segments=--+--<dcss>--+----------------------------------><
                       |          |
                       +--(local)-+
```

where *<dcss>* specifies the name of a DCSS and **(local)** sets the access mode to exclusive-writable after the DCSS has been loaded. The specification for *<dcss>* is converted from ASCII to uppercase EBCDIC.

Example
The following parameter in the kernel parameter line loads three DCSSs during the boot process: DCSS1, DCSS2, and DCSS3. DCSS2 is accessed in exclusive-writable mode and can be included in `/etc/fstab` and used as a swap device.

```
dcssblk.segments=dcss1,dcss2(local),dcss3
```

Module parameters

This section describes how to load and configure the DCSS device driver if the DCSS block device support has been compiled as a separate module.

Load the DCSS block device driver with **modprobe** or **insmod**. Use the segments module parameter to load one or more DCSSs when the DCSS device driver is loaded.
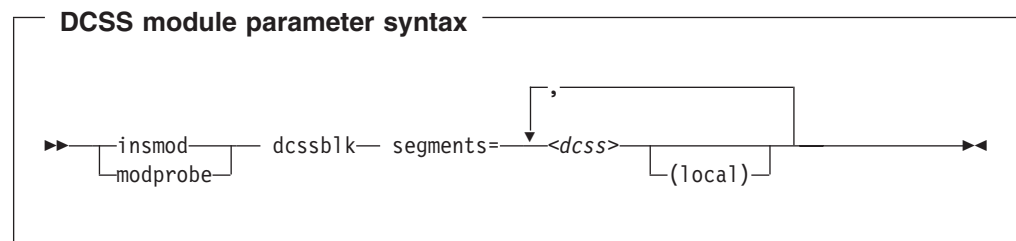
DCSS module parameter syntax

```
>>--┬--insmod---┬--dcssblk--segments=--┬-<dcss>----------┬--><
    └-modprobe--┘                      │      ┌--,-----┐ │
                                       └------<dcss>----┘
                                           └-(local)-┘
```

where *<dcss>* specifies the name of a DCSS and **(local)** sets the access mode to exclusive-writable after the DCSS has been loaded. The specification for *<dcss>* is converted from ASCII to uppercase EBCDIC.

Example
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
```

Working with the DCSS device driver

This section describes typical tasks that you need to perform when working with DCSS devices:
* Adding a DCSS
* Finding the minor number for a DCSS
* Setting the access mode
* Saving an updated DCSS
* Removing a DCSS

Adding a DCSS

Before you start:
* You need to have set up a DCSS on z/VM and know the name assigned to the DCSS on z/VM.

- You must have set the mem kernel parameter to cover the upper limit of the DCSS.
- If you use the watchdog device driver, turn off the watchdog before adding or saving a DCSS. Adding or saving a DCSS may result in a watchdog timeout, if it is active.

Restrictions:
- You cannot concurrently access overlapping DCSSs.
- You cannot access a DCSS that overlaps with your guest virtual storage.

To add a DCSS device write the name of the DCSS to /sys/devices/dcssblk/add. Issue a command of this form:

```
# echo <name> > /sys/devices/dcssblk/add
```

where name is the name of the DCSS as defined to z/VM.

Example
To add a DCSS called "MYDCSS" issue:

```
# echo MYDCSS > /sys/devices/dcssblk/add
```

Finding the minor number for a DCSS

When you add a DCSS, 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 157).

When you add a DCSS, a directory of this form is created in sysfs:

- /sys/devices/dcssblk/<name>

where <name> is the name of the DCSS.

This directory contains a symbolic link, block, that helps you to find out the standard 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 "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 with write access to change the DCSS content. There are two possible write access modes to the DCSS:

shared
  In the shared mode, changes to the DCSS are immediately visible to all guests that access the DCSS. Shared is the default.

Note: Writing to a shared DCSS bears the same risks as writing to a shared disk.

exclusive-writable

In the exclusive-writable mode you write to a private copy of the DCSS. The private copy is writable, even if the original DCSS is read-only. The changes in your private copy are invisible to other guests until you save the changes (see "Saving an updated DCSS").

After saving the changes 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.

For either access mode the changes are volatile until they are saved (see "Saving an updated DCSS").

Set the access mode before you mount the DCSS. 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/<name>/shared
```

where *<name>* is the name of the DCSS.

You can read the shared attribute to find out the current access mode.

Example

To find out the current access mode of a DCSS "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 an updated DCSS

If you use the watchdog device driver, turn off the watchdog before adding or saving a DCSS. Adding or saving a DCSS may result in a watchdog timeout, if it is active.

Do not place save requests before you have accessed the DCSS. To place a save request for saving changes to a DCSS permanently on the spool disk write "1" to the DCSS device's save attribute.

Issue a command of this form:

```
# echo 1 > /sys/devices/dcssblk/<name>/save
```

where *<name>* is the name of the DCSS.

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 "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
```

Removing a DCSS

Before you start: A DCSS device can only be removed when it is not in use.

To remove a DCSS device write the name of the DCSS to `/sys/devices/dcssblk/remove`. Issue a command of this form:

```
# echo <name> > /sys/devices/dcssblk/remove
```

where *<name>* is the name of the DCSS.

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 "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:
- Our Linux instance runs as a VM guest with class E user privileges.
- A DCSS has been set up and made accessible to our Linux guest.
- The DCSS does not overlap with our guest's main storage and the mem parameter has been set to cover the DCSS's upper limit.
- There is only a single DCSS named "MYDCSS".

- The DCSS block device driver has been set up and is ready to be used.

Perform the following steps to change the contents of a DCSS:

1. Access the DCSS by adding it 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 if our example is 254.

   - Find out the minor number used for MYDCSS. If MYDCSS is the first DCSS that has been added the minor 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 1 > /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 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 12. z/VM *MONITOR record reader device driver

The z/VM *MONITOR record reader device driver gives monitoring software on Linux access to z/VM guest data.

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.

Features

The z/VM *MONITOR record device driver supports the following devices and functions:

- Read access to the z/VM *MONITOR DCSS.
- Reading *MONITOR records for z/VM 4.4 and later.
- Access to *MONITOR records as described on: `ibm.com/vm/pubs/ctlblk.html`
- Access to the records provided by the Linux monitor stream (see Chapter 13, "Linux monitor stream support for z/VM," on page 173).

What you should know about the z/VM *MONITOR record device driver

The data that is collected by *MONITOR depends on how you have set up the service. The z/VM *MONITOR record 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 "Making the DCSS addressable for your Linux guest" on page 166 for information on how to check if MONDCSS exists.

Further information

- Refer to *Saved Segments Planning and Administration*, SC24-6116 for general information on DCSSs.
- Refer to *z/VM Performance*, SC24-6109 for information on how to create a monitor DCSS.
- Refer to *CP Command and Utility Reference*, SC24-6081 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 visit `ibm.com/vm/pubs/mon440/index.html` and refer to Chapter 13, "Linux monitor stream support for z/VM," on page 173.

Building a kernel with the z/VM *MONITOR record device driver

This section is intended for those who want to build their own kernel.

You need to select the kernel configuration option CONFIG_MONREADER to be able to access the z/VM monitor DCSS.

```
Character device drivers --->
    API for reading z/VM monitor service records      (CONFIG_MONREADER)
```

*Figure 38. z/VM *MONITOR record kernel configuration menu options*

The z/VM *MONITOR record reader device driver can be compiled into the kernel or as a separate module, monreader.

You also need IUCV support.

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 165 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 DIRECT 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
```

Making the DCSS addressable for your Linux guest

You need to know the start and end address of the DCSS. You can find out this information 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
...
```

Your guest storage must not overlap with the address range of the DCSS. Depending on the start and end address of your DCSS you can do either:

- Define the guest storage as two or more discontiguous storage extents such that a storage gap covers the entire DCSS address range
- Enable Linux to handle real memory addresses that are beyond the guest storage (considered as real memory by Linux) to cover a DCSS that is located above the guest storage

Defining the guest storage with storage gaps

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 your entire DCSS. Issue a command of this form:

```
DEF STOR CONFIG 0.<storage_gap_beginn> <storage_gap_end>.<storage above gap>
```

where:

<storage_gap_begin>
   is the lower limit of the storage gap. The lower limit must be at least 64 MB and at or below the lower limit of the DCSS.

<storage_gap_end>
   is the upper limit of the storage gap. The upper limit must be above the upper limit of the DCSS.

<storage above gap>
   is the amount of storage above the storage gap. The total guest storage is *<storage_gap_beginn>* + *<storage above gap>*.

All values can be suffixed with M to provide the values in megabyte. Refer to *CP Command and Utility Reference*, SC24-6081 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 256 MB guest storage:

```
DEF STORE CONFIG 0.140M 160M.116M
```

The storage gap in the example ranges from 140 MB to 160 MB and thus covers the entire DCSS range. The total guest storage is 140 MB + 116 MB = 256 MB.

Extending the Linux address range

If your entire DCSS address range is above the guest storage you can use the mem= Linux kernel parameter to make the DCSS accessible to the Linux guest. Add the following to the kernel parameter line:

```
mem=<address>
```

where *<address>* is an address at or above the upper limit of the DCSS. Upper can be in kilobyte or megabyte and must be suffixed with K or M, accordingly.

Example: To make a DCSS that starts at 144 MB and ends at 152 MB accessible to a z/VM guest with 128 MB guest storage:

```
mem=160M
```

For a guest with guest storage 256 MB you would not be able to use this method because then the guest storage would overlap with the DCSS.

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. Proceed according to your distribution:

- If your device driver has been compiled into the kernel, specify the DCSS name as a kernel parameter.
- If your device driver has been compiled as a separate module, specify the DCSS name as a module parameter when you load the module.

Kernel parameter

This section describes how you can specify a DCSS name if the z/VM *MONITOR record reader device driver has been compiled into the kernel.

You can specify a DCSS name by adding the mondcss parameter to the kernel parameter line.

```
┌─ z/VM *MONITOR record reader device driver kernel parameter syntax ─┐
│                                                                      │
│              ┌─monreader.mondcss=MONDCSS─┐                           │
│   ►►─────────┤                           ├──────────────────────►◄   │
│              └─monreader.mondcss=<dcss>──┘                           │
│                                                                      │
└──────────────────────────────────────────────────────────────────┘
```
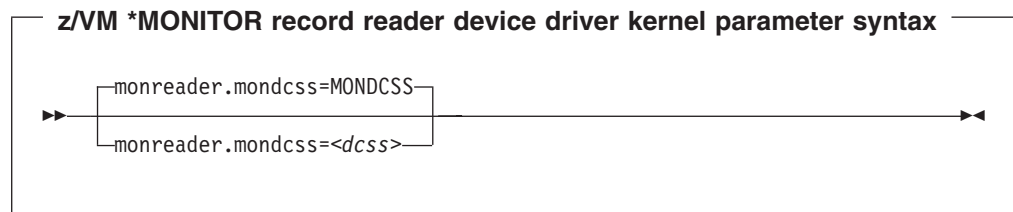
where *<dcss>* is the name of the DCSS that z/VM uses for the monitor records.

Example: To specify MYDCSS as the DCSS name add the following parameter to the kernel parameter line:

```
mondcss=MYDCSS
```

The value is automatically converted to upper case.

Module parameter

This section describes how to load the z/VM *MONITOR record reader device driver if it has been compiled as separate module. It also tells you how to specify a DCSS name, if applicable.

Load the z/VM *MONITOR record reader device driver module with **modprobe** to assure that any other required modules are also loaded. You need IUCV support if you want to use the z/VM *MONITOR record reader device driver.

```
 ┌─ z/VM *MONITOR record reader device driver 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 z/VM *MONITOR record device driver module and specify MYDCSS as the DCSS issue:

```
modprobe monreader mondcss=mydcss
```

Assuring that the required device node exists

You need a device node for a miscellaneous character device to access the monitor DCSS. Your distribution might create this device node for you (for example, by using udev). To find out if there is already a device node issue:

```
# find / -name monreader
```

If there is no device node, you need to create one. To find out the major and minor number for your monitor device read the dev attribute of the device's representation in sysfs:

```
# cat /sys/class/misc/monreader/dev
```

The value of the dev attribute is of the form *<major>*:*<minor>*.

To create issue a command of the form:

```
# mknod <node> c <major> <minor>
```

where *<node>* is your device node.

Example:
To create a device node /dev/monreader:

```
# cat /sys/class/misc/monreader/dev
10:63
# mknod /dev/monreader c 10 63
```

In the example, the major number was 10 and the minor 63.

Working with the z/VM *MONITOR record device driver

This section describes how to work with the z/VM *MONITOR record reader device driver.
- 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. Refer to *z/VM Performance* for details on 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* for a description of the monitor control element layout. The layout of the monitor records can be found on: `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 "Making the DCSS addressable for your Linux guest" on page 166).

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 13. Linux monitor stream support for z/VM

z/VM is a convenient point for collecting VM guest performance data and statistics for an entire server farm. Linux guests 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 monitor stream 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 176).

Building a kernel that is enabled for monitoring

This section is intended for those who want to build their own kernel.

Figure 39 summarizes the kernel configuration menu options that are relevant to the Linux monitor stream support for z/VM:

```
Base setup
   Virtual CPU timer support                          (CONFIG_VIRT_TIMER)
    └─Linux - VM Monitor Stream, base infrastructure (CONFIG_APPLDATA_BASE)
      ├─Monitor memory management statistics          (CONFIG_APPLDATA_MEM)
      ├─Monitor OS statistics                         (CONFIG_APPLDATA_OS)
      └─Monitor overall network statistics            (CONFIG_APPLDATA_NET_SUM)
```

Figure 39. Linux monitor stream kernel configuration menu options

CONFIG_VIRT_TIMER
> This option is a prerequisite for the Linux monitor stream support.

CONFIG_APPLDATA_BASE
> This option provides the base component for the Linux monitor stream support.

CONFIG_APPLDATA_MEM
> This option provides monitoring for memory related data. It can be compiled into the kernel or as a separate module, appldata_mem.

CONFIG_APPLDATA_OS
> This option provides monitoring for operating system related data, for example, CPU usage. It can be compiled into the kernel or as a separate module, appldata_os.

CONFIG_APPLDATA_NET_SUM
> This option provides monitoring for network related data. It can be compiled into the kernel or as a separate module, appldata_net_sum.

Setting up the monitor stream support

There are no kernel or 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 VM guest for the monitor stream support.

Loading data gathering modules

One or more of the data gathering components might have been compiled as separate modules. Use the **insmod** or **modprobe** command to load any required modules. Refer to the respective man pages command details.

```
  Monitor stream support module parameter syntax

►►──┬─insmod───┬──┬─appldata_mem─────┬──────────────────►◄
    └─modprobe─┘  ├─appldata_os──────┤
                  └─appldata_net_sum─┘
```
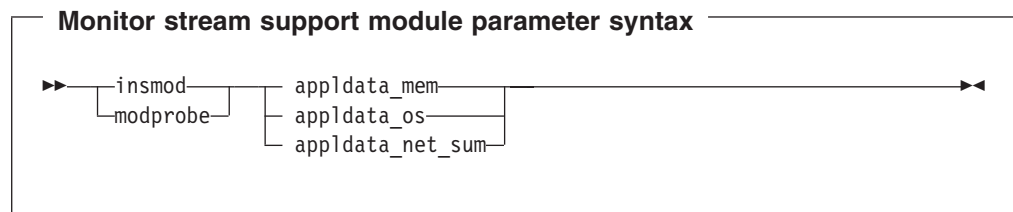
where appldata_mem, appldata_os, and appldata_net_sum are the modules for gathering memory related data, operating system related data, and network related data.

Enabling your VM guest for data gathering

To enable you Linux guest for data gathering ensure that the Linux guest directory includes the option APPLMON.

Working with the monitor stream 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 on or off the monitoring support

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 on or off the monitoring support" on page 174).

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 21 on page 177)
*   Processor data (see Table 22 on page 178)
*   Networking (see Table 23 on page 179)

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

> **Important**
>
> On 31-bit Linux systems, the u64 values are actually only 32-bit values. That is, the lower 32 bit wrap around like 32-bit counters and the upper 32 bit are always zero.

Table 21. 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 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 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, currently set to 0 by Linux kernel (2.4 and 2.6)
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 22. 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 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 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 23. 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 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 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

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 176 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. Monitor stream application support

The monitor stream application support provides a programmatic interface to allow applications to easily write application-specific monitor data (APPLDATA) to the z/VM monitor stream. This character device enables writing of z/VM monitor APPLDATA records.

Features

The monitor stream application support 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 monitor stream application support

The monitor stream application support 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 *Saved Segments Planning and Administration*, SC24-6116 for general information on DCSSs.
- Refer to *z/VM CP Programming Services*, SC24-6084 for information on the DIAG x'DC' instruction.
- Refer to *CP Command and Utility Reference*, SC24-6081 for information on the CP commands.
- Refer to *z/VM Performance*, SC24-6109 for information on monitor APPLDATA.

Building a kernel with monitor stream application support

This section is intended for those who want to build their own kernel.

To build a kernel with monitor stream application support you need to select option CONFIG_MONWRITER in the configuration menu:

```
Character device drivers
   API for writing z/VM monitor service records (CONFIG_MONWRITER)
```

The monitor stream application support can be compiled into the kernel or as a separate module, MONWRITER.

Setting up the monitor stream application support

This section describes the parameters that you can use to configure the monitor stream application support.

Kernel parameters

This section describes how to configure the monitor stream application support if it has been compiled into the kernel. You configure the device driver by adding parameters to the kernel parameter line.

```
┌─ Monitor stream application support kernel parameter syntax ──────────────┐
│                                                                          │
│                             ┌─255─────┐                                   │
│   ▶▶──monwriter.max_bufs=───┤         ├──────────────────────────────▶◀  │
│                             └─<NUMBUFS>─┘                                 │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```
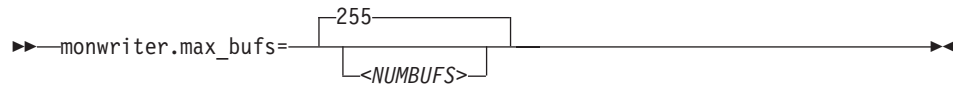
where *NUMBUFS* is the maximum number of monitor sample and configuration data buffers that can exist in the Linux guest at one time. The default is 255.

Module parameters

This section describes how to load and configure those components that have been compiled as separate modules.

```
┌─ Monitor stream application support module parameter syntax ─────────────┐
│                                                                          │
│      ┌─insmod───┐                          ┌─255─────┐                    │
│   ▶▶─┤          ├─monwriter──max_bufs=─────┤         ├───────────────▶◀  │
│      └─modprobe─┘                          └─<NUMBUFS>─┘                   │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```
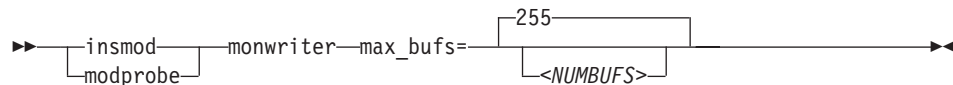
where *NUMBUFS* is the maximum number of monitor sample and configuration data buffers that can exist in the Linux guest at one time. The default is 255.

Example
If you have compiled the monitor stream application support as a separate module, you need to load it before you can work with it. 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 user

Before you begin:
- z/VM version 4.4 or later is required.

Set these options in the CP 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 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 guest.

Refer to *CP Command and Utility Reference*, SC24-6081 for information on the CP MONITOR command.

Working with the monitor stream application support

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* 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 pppppppffnvvrrmm, 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. z/VM recording device driver

The z/VM recording device driver can be used by Linux systems that run as z/VM guests. 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 188).

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 9). There is one device for each recording service. The devices are created for you if the z/VM recording device driver is included in the kernel or they are created when the z/VM recording device driver is loaded as a module.

z/VM recording device nodes

Each recording service has a fixed minor number and a name that corresponds to the name of the service as shown in Table 24:

Table 24. Device names and minor numbers

z/VM recording service	Standard device name	Minor number
*LOGREC	logrec	0
*ACCOUNT	account	1
*SYMPTOM	symptom	2

The major device number for the z/VM recording device driver is assigned dynamically. Read the dev attribute of any one of the z/VM recording devices to find out the major number. The dev attribute is of the form *<major>*:*<minor>*.

Example:

To read the dev attribute of the logrec device:

```
# cat /sys/class/vmlogrdr/logrec/dev
254:0
```

While vmlogrdr registers its driver and device structures with the iucv bus, it also needs to register a class and a class device under /sys/class. The dev attribute is member of that class device. In the example, the major number 254 has been assigned and the minor number is 0 as expected.

185

Creating device nodes for the z/VM recording devices

You access z/VM recording data through device nodes. The required device nodes might be provided for you by udev or by your distribution.

If there are no device nodes, use a command of this form to create a node:

```
mknod -m 440 /dev/<file> c <major> <minor>
```

where:

<file>
> is the file name that you assign to the device node.

<major>
> is the major number that has been dynamically assigned to the z/VM recording device driver (see "z/VM recording device nodes" on page 185).

<minor>
> is the minor number of the recording service for which you are creating the device node.

Example: Using the standard device names (see Table 24 on page 185) and assuming that the major number 254 has been assigned to the z/VM recording device driver you could create the device nodes like this:

```
# mknod -m 440 /dev/logrec c 254 0
# mknod -m 440 /dev/account c 254 1
# mknod -m 440 /dev/symptom c 254 2
```

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.
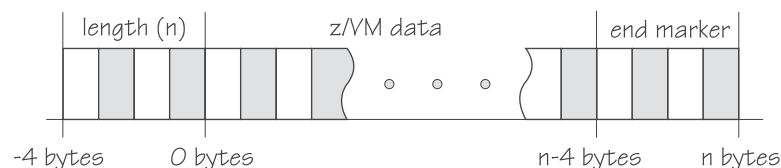
Figure 40. Record structure

Figure 40 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 `ibm.com/vm/pubs/ctlblk.html`.

Further information

For general information about CP recording system services refer to *z/VM CP Programming Services*, SC24-6001.

Building a kernel with the z/VM recording device driver

This section is intended for those who want to build their own kernel.

To build a Linux kernel that supports the z/VM recording device driver you need a kernel that includes the IUCV device driver. You also need to select the CONFIG_VMLOGRDR configuration menu option:

```
Character device drivers
   Support for the z/VM recording system services    (CONFIG_VMLOGRDR)
```

Figure 41. z/VM recording kernel configuration menu option

The z/VM recording device driver can be compiled into the kernel or as a separate module, vmlogrdr.

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 if it has been compiled as a 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 kernel or module parameters for the z/VM recording device driver.

If you have compiled the z/VM recording device driver as a separate module, you need to load it 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 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 189).

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 VM recording status."

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 VM recording status

You can use the record_status attribute of the z/VM recording device driver representation in sysfs to query the VM recording status.

Example

This example runs the 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 Command and Utility Reference*, SC24-6008.

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 188) to enable automatic record collection while a device is open.

You can use a device's autopurge attribute (see "Purging existing records" on page 189) 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 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 16. Watchdog device driver

The watchdog device driver provides Linux user space watchdog applications with access to the z/VM watchdog timer.

Watchdog applications can be used to set up automated restart mechanisms for Linux guests. Watchdog based restart mechanisms are an alternative to a networked heartbeat in conjunction with STONITH (see "STONITH support (snipl for STONITH)" on page 322).

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 5.1 watchdog timer.
- An API for watchdog applications (see "External programming interfaces" on page 197).

Building a kernel with the watchdog device driver

This section is intended for those who want to build their own kernel.

You need to select the kernel configuration option CONFIG_ZVM_WATCHDOG to be able to use the watchdog device driver.

```
Character device drivers --->
   Watchdog Cards  --->
      z/VM Watchdog Timer                     (CONFIG_ZVM_WATCHDOG)
```

Figure 42. Watchdog kernel configuration option

The watchdog device driver can be compiled into the kernel or as a separate module, vmwatchdog.

CONFIG_ZVM_WATCHDOG depends on the common code option CONFIG_WATCHDOG.

What you should know about the watchdog device driver

The watchdog function comprises of 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 43 on page 194).
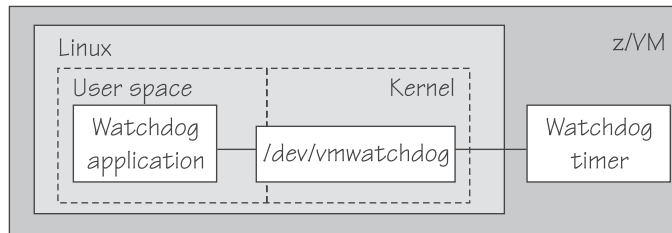
Figure 43. Watchdog application and timer

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.

The default 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

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.

Your distribution might contain a watchdog application. You can also obtain a watchdog application from:

`http://www.ibiblio.org/pub/Linux/system/daemons/watchdog/`

See also the generic watchdog documentation in your 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.

Kernel parameters

This section describes how to configure the watchdog device driver by adding parameters to the kernel parameter line if the watchdog support has been compiled into the kernel.

```
┌─ watchdog kernel parameter syntax ─────────────────────────────────┐
│                                                                     │
│            ┌──── vmwatchdog.cmd="IPL" ────┐  ┌── vmwatchdog.conceal=0 ──────┐        │
│  ▶▶─────────┤                              ├──┤                               ├────────▶ │
│            └── vmwatchdog.cmd=<command> ──┘  └── vmwatchdog.conceal=<conceal_flag> ──┘   │
│                                                                     │
│   ▶──────────┬─────────────────────────────────────────┬────────────────▶◀ │
│             └── vmwatchdog.nowayout=<nowayout_flag> ──┘                 │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```
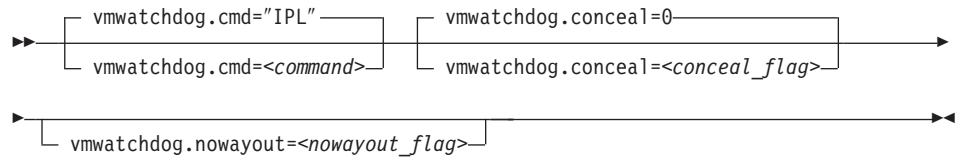
where:

<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" for details. For more information on CP commands refer to *z/VM CP Command and Utility Reference*, SC24-6008.

    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, refer to the "SET CONCEAL" section of *z/VM CP Command and Utility Reference*, SC24-6008.

<nowayout_flag>
    determines what happens when the watchdog device node is closed by the watchdog application.

    If the flag is set to "1", 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.

    The default is determined by the common code kernel configuration option CONFIG_WATCHDOG_NOWAYOUT.

Examples
The following kernel parameters determine 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.

```
vmwatchdog.cmd="ipl b1a0" vmwatchdog.nowayout=0
```

The following example shows how to specify multiple commands to be issued. This is true for both the built-in and module version, after booting the kernel or loading the module.

```
echo -en "cmd1\ncmd2\ncmd3" | cat > /sys/module/vmwatchdog/parameters/cmd
```

To verify that your commands have been accepted, issue:

```
cat /sys/module/vmwatchdog/parameters/cmd
cmd1
cmd2
cmd3
```

Note that it is not possible to specify the multiple commands as kernel parameters during boot time or module parameters while loading the module.

Module parameters

This section describes how to load and configure the watchdog device driver if it has been compiled as separate module.

```
 watchdog module parameter syntax

                               ┌─ cmd="IPL CLEAR" ─┐
►►─ modprobe ─ vmwatchdog ─┼───────────────────────┼─►
                               └─ cmd=<command> ────┘

    ┌─ conceal=0 ──────────────┐
►─┼──────────────────────────┼──────────────────────────────────────►◄
    └─ conceal=<conceal_flag> ─┘   └─ nowayout=<nowayout_flag> ─┘
```
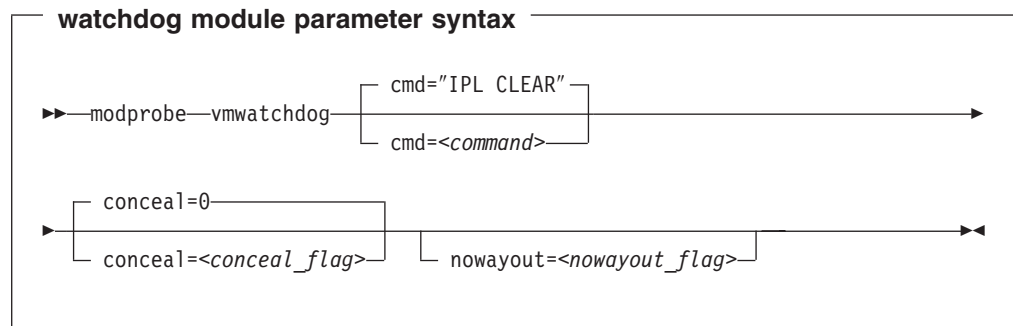
The variables have the same meaning as in "Kernel parameters" on page 194.

Example
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
```

Assuring that a device node exists

The watchdog application on Linux needs a misc character device to communicate with the z/VM watchdog timer. This device node is typically called /dev/vmwatchdog. If your distribution does not create the device node for you (for example, with udev), you need to create a node.

To check if there is already a node issue:

```
# find / -name watchdog
```

If your distribution provides the watchdog device driver as a separate module, be sure to load the module before you check for the node.If there is no node, use major number 10 and minor number 130 to create one. Issue

```
# mknod /dev/watchdog c 10 130
```

External programming interfaces

This section provides information for those who want to program watchdog applications that work with the watchdog device driver.

For information on the API refer to the following files in the Linux source tree:

- /Documentation/watchdog/watchdog-api.txt
- inlcude/linux/watchdog.h

Chapter 17. z/VM CP interface device driver

Using the z/VM CP interface device driver (vmcp), you can send control program (CP) commands to the VM hypervisor and display VM's response.

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:

- As a device node (usually /dev/vmcp)
- As a user space tool (see "vmcp - Send CP commands to the VM hypervisor" on page 328)

If vmcp is built as a module, you must load the module vmcp.ko before you can use vmcp. If your system runs under VM, load the vmcp kernel module automatically.

The vmcp device driver only works under z/VM and does not load if the Linux system runs in an LPAR. If your Linux guest runs under z/VM, you can advise the startup scripts to load the vmcp kernel module automatically during boot, for example:

- On SUSE SLES9 or SLES10 add ″vmcp″ to MODULES_LOADED_ON_BOOT in /etc/sysconfig/kernel.
- On RedHat RHEL4, you can add ″modprobe vmcp″ to the /etc/rc.local.

For any other distribution, refer to the documentation.

The vmcp tool can coexist with the cpint package.

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 *CP Programming Services*, SC24-6084-02) 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 *CP Commands and Utilities Reference*, SC24-6081.

Creating device nodes

User space programs access vmcp devices through *device nodes*. Your distribution might create these device nodes for you or provide udev to create them (see "Device nodes provided by udev" on page 4).

If no device nodes are created for you, you need to create them yourself, for example, with the **mknod** command. Refer to the **mknod** man page for further details.

The `/dev/vmcp` device node is a character device node (major number 10) with a dynamic minor number. During load, a sysfs folder called `class/misc/vmcp/` is created, which contains the dev file for getting the major and minor number of vmcp.

You can use the vmcp 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 25. The vmcp ioctls

Name	Code definition	Description
VMCP_GETCODE	_IOR (0x10, 1, int)	Queries the return code of 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.

Building a kernel with the z/VM CP interface

This section is intended for those who want to build their own kernel. It describes the options you must select in the Linux configuration menu to include the vmcp device driver.

You need to select the kernel configuration option CONFIG_VMCP to be able to build a kernel with user space access to CP commands.

```
Character device drivers
   Support for the z/VM CP interface (VM only)              (CONFIG_VMCP)
```

The vmcp device driver can be compiled into the kernel or as a separate module. In case of a modular build, the module vmcp.ko must be loaded before vmcp can be used.

Setting up the z/VM CP interface

There are no kernel or module parameters for the vmcp device driver.

If you have compiled the vmcp component as a separate module, you need to load the module vmcp.ko before you can work with vmcp. As vmcp does not depend on other modules you can use insmod or modprobe to load vmcp:

```
# modprobe vmcp
```

Chapter 18. Cooperative memory management

The cooperative memory management (CMM) allows an external entity, such as the z/VM resource monitor VMRM, to reduce the memory size of a Linux system. CMM allocates pages to a special page pool. The diagnose 0x10 (or X'10' on the mainframe) is used to indicate 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. There are two ways to grow and shrink the size of the page pool; a /proc based interface and the special message interface which is used by VMRM.

Building a kernel with cooperative memory management

This section is intended for those who want to build their own kernel. To build a kernel with support for cooperative memory management you need to select option CONFIG_CMM in the configuration menu:

```
Base setup --->
 Cooperative memory management                                              (CONFIG_CMM)
 └ /proc interface to cooperative memory management               (CONFIG_CMM_PROC)
 IUCV special message interface to cooperative memory management    (CONFIG_CMM_IUCV)
```

Figure 44. CMM kernel configuration menu option

The cooperative memory management support is available as a module, cmm, or built-in. Note that the special message interface to CMM depends on the configuration setting of the special message support CONFIG_SMSGIUCV:

```
Networking support --->
 S/390 network device drivers
   IUCV support (VM only)
     └ IUCV special message support (VM only)                      (CONFIG_SMSGIUCV)
```

Figure 45. IUCV special message support kernel configuration menu option

Setting up cooperative memory management

This section describes how to set up a Linux guest to participate in the cooperative memory management under z/VM.

Kernel parameters

This section describes how to configure cooperative memory management if it has been compiled into the kernel. You configure cooperative memory management by adding parameters to the kernel parameter line.
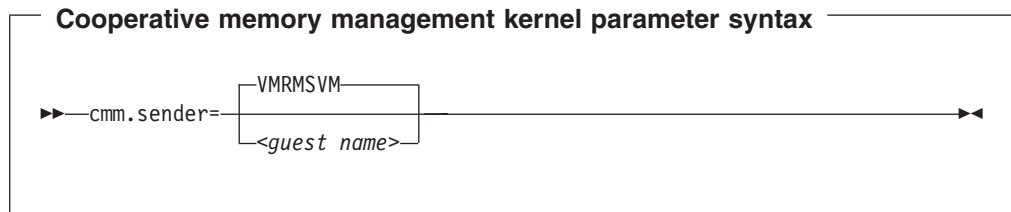
```
┌─ Cooperative memory management kernel parameter syntax ──────────────────┐
│                                                                          │
│                              ┌─VMRMSVM──────┐                            │
│  ►►── cmm.sender=────────────┤              ├──────────────────────►◄    │
│                              └─<guest name>─┘                            │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```

where:

<guest name>
> is the name of the z/VM guest that is allowed to send messages to the module
> through the special messages interface. The default guest name is VMRMSVM.
> This is used if the parameter is omitted.

Loading the cooperative memory management module

The cooperative memory management module might have been compiled as a
module. Use the **insmod** or **modprobe** command to load the module. Refer to the
respective man pages for command details.
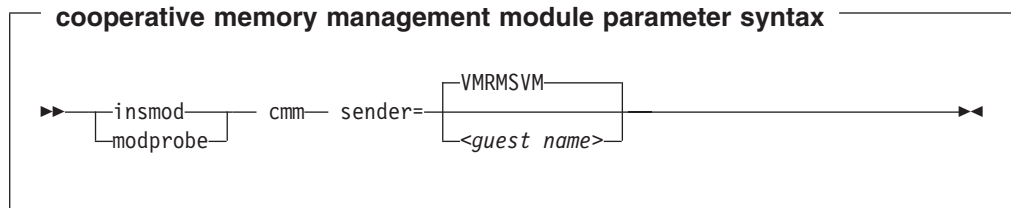
```
┌─ cooperative memory management module parameter syntax ──────────────────┐
│                                                                          │
│        ┌─insmod───┐                   ┌─VMRMSVM──────┐                   │
│  ►►────┤          ├── cmm── sender=───┤              ├──────────────►◄    │
│        └─modprobe─┘                   └─<guest name>─┘                   │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```

where <guest name> is the name of the z/VM guest that is allowed to send
messages to the module through the special messages interface. The default guest
name is VMRMSVM. This is used if the parameter is omitted.

Example

To load the cooperative memory management module and allow the guest TESTID
to send messages:

```
# modprobe cmm sender=TESTID
```

The setup of the external resource manager that is supposed to send messages to
the cooperative memory management module is beyond the scope of this book.

Further information

For more information on VMRM, see the chapter on VMRM in *z/VM V5R1.0
Performance*, SC24-6109.

Working with cooperative memory management

This section describes typical tasks that you need to perform when working with cooperative memory management.

To read the current size of the static page pool:

```
# cat /proc/sys/vm/cmm_pages
```

To set the target size of the static page pool:

```
# echo <pages>> /proc/sys/vm/cmm_pages
```

To read the current size of the timed page pool:

```
# cat /proc/sys/vm/cmm_timed_pages
```

To add a number of pages to the target size of the static page pool

```
# echo <pages>> /proc/sys/vm/cmm_timed_pages
```

To read the release rate of the timed page pool

```
# cat /proc/sys/vm/cmm_timeout
```

The result is a pair of numbers ″<pages> <seconds>″. The pages from the timed page pool are freed with a rate of <pages>/<seconds>.

To set the release rate of the timed page pool

```
# echo "<pages> <seconds>" > /proc/sys/vm/cmm_timeout
```

Part 5. Generic features

This part describes:
- Chapter 19, "Console device drivers"
- Chapter 20, "Generic cryptographic device driver"
- Chapter 21, "Channel measurement facility"
- Chapter 22, "Control program identification"
- Chapter 23, "OSA-Express SNMP subagent support"
- Chapter 25, "Data execution protection for user processes"
- Chapter 24, "IPL parameter interface"

It also provides a summary of the System z-specific kernel configuration options you can find in the Linux kernel configuration menu. Some of these options provide additional features that are not otherwise described in this book.
- Chapter 26, "Other features kernel builders should know about," on page 251

> **Note**
>
> For prerequisites and restrictions for these device drivers and features refer to the kernel 2.6 October 2005 stream pages on developerWorks at:
>
> `ibm.com/developerworks/linux/linux390/october2005_recommended.html`

Chapter 19. Console device drivers

The Linux on System z console device drivers support devices for basic Linux control, for example, for booting Linux.

The control device can be a 3270 terminal or an IBM mainframe's *hardware console*. The device driver accesses the hardware console through the service-call logical processor (SCLP) interface.

If Linux is running as a VM guest or on a P/390, the control device can also be a 3215 terminal.

Note that "terminal" and "console" have special meanings in Linux, which should not be confused with the System z usage.

The Hardware Management Console (HMC) is a Web application that provides access to the System z9 or zSeries Service Element (SE) which is in overall control of the mainframe hardware.

A mainframe *terminal* is any device which gives a user access to applications running on the mainframe. This could be a real device such as a 3215 linked to the system through a controller, or it can be a terminal emulator on a networked device.

The Linux console and the Linux terminals are different applications which both run on System z *terminals*. Linux terminals are devices through which users interact with Linux and Linux applications. The Linux console is a device that handles user interactions with the Linux kernel. The Linux console traffic is directed to one of the Linux terminals.

In this chapter, we use console and terminal to refer to a Linux console or terminal, unless indicated otherwise.

Console features

The console device drivers support the following devices and functions:
- Supports an SCLP console
- Supports a 3215 terminal for the United States code page (037)
- Supports a 3270 terminal
- Provides a line mode or full-screen mode typewriter terminal
- Provides Linux console output on a designated terminal

What you should know about console devices

This section provides information on console device names and nodes, on different modes of console devices, and considerations when running the console in specific environments.

Console modes

The console can be a line-mode terminal or a full-screen mode terminal.

On a full-screen mode terminal, pressing any key immediately results in data being sent to the TTY routines. Also, terminal output can be positioned anywhere on the screen. This allows for advanced interactive capability when using terminal based applications.

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.

Console device names

Table 26 summarizes the supported console devices, with their names, and device numbers:

Table 26. Supported console devices

Device driver	Device name	Major	Minor
Line-mode hardware console	ttyS0	4	64
Full-screen mode hardware console	ttyS1	4	65
3215 terminal	ttyS0	4	64
3270 terminal	tty3270	227	0

Note: The specifications for the line-mode hardware console and the 3215 device are identical. The device name and numbers are assigned to whatever device is present or to the device you specify with the conmode parameter (see "conmode parameter" on page 214). You cannot have both devices simultaneously.

You require a device node to make a console device available to other programs (see "Assuring device nodes" on page 215).

Using the hardware console

The following applies to both the line-mode terminal and the full-screen mode terminal on the hardware console:

* There can only be one active terminal session on an HMC.
* Security hint: Always end a terminal session by explicitly logging off (for example, type "exit" and press Enter). Simply closing the terminal window leaves the session active and the next user opening the window resumes the existing session without a logon.
* Slow performance of the hardware console 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.

Magic sysrequest function

This section applies to systems where the common code kernel option CONFIG_MAGIC_SYSRQ has been activated only.

The two characters "^-" followed by a third character invoke the *magic sysrequest* function. The various debugging and emergency functions that can be performed are specified by the third character.

This feature can be switched on or off during runtime by echoing "1" (on) or "0" (off) to `/proc/sys/kernel/sysrq`. The possible sequences are:

Table 27. Magic sysrequest commands

Enter	To
^-b	Re-IPL immediately.
^-s	Emergency sync all file systems.
^-u	Emergency remount all mounted file systems read-only.
^-t	Show task info.
^-m	Show memory.
^- followed by a digit (0 to 9)	Set the console log level.
^-e	Terminate all tasks.
^-i	Kill all tasks except init.

Console special characters on line-mode terminals

The line-mode console does not have a control key. That makes it impossible to enter control characters directly. To be able to enter at least some of the more important control characters, the character "^" has a special meaning in the following cases:

Table 28. Control characters

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.

If you are running under VM using the hardware console emulation, you will have to use the **CP VINPUT** command to simulate the Enter and Spacebar keys.

The Enter key is simulated by entering:

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

The Spacebar key is simulated by entering two blanks followed by "\n":

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

VM console line edit characters

When running under VM, the control program (CP) defines five characters as line editing symbols. Use the **CP QUERY TERMINAL** command to see the current settings. The defaults for these depend on the terminal emulator you are using, and can be reassigned by the CP system operator or by yourself using the **CP TERMINAL** command to change the setting of LINEND, TABCHAR, CHARDEL, LINEDEL or ESCAPE. Table 29 shows the most commonly used settings:

Table 29. 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 the line edit symbols # │ @ [" (or # │ @ ¢ ") you need to type the character pairs "# "│ "@ "["" (or "# "│ "@ "¢ ""). In particular, to enter the quote character you must type it twice.

Example

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"
```

Using VInput

VINPUT is a VM CP command. It can be abbreviated to **VI** but must not be confused with the Linux command **vi**.

If you use the hardware console driver running under VM (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 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.

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 VMSG LS -L
```

and

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

are equivalent.

Case conversion

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

Linux and bash are case sensitive and require some specifications with uppercase characters. To allow uppercase characters to be passed to Linux, the hardware console uses an escape character "%" under VM to distinguish between uppercase and lowercase characters. Characters enclosed by two "%" are treated as uppercase.

This behavior and the escape character "%" are adjustable at build-time by editing the driver sources.

Examples: If you type `VInput VMSG echo $PATH`, the device driver gets `ECHO $PATH` and converts it into `echo $path`.

* The first line is your input line, the second the line processed by CP and the third 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 with a variable $path that 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

To include the escape character in the command passed to Linux, you need to type it twice. If you are using the standard settings according to "VM console line edit characters" on page 210, you need to specify two quotes to pass a single quote to Linux.

Example: The following command passes an string in quotes 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 "VM console line edit characters" on page 210, you need to specify "# to pass # to Linux.

If you specify the end of line character without a leading escape character, 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
```

Console 3270 emulation

If you are accessing the VM console from Linux by using the x3270 emulator, 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
```

Further information

For more information on VINPUT refer to *z/VM CP Command and Utility Reference*, SC24-6008.

Building a kernel with the console device drivers

This section is intended for those who want to build their own kernel. It describes the options you must select in the Linux configuration menu to include the console device drivers.

Figure 46 summarizes the kernel configuration menu options that are relevant to the console device drivers:

```
Character device drivers
  Support for locally attached 3270 tubes        (CONFIG_TN3270)
    └─Support for console on                     (CONFIG_TN3270_CONSOLE)
        3270 line mode terminal
  Support for locally attached 3215 tubes        (CONFIG_TN3215)
    └─Support for console on                     (CONFIG_TN3215_CONSOLE)
      3215 line mode terminal
  Support for SCLP                               (CONFIG_SCLP)
  ├─Support for SCLP line mode terminal          (CONFIG_SCLP_TTY)
  │  └─Support for console on                    (CONFIG_SCLP_CONSOLE)
  │    SCLP line mode terminal
  └─Support for SCLP VT220-compatible terminal   (CONFIG_SCLP_VT220_TTY)
     └─Support for console on                    (CONFIG_SCLP_VT220_CONSOLE)
       SCLP VT220-compatible terminal
```

Figure 46. Console kernel configuration menu options

CONFIG_TN3270
  Allows an IBM 3270 line-mode terminal to run a Linux system console.

CONFIG_TN3270_CONSOLE
  Prints kernel errors and kernel warnings to the IBM 3270 line-mode terminal in addition to the normal output.

CONFIG_TN3215
  Allows an IBM 3215 line-mode terminal to run a Linux system console.

CONFIG_TN3215_CONSOLE
  Prints kernel errors and kernel warnings to the IBM 3215 line-mode terminal in addition to the normal output.

CONFIG_SCLP
  Includes support for the IBM SCLP interface to the service element. This option is required to enable Linux to respond to "signal quiesce" requests.

CONFIG_SCLP_TTY
  Allows an SCLP line-mode terminal to run a Linux system console.

CONFIG_SCLP_CONSOLE
  Prints kernel errors and kernel warnings to the SCLP line-mode hardware console in addition to the normal output on the TTY device.

CONFIG_SCLP_VT220_TTY
  Allows an SCLP VT220-compatible terminal in full-screen mode to run a Linux system console.

CONFIG_SCLP_VT220_CONSOLE
  Prints kernel errors and kernel warnings to the full-screen hardware console in addition to the normal output on the TTY device.

All console device driver components are compiled into the kernel.

Setting up the console device drivers

This section describes the kernel parameters, commands, and `/etc/inittab` entries that you can use to configure the console device drivers. Because all console device driver components are compiled into the kernel, there are no module parameters.

Console kernel parameter syntax

There are three kernel parameters for the console:

conmode   to override the default console of an environment.

condev   to define a 3215 device to a P/390.

console   to activate a console and designate a *preferred* console.

Console kernel parameter syntax

```
►►──┬─────────────────────────────┬──┬──────────────────┬──────────────►
    └─conmode=──┬─hwc──┬───────────┘  └─condev=<cuu>──────┘
               ├─3270─┤
               └─3215─┘


►──┬─console=ttyS0───────────────────────────┬───────────────────────►◄
   └─console=<name1>─┬────────────────────┬──┘
                     └─console=<name2>────┘
```

Note: If you specify multiple parameter=value pairs, separate them with a blank.

---
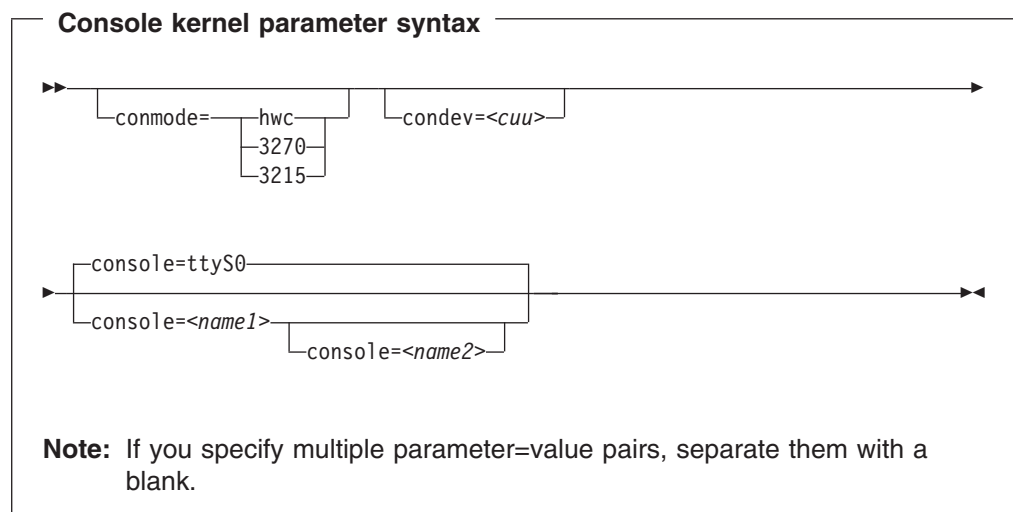
where:

<cuu>
> is the device 'Control Unit and Unit' number, and may be expressed in hexadecimal form (preceded by `0x`) or in decimal form.

<name1>
> is the standard name of the active console (ttyS0 | ttyS1).

<name2>
> is the standard name of the second console if both are to be active (ttyS0 | ttyS1). If both consoles are specified, this is the preferred console.

conmode parameter
The device drivers for the 3215 terminal, for the 3270 terminal, and for the hardware console can be compiled into the Linux kernel. If more than one driver is present, the default console device driver is chosen at runtime according to the environment:

Table 30. Default console device driver

LPAR or native	hardware console device driver or 3270 device driver
VM	3215 or 3270 console device driver, depending on the guest's console settings (the `CONMODE` field in the output of `#CP QUERY TERMINAL`).
P/390	3215 console device driver

Use the conmode parameter to override the default.

Example: To use the hardware console in a z/VM environment specify:

```
conmode=hwc
```

condev parameter

This kernel parameter applies only to the 3215 console device driver if used on a P/390. This supplies the device driver with the device number of the 3215 device. The reason that this parameter is needed is that there is no guaranteed method of recognizing a 3215 device attached to a P/390.

Example: To instruct the device driver to use device number 0x001F for the 3215 terminal specify:

```
condev=0x001f
```

or:

```
condev=31
```

console parameter

The console parameter applies only if there is more than one console device available (for example, when using the SCLP console device driver).

Only active consoles receive Linux operating system messages and only one console can be the *preferred* console. The preferred console is used as initial input and output device, beginning at the stage of the boot process when the 'init'-script is called. Messages issued by programs that are run at this stage are therefore only displayed on the preferred console. On default, ttyS0 is active and used as the preferred console.

Use the console parameter to activate ttyS1 instead of ttyS0, or to activate both ttyS0 and ttyS1 and designate one of them as the preferred console.

Examples:

*   To activate the full-screen console device driver instead of the line-mode console driver, add the following line to the kernel command line:
    ```
    console=ttyS1
    ```
*   To activate both consoles, provide a specification for both drivers, for example:
    ```
    console=ttyS0 console=ttyS1
    ```

    The last statement determines the preferred console, `ttyS1` in the example.

Assuring device nodes

User space programs access console devices by *device nodes*. If your distribution does not create these device nodes early in the boot process, Linux will not boot and you will not have a command prompt from where you can create the nodes yourself.

In this case, you can create the nodes from a support system that has access to the failed system's devices. For example, you can use the following commands to create the nodes:

```
# mknod /dev/ttyS0 c 4 64
# mknod /dev/ttyS1 c 4 65
# mknod /dev/tty3270 c 227 0
```

Setting up a line-mode terminal

The line-mode terminals are primarily intended for booting Linux. The preferred user access to a running Linux on System z is through a networked terminal emulation such as telnet or ssh. The 3215 and 3270 console device drivers always provide a line-mode terminal. The hardware console device driver can provide a line-mode terminal or a VT220-like full-screen mode terminal.

Tip: If the terminal does not provide the expected output, ensure that dumb is assigned to the TERM environment variable. For example, issue 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. Only the hardware console device driver can provide a VT220-like full-screen mode terminal.

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

```
# export TERM=linux
```

To set TERM=linux at startup add a line of this form to the /etc/inittab file:

```
<id>:2345:respawn:/sbin/agetty -L 9600 ttyS1 linux
```

where *<id>* is a unique identifier for the entry in the inittab file.

Be sure not to provide multiple entries for ttyS1. For more details see the man page for the inittab file.

Enabling a terminal for user log-ins

To allow user log-ins from a terminal, add a line of this form to the /etc/inittab file:

```
<id>:2345:respawn:/sbin/mingetty <dev> --noclear
```

where:

<id>   is a unique identifier for the entry in the inittab file.

<dev>   is the device name (see Table 26 on page 208).

Your Linux system's /etc/inittab file might already have an entry for a terminal. Be sure not to provide multiple entries for the same device. See Table 26 on page 208 for the device node names. If an existing entry uses a different name and you

are not sure how it maps to the names of Table 26, you can comment it out and replace it. When referring to the device node in a command or parameter, always use the names of Table 26.

For more details see the man page for the inittab file.

Example
To enable a device `ttyS0` for user log-ins specify, for example:

```
1:2345:respawn:/sbin/mingetty ttyS0 --noclear
```

Setting a TTY device online or offline

You can use the **chccwdev** command ("chccwdev - Set a CCW device online" on page 269) to set a TTY 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.

Note: Setting a CCW device online and offline in this context is only useful with additional TTYs attached through the 3270 TTY device driver and not for the console device.

Examples
- To set a TTY device `ttyS0` online issue:

```
# chccwdev -e ttyS0
```

or

```
# echo 1 > /sys/bus/ccw/drivers/3270/ttyS0/online
```

- To set a TTY device `ttyS0` online issue offline issue:

```
# chccwdev -d ttyS0
```

or

```
# echo 0 > /sys/bus/ccw/drivers/3270/ttyS0/online
```

Chapter 20. Generic cryptographic device driver

Some cryptographic processing in Linux can be off-loaded from the CPU and performed by dedicated coprocessors. Several of these coprocessors are available offering a range of performance and features. The generic cryptographic device driver (zcrypt) 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 coprocessors supported are:
- PCI Cryptographic Coprocessor (PCICC)
- PCI Cryptographic Accelerator (PCICA)
- PCI-X Cryptographic Coprocessor (PCIXCC) - see below
- Crypto Express2 Coprocessor (CEX2C)
- Crypto Express2 Accelerator (CEX2A)

PCIXCC coprocessors are further distinguished by their licensed internal code (LIC) level:
- PCIXCC (MCL3) as of LIC EC J12220 level 29
- PCIXCC (MCL2) with a LIC prior to EC J12220 level 29

Note: When Linux is running under VM and an accelerator card (PCICA or CEX2A) is present any cryptographic coprocessor cards will be hidden. For VM guests all cards are supported as of z/VM 5.1.

Supported facilities

The cryptographic device driver supports these cryptographic operations:
- Encryption using the Rivest-Shamir-Adleman (RSA) exponentiation operation using either a modulus-exponent (Mod-Expo) or Chinese-Remainder Theorem (CRT) key.
- Decryption using the counterpart of the operation and key used for encryption.

Elements of zcrypt

This section provides information on the software you need to use zcrypt and the use it makes of cryptographic hardware.

Software components

To run programs that use the zcrypt device driver for encryption, you need:
- the device driver module or modules (see "Building a kernel with the zcrypt device driver" on page 221),
- the libica library, unless applications call the device driver directly

  (the libica library can also perform generation of public/private key pairs, encryption, decryption, signing and signature verification through software),
- and the openCryptoki library if applications use the PKCS #11 API.

The openCryptoki library requires the libica library. Applications can either call the libica library directly or use it indirectly through the openCryptoki PKCS #11 API. Applications can also interface directly to the z90crypt driver module. Figure 47 illustrates the software relationships:

Application x	Application y	Application z
openCryptoki (pkcs#11)		
libica		
z90crypt		
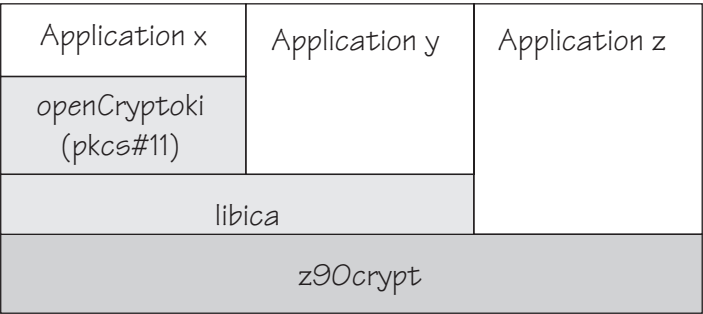
Figure 47. zcrypt device driver interfaces

In Figure 47 "Application x" is an application that uses the PKCS #11 API, "Application y" uses the libica library, and "Application z" uses the z90crypt interface directly (see "External programming interfaces" on page 227).

See "Setting up the zcrypt device driver" on page 222 for information on how to set up the different components.

Dependencies

If you are running Linux under VM then z/VM 5.1 is required.

For the newer cards (CEX2A and CEX2C) you require a System z9 or later.

For the cryptographic functions you should use the libica library. This library is part of the openCryptoki project (see "The libica library" on page 225).

Hardware restrictions

If you have a PCICC only, or are attempting to use a CRT key on a system with PCIXCC (MCL2) only, you need to ensure that your data is PKCS-1.2 padded. In this case, the zcrypt device driver or the cryptographic hardware might change the padding. If the padding is not done correctly, the cryptographic operations are performed in software.

If you have at least one PCICA, PCIXCC (MCL3), CEX2C, or CEX2A, or if you are only using Mod-Expo keys with a PCIXCC (MCL2), you do not need to ensure PKCS-1.2 padding. In this case the padding remains unchanged.

Performance considerations

Polling thread
This driver can run in two modes: 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 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 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 notable CPU load.

Load balancing

To maximize performance the driver uses a load balancing algorithm to distribute requests across all available AP bus devices[1]. 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. When a device processed a request it 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 zcrypt device driver assigns work to cryptographic devices according to device type in the following order of preference:
1. CEX2A
2. PCICA
3. PCIXCC
4. CEX2C
5. PCICC

Further information

- For information on RSA–PKCS 1.2-padding visit http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/.
- For information on how to set up cryptographic hardware on your mainframe refer to *zSeries Crypto Guide Update*, SG24-6870.
- For CP commands related to cryptographic devices refer to *z/VM CP Command and Utility Reference*, SC24-6008.

Building a kernel with the zcrypt device driver

This section describes the options you must select in the Linux configuration menu to include the zcrypt device driver.

You need to select the option CONFIG_ZCRYPT to include the cryptographic device driver.

```
Cryptographic devices
    Support for PCI-attached cryptographic adapters     (CONFIG_ZCRYPT)
```

Figure 48. zcrypt kernel configuration menu option

The CONFIG_ZCRYPT option controls whether the driver is compiled into the kernel or built as a separate module.

The device driver can be compiled as multiple kernel modules or a monolithic kernel module. The default is to create multiple modules. You need to select the option CONFIG_ZCRYPT_MONOLITHIC to build the monolith.

---

1. See "The sysfs interface" on page 225

```
Cryptographic devices
    Support for monolithic cryptographic device driver   (CONFIG_ZCRYPT_MONOLITHIC)
```

Figure 49. zcrypt monolithic configuration menu option

If configured as multiple kernel modules the following modules will be built:

ap.ko   AP bus module

zcrypt_api.ko
        request router module

zcrypt_cex2a.ko
        card driver for CEX2A cards

zcrypt_pcica.ko
        card driver for PCICA cards

zcrypt_pcicc.ko
        card driver for PCICC cards

zcrypt_pcixcc.ko
        card driver for PCIXCC and CEX2C cards

If configured as monolithic kernel module a single module will be built:

- **z90crypt.ko** (this name is used for backward compatibility).

Setting up the zcrypt device driver

This section describes the zcrypt kernel parameters and the z90crypt module, and how to install additional components required by the device driver.

This section also describes how to create the required device node. It further describes how to set up the cryptographic devices on your LPAR, and where applicable, VM, to make it available to your Linux instance.

Kernel parameters

This section describes how to configure the zcrypt device driver if zcrypt has been compiled into the kernel. You can configure the device driver by adding the parameters to the kernel parameter line.

The driver supports two kernel parameters: **domain** and **poll_thread**. **domain** is an integer argument and sets the cryptographic domain index to be used by the devices. If not set the domain index with the maximum number of devices will be used. **poll_thread** is a integer argument and enables a polling thread to increase cryptographic performance (see also "Polling thread" on page 220). The default value is 1, which means the polling thread is enabled.

You need to specify the domain parameter only if you are running Linux in an LPAR for which multiple cryptographic domains have been defined.

```
┌─ zcrypt kernel parameter syntax ─────────────────────────────────────┐
│                                                                       │
│            ┌─domain=-1──────────┐   ┌─poll_thread=1─┐                  │
│   ►►──────┤                     ├───┤               ├─────────────►◄   │
│            └─domain=<domain>────┘   └─poll_thread=0─┘                  │
│                                                                       │
└───────────────────────────────────────────────────────────────────────┘
```
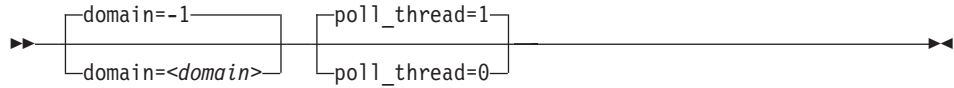
where *<domain>* is an integer in the range from 0 to 15 that identifies the cryptographic domain for the Linux instance and *<poll_thread>* is 1 (enabled) or 0 (disabled).

The specification "domain=-1" causes the device driver to attempt to autodetect the domain to use, which will be the one with the most devices attached. This is the default.

Example

The following kernel parameter line specification makes the zcrypt device driver operate within the cryptographic domain "7" with **poll_thread** enabled:

```
domain=7 poll_thread=1
```

Module parameters

If zcrypt is not compiled into the kernel it can be compiled as a single monolithic module or as a set of discrete modules. See "Building a kernel with the zcrypt device driver" on page 221 for details.

Monolithic module parameters

This section describes how to load and configure the zcrypt device driver if it has been compiled as a separate monolithic module.

```
┌─ z90crypt module syntax ──────────────────────────────────────────────┐
│                                                                        │
│            ┌─modprobe─┐          ┌─domain=-1──────────┐  ┌─poll_thread=1─┐ │
│   ►►──────┤          ├── z90crypt ─┤                   ├──┤               ├─►◄│
│            └─insmod───┘          └─domain=<domain>────┘  └─poll_thread=0─┘ │
│                                                                        │
└────────────────────────────────────────────────────────────────────────┘
```
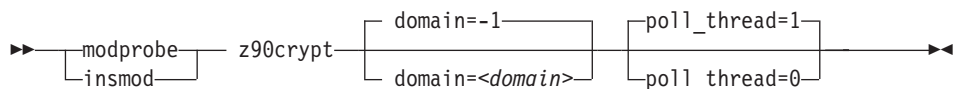
where *<domain>* is an integer in the range from 0 to 15 that identifies the cryptographic domain for the Linux instance and *<poll_thread>* is 1 (enabled) or 0 (disabled). Omit the domain parameter if only one cryptographic domain is defined for the LPAR where your Linux instance runs.

The specification "domain=-1" causes the device driver to attempt to autodetect the domain to use, which will be the one with the most devices attached. This is the default.

Refer to the respective man page for details on **modprobe** or **insmod**.

Examples:

* This example loads the zcrypt device driver module if Linux runs in an LPAR with only one cryptographic domain:

```
# modprobe z90crypt
```

- This example loads the zcrypt device driver module and makes zcrypt operate within the cryptographic domain "1":

```
# modprobe z90crypt domain=1
```

Discrete module parameters

This section describes how to load and configure the zcrypt device driver if it has been compiled as discrete modules.

To load the AP bus module:

ap module syntax

```
►►──┬─modprobe─┬── ap ──┬──────domain=-1──────┬──┬──poll_thread=1──┬──►◄
    └─insmod───┘        └─ domain=<domain> ───┘  └──poll_thread=0──┘
```
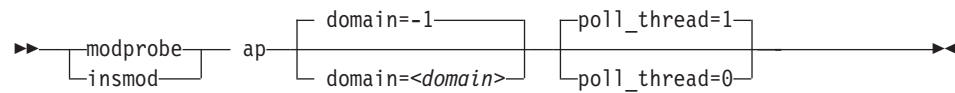
where the parameters are the same as those described in "Monolithic module parameters" on page 223.

All other modules required will be loaded automatically.

Note: To support automatic load of card driver modules the following rule has to be added to udev:

```
ACTION=="add", SUBSYSTEM=="ap", MODALIAS=="*", \ RUN+="/sbin/modprobe $modalias"
```

To unload the zcrypt device driver you must unload all modules manually:

Unload module syntax

```
►►──modprobe -r zcrypt_cex2a zcrypt_pcixcc zcrypt_pcicc zcrypt_pcica zcrypt_api ap──►◄
```

It is important to list them in the order given. You can also use **rmmod** but in this case you must only unload modules which have actually been loaded. For example:

```
rmmod zcrypt_pcicc zcrypt_pcica zcrypt_api ap
```

may be used if only the PCICC and PCICA modules are loaded.

Examples:

- This example loads the discrete zcrypt device driver module **ap** if Linux runs in an LPAR with only one cryptographic domain:

```
# modprobe ap
```

- This example loads the discrete zcrypt device driver module **ap** to operate within the cryptographic domain "1":

```
# modprobe ap domain=1
```

The libica library

You can obtain the libica library from the SourceForge Web site at:

`http://sourceforge.net/projects/opencryptoki`

You can find the release details with the module under the **Files** category.

Both a 31-bit and a 64-bit version are available. The 64-bit version includes the 31-bit compatibility code.

The openCryptoki library

You can obtain the openCryptoki library from the SourceForge Web site at:

`http://sourceforge.net/projects/opencryptoki`

You can find the release details with the module on the **Files** category.

To be able to configure openCryptoki (with pkcsconf) user root must be a member of group pkcs11.

The sysfs interface

The zcrypt cryptographic driver utilizes the device model introduced with the Linux 2.6 kernel series. It introduces a new bus called "AP" which can be found under **/sys/bus/ap**.

The following attributes are defined at AP bus level:

/sys/bus/ap/ap_domain
Read-only attribute representing the domain index used for all AP devices.

/sys/bus/ap/config_time
Read-write attribute representing the interval in seconds for re-scanning the AP bus for new or gone devices.

/sys/bus/ap/poll_thread
Read-write attribute indicating whether a polling thread is to be used to increase cryptographic performance. By writing 0 or 1 to this attribute the poll thread can be disabled or enabled. (See "Polling thread" on page 220).

For each cryptographic adapter a new directory in /sys/bus/ap/devices is created using the following naming convention: 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. Within each device directory the following attributes can be found:

depth Read-only attribute representing the input queue length for this device.

hwtype
Read-only attribute representing the hardware type for this device. The following values are defined:

3   PCICC cards,

4   PCICA cards,

5   PCIXCC cards,

| | **6** | CEX2A cards,
| | **7** | CEX2C cards.

modalias
Read-only attribute representing an internally used device bus-ID.

online Read-write attribute representing the online status for this AP device. Writing 0 or 1 to this attribute sets this device offline or online.

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:
- PCICC
- PCICA
- PCIXCC_MCL2
- PCIXCC_MCL3
- CEX2C
- CEX2A

Debugging messages

By defining a macro called ZCRYPT_DEBUG at compile time debugging messages can be enabled.

Setting up for the 31-bit compatibility mode

31-bit applications can access the 64-bit zcrypt driver by using the 31-bit compatibility mode.

Assuring that you have a device node

User space programs address cryptographic devices through a single device node. Both the major and minor number can be dynamic, depending on your Linux distribution and configuration. To provide the node you need either udev or hotplug support.

Using udev

If udev support is enabled (see "Device nodes provided by udev" on page 4), z90crypt is assigned to the miscellaneous devices. The major device number is then that of the misc devices. You can find it as the value for the entry "misc" in `/proc/devices`.

The minor number is dynamically assigned and you can find it in `/proc/misc` as the value for the entry "z90crypt".

If the device node `/dev/z90crypt` is not created for you, you can create it yourself by issuing a command of this form:

```
# mknod /dev/z90crypt c <misc_major> <dynamic_minor>
```

Checking hardware status, activating and deactivating devices under zcrypt

Cryptographic devices in general are not directly accessed by users but through user programs. The functions of checking hardware status, activating devices, and deactivating them, are performed by program calls to the sysfs interface. See "The sysfs interface" on page 225 for more details.

Examples

- To set a cryptographic device with bus device 0x3e online issue:

```
echo 1 > /sysfs/bus/ap/devices/card3e/online
```

- To set a cryptographic device with bus device 0x3e offline issue:

```
echo 0 > /sysfs/bus/ap/devices/card3e/online
```

- To check the online status of the cryptographic device with bus ID 0x3e issue:

```
cat /sysfs/bus/ap/devices/card3e/online
```

The value is '1' if device is online or '0' otherwise.

External programming interfaces

This section provides information for those who want to circumvent libica and directly access the zcrypt device driver.

Refer to `include/asm-s390/zcrypt.h` in the Linux kernel source tree for different structures that you might want to use.

Outline of a decryption program

The following steps are required:

1. Get a device handle for zcrypt.

   Example:

   `dh= open("/dev/zcrypt", O_RDWR)`

2. Create and load one of the following structures

   - ica_rsa_modexpo (see "The ica_rsa_modexpo structure" on page 228)
   - ica_rsa_modexpo_crt (see "The ica_rsa_modexpo_crt structure" on page 228)

   Both structures are defined in zcrypt.h. In the structure, you define the private key to be used and set the input buffer pointer to the data you want to decrypt and the output buffer pointer for the decrypted data.

3. Invoke ioctl to activate zcrypt:

   `<rc>= ioctl(dh, <function code>, <structure pointer>)`

   where:

<function code>	is ICARSAMODEXPO for structure type ica_rsa_modexpo or ICARSACRT for structure type ica_rsa_modexpo_crt

<structure pointer>	is a pointer to the structure you created, of the type ica_rsa_modexpo or ica_rsa_modexpo_crt
<rc>	is the variable for the return code

Example:

```
myrc = ioctl(dh, ICARSAMODEXPO, &mycryptmex);
```

4. Obtain the decrypted and decoded data from the output buffer in the structure.

The ica_rsa_modexpo structure

The ica_rsa_modexpo structure is defined in the zcrypt header file, zcrypt.h.

The (private) key consists of the exponent in *b_key and the modulus in *n_modulus. Both of these are hexadecimal representations of large numbers. The length L of *n_modulus must be in the range 64 – 256.

Both the input data and the exponent b_key must be of the same length L as the modulus. If they are shorter than the modulus, they must be padded on the left with zeroes. The output data length must be at least L.

The ica_rsa_modexpo_crt structure

The ica_rsa_modexpo_crt structure is defined in the zcrypt header file, zcrypt.h.

The ica_rsa_modexpo_crt structure is similar to the ica_rsa_modexpo structure but has been defined so that the Chinese Remainder Theorem (CRT) can be used in decryption. zcrypt performs better if the CRT definition is used. The key-definition fields are all in hexadecimal representation. They have these meanings and limitations:

- *bp_key and *bq_key are the prime factors of the modulus. In zcrypt the modulus is (*bp_key) × (*bq_key). The resulting length L of the modulus, in hexadecimal representation, must be found before these fields are defined.
- *np_prime and *nq_prime are exponents used for *bp_key and *bq_key respectively.
- *u_mult_inv is a coefficient used in the zcrypt implementation of decryption by CRT.
- *bp_key, *np_prime, and *u_mult_inv must all be of length 8 + L/2
- *bq_key and *nq_prime must both be of length L/2

The input data length must be L, and the output data length must be at least L

Querying the hardware status

There is an ioctl interface for checking on underlying hardware in zcrypt. There are a number of ioctls for each status needed. These ioctls are defined in the header file zcrypt.h. When control returns, you will have the information you requested.

Example:

```
<rc> = ioctl(<dh>, Z90STAT_PENDINGQ_COUNT, &<int_variable>);
```

where:

<rc>	is the variable for the ioctl return code
<dh>	is the variable for the zcrypt device handle

<int_variable>   is the variable you want to use for the returned information; in the
example the number of requests sent to the devices awaiting the
reply.

Returns from ioctl

0 means everything went well and the result is in your output buffer.

A return code of -1 indicates that an error has occurred and that an error code is
also returned. For a list of possible error codes returned in **errno** refer to
`/usr/include/asm/errno.h`

Chapter 21. 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.
- 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)

Building a kernel with the channel measurement facility

This section is intended for those who want to build their own kernel.

The channel measurement facility is always included in the Linux 2.6 kernel. You do not need to select any options.

If you want to access DASD data with applications written for the kernel 2.4 DASD channel measurement facility, see "the CONFIG_DASD_CMB option" on page 36 for required options.

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. │
│                                                                    │
└────────────────────────────────────────────────────────────────────┘
```
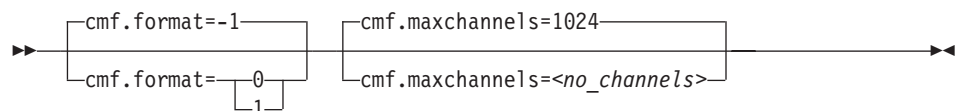
where:

cmf.format
defines the format, "0" for basic and "1" for extended, of the channel measurement blocks. The default, "-1", uses the extended format for z990 and later mainframes and the basic format for earlier mainframes.

cmf.maxchannels=<i><no_channels></i>
limits the number of devices for which data measurement can be enabled

concurrently with the basic format. The maximum for *<no_channels>* is 4096. 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

Before you start: You need root authority to enable 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>*/cmb 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>*/cmb 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, cmb, 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>/cmb/<attribute>
```

where /sys/bus/ccw/devices/*<device-bus-id>* is the directory that represents the device, and *<attribute>* the attribute to be read. Table 31 on page 233 summarizes the available attributes.

Table 31. Attributes with collected I/O data

Attribute	Value
ssch_rsch	An integer representing the ssch rsch count 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.

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/avg_device_busy_time
21
```

Chapter 22. Control program identification

This section applies to Linux instances in LPAR mode only.

If your Linux instance runs in LPAR mode, you can use the control program identification (CPI) module, sclp_cpi, to assign a name to your Linux instance. The system name is used, for example, to identify the Linux instance on the HMC.

Building a kernel with CPI support

This section is intended for those who want to build their own kernel. It describes the options you must select in the Linux configuration menu to include the CPI support.

Figure 50 summarizes the kernel configuration menu options that are relevant to the CPI support:

```
Character device drivers
    Support for SCLP                              (CONFIG_SCLP)
     └─Control-Program Identification            (CONFIG_SCLP_CPI)
```

Figure 50. CPI kernel configuration menu options

CONFIG_SCLP
> This option includes support for the IBM SCLP interface to the service element. It is required for CPI.

CONFIG_SCLP_CPI
> This option allows control program identification through the SCLP interface. Compile it as a separate module, sclp_cpi.

Assigning a name to your Linux instance

You provide the name as a parameter when you load the CPI module.

```
┌─ CPI module parameter syntax ─────────────────────────────────┐
│                                                               │
│ ►►──┬─insmod───┬── sclp_cpi── system_name=<system>──────────►◄ │
│     └─modprobe─┘                                               │
│                                                               │
└───────────────────────────────────────────────────────────────┘
```
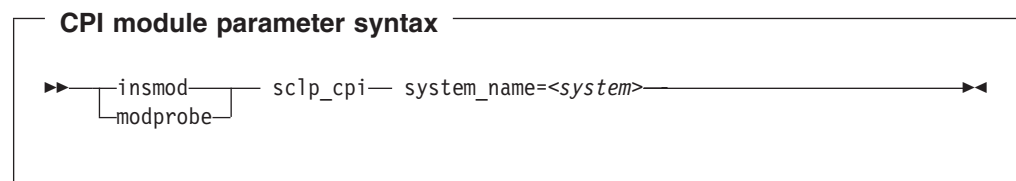
where:

<system>   is an 8-character system name. The specification is converted to uppercase.

Be sure to specify the system name correctly. You cannot change the name after you have loaded the module. A different name can be specified when loading sclp_cpi after a reboot.

Example

To assign system name "LNXA" to a Linux instance running in LPAR mode issue:

```
insmod sclp_cpi system_name=LNXA
```

Chapter 23. 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
  - Token Ring
  - ATM (running Ethernet LAN emulation)
- OSA-Express2
  - Gigabit Ethernet
  - 10 Gigabit Ethernet
  - 1000Base-T Ethernet (System z9 only)

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.

See "osasnmpd – Start OSA-Express SNMP subagent" on page 301 for information on the **osasnmpd** command itself.

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
- One of:
  - net-snmp package 5.1.x or higher
  - ucd-snmp package 4.2.x (recommended 4.2.3 or higher)

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:

```
http://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 239), but you do not need to update the subagent. Place the updated MIB file in a directory that is searched by the master agent.
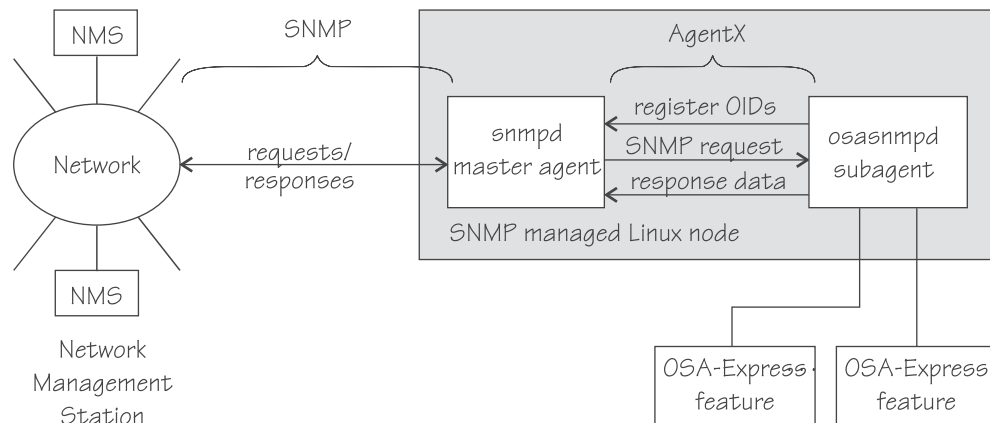
Figure 51. OSA-Express SNMP agent flow

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 `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 241).

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

`http://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 your distribution documentation to find out where you can find a template for snmpd.conf and where you need to place it. Some of the possible locations are:
   - `/usr/local/share/snmp`
   - `/etc/snmp`
   - `/usr/share/snmp`

2. Open snmpd.conf with your preferred text editor.

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 240.

<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 239.

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 240.

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

You start the osasnmpd subagent using the **osasnmpd** command:

```
# osasnmpd
```

The osasnmpd subagent starts a daemon called osasnmpd-2.6.

For command options see "osasnmpd – Start OSA-Express SNMP subagent" on
page 301.

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 practise 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 295) 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 239.

If you have provided defaults for the SNMP version and the community (see step 12 on page 242), 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 (see 301). Otherwise you can issue a **ps** command to find it out.

Example: The osasnmpd subagent starts a daemon called osasnmpd-2.6. 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-2.6
# killall osasnmpd-2.6
# kill 729
```

Chapter 24. IPL parameter interface

Use the IPL parameter sysfs interface to derive the IPL device. You can use this information to set the IPL device online and later use it, for example, for installation purposes.

The IPL parameter interface is always part of the kernel and is not configurable.

Working with the IPL parameter interface

For the IPL parameters, a sysfs userspace 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/.

Example

If the device is CCW, the additional file `device` will be present. It contains the bus ID of the IPL device, for example:

```
# cat /sys/firmware/ipl/device
0.0.1234
```

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 operating system specific parameters.

binary_parameter
>Contains all of the above information in binary format.

Setting the IPL device online

You can set the IPL device online using chccwdev, for example:

```
# chccwdev -e 0.0.50dc
```

Chapter 25. 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. The feature also allows you to 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. This has a negative performance impact if you enable the feature through kernel parameter. Selecting the config options will not have this negative effect.

Building a kernel with the data execution protection feature

This section is intended for those who want to build their own kernel. It describes the options you must select in the Linux configuration menu to include the data execution protection feature.

You need to select the kernel configuration option S390_SWITCH_AMODE and S390_EXEC_PROTECT to enable the execute-protection option.

Note: This feature will only be active if the kernel parameter `noexec` is also specified. Otherwise, the config options will have no effect.

```
Base setup
   Switch kernel/user addressing modes      (S390_SWITCH_AMODE)
   Data execution protection                (S390_EXEC_PROTECT)
```

The S390_SWITCH_AMODE config option will be automatically selected if you chose the S390_EXEC_PROTECT option. If you only select S390_SWITCH_AMODE, you also need to specify the kernel parameter `switch_amode` to switch the addressing modes. Otherwise the config option will have no effect.

Setting up the data execution protection feature

To enable the data execution protection, configure the kernel with the S390_EXEC_PROTECT config option and add the kernel parameter `noexec` to your parmfile or zipl.conf. This will also switch the addressing modes of kernel and user space. Specifying `noexec=off` or no parameter at all will disable the feature (this is the default).

A kernel message will indicate the status of the execute protection at boot time, for example like this (without z9-109 EC or BC hardware support it will say "mvcos not available"):

```
...
We are running under VM (64 bit mode)
S390 execute protection active, mvcos available
Detected 4 CPUs
...
```

To enable only the addressing mode switch, configure the kernel with the S390_SWITCH_AMODE config option and add the kernel parameter `switch_amode` to your parmfile or zipl.conf. A kernel message will indicate the status of the addressing mode switch at boot time, for example like this (with z9-109 EC/BC hardware support it will say "mvcos available"):

```
...
We are running under VM (64 bit mode)
S390 address spaces switched, mvcos not available
Detected 4 CPUs
...
```

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).

Example

Set and query the executable stack flag (stack is executable):

```
$ execstack -s /usr/bin/find
$ execstack -q /usr/bin/find
X /usr/bin/find
```

> **Example**
>
> 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.

> **Example**
>
> 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
> ```

> **Example**
>
> 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
> ```

Chapter 26. Other features kernel builders should know about

This section is intended for those who want to build their own kernel. It summarizes the System z-specific kernel configuration options, including those options that do not correspond to a particular device driver or feature.

The options described in this section are sorted into two groups:
- "General architecture-specific options"
- "Device driver-related options" on page 258

For each group there is an overview of the options in the order in which you find them in the kernel configuration menu (see Figure 52 on page 252 and Figure 54 on page 259). Each overview is followed by an alphabetically sorted list of the options with a description.

Dependencies between options

Simple dependencies, where an option depends on another option that directly precedes it in the configuration menu, are shown in the overviews (Figure 52 on page 252 and Figure 54 on page 259). The dependent option is shown indented and graphically joined (└) to the option it depends on. Options that have more complex dependencies are marked with an asterisk (*).

The option descriptions that follow the overviews include more detailed information on the dependencies. This more detailed information is provided in boolean format as it appears in the Kconfig files in the Linux source tree, with the CONFIG_ prefix omitted.

Common code options are not included in this summary. Refer to the Linux source tree for descriptions of common code options. To locate the description of an option in the Linux source tree, open a command prompt and change the working current directory at the root of the Linux source tree. Issue a command of this form:

```
# grep -rl --include='Kconfig' '^config <OPTION>' *
```

where *<option>* is the option you are looking for.

Note: In the Kconfig files, the options do not have a the CONFIG_ prefix. Be sure to omit the CONFIG_ when searching for the option.

Example: To locate the Kconfig file with the description of the common code kernel configuration option CONFIG_EXPERIMENTAL issue:

```
# grep -rl --include='Kconfig' '^config EXPERIMENTAL' *
init/Kconfig
```

General architecture-specific options

Figure 52 on page 252 provides an overview of the general architecture-specific options in the order in which you find them in the kernel configuration menu. The following pages provide explanations for each option in alphabetical order. For device driver-specific options see "Device driver-related options" on page 258.

```
                                                          (CONFIG_S390)
   ...
Base setup --->
   --- Processor type and features ---
   Symmetric multi-processing support                     (CONFIG_SMP)
   ├─ Maximum number of CPUs (2-64)                        (CONFIG_NR_CPUS)
   └─ Support for hot-pluggable CPUs                       (CONFIG_HOTPLUG_CPU)
                                                          (CONFIG_DEFAULT_MIGRATION_COST)

   IEEE FPU emulation
   Kernel support for 31 bit ELF binaries                 (CONFIG_BINFMT_ELF32)
   --- Code generation options ---
   Processor type (selection)
         • S/390 model G5 and G6                           (CONFIG_MARCH_G5)
         • IBM eServer zSeries model z800 and z900         (CONFIG_MARCH_Z900)
         • IBM eServer zSeries model z890 and z990         (CONFIG_MARCH_Z990)
   Pack kernel stack                                       (CONFIG_PACK_STACK)
   └─ Use 4kb/8kb for kernel stack instead of 8kb/16kb     (CONFIG_SMALL_STACK)
   Detect kernel stack overflow                            (CONFIG_CHECK_STACK)
   └─ Size of the guard area (128-1024)                    (CONFIG_STACK_GUARD)
   Emit compiler warnings for function with broken stack usage  (CONFIG_WARN_STACK)
   └─ Maximum frame size considered safe (128-2048)        (CONFIG_WARN_STACK_SIZE)

   ...
   --- I/O subsystem configuration ---
   Process warning machine checks                          (CONFIG_MACHCHK_WARNING)
   QDIO support                                            (CONFIG_QDIO)
   ├─ Performance statistics in /proc                      (CONFIG_QDIO_PERF_STATS)
   └─ Extended debugging information                       (CONFIG_QDIO_DEBUG)
   --- Misc ---
   Preemptible Kernel                                      (CONFIG_PREEMPT)
   Builtin IPL record support                              (CONFIG_IPL)
   └─ IPL method generated into head.S (selection)
         • tape                                            (CONFIG_IPL_TAPE)
         • vm_reader                                       (CONFIG_IPL_VM)
   ...
   Show crashed user process info                          (CONFIG_PROCESS_DEBUG)
   Pseudo page fault support                               (CONFIG_PFAULT)
   VM shared kernel support                                (CONFIG_SHARED_KERNEL)
   Cooperative memory management                           (CONFIG_CMM)
   └─ /proc interface to cooperative memory management     (CONFIG_CMM_PROC)
   └─ IUCV special message interface to cooperative memory management  (CONFIG_CMM_IUCV)*
   Virtual CPU timer support                               (CONFIG_VIRT_TIMER)
   └─ Base user process accounting on virtual cpu timer    (CONFIG_VIRT_CPU_ACCOUNTING)
   └─ Linux - VM Monitor Stream, base infrastructure       (CONFIG_APPLDATA_BASE)*
         ├─ Monitor memory management statistics           (CONFIG_APPLDATA_MEM)
         ├─ Monitor OS statistics                          (CONFIG_APPLDATA_OS)
         └─ Monitor overall network statistics             (CONFIG_APPLDATA_NET_SUM)
   No HZ timer ticks in idle                               (CONFIG_NO_IDLE_HZ)
   └─ HZ timer in idle off by default                      (CONFIG_NO_IDLE_HZ_INIT)
   s390 hypervisor file system support                     (CONFIG_S390_HYPFS_FS)
   ...
```

Figure 52. General architecture-specific kernel configuration menu options. The └─ symbols indicate dependencies on preceding options. Options with more complex dependencies are marked with an asterisk (*).

```
    Instrumentation Support --->
        ...
        Profiling support                                       (CONFIG_PROFILING)
        └ OProfile system profiling                             (CONFIG_OPROFILE)
|       Statisics infrastructure                                (CONFIG_STATISTICS)*
        ...
    Cryptographic options  --->
        ...
        SHA1 digest algorithm (s390)                            (CONFIG_CRYPTO_SHA1_S390)
        ...
        SHA256 digest algorithm (s390)                          (CONFIG_CRYPTO_SHA256_S390)
        ...
        DES and Triple DES cipher algorithms (s390)            (CONFIG_CRYPTO_DES_S390)
        ...
        AES cipher algorithms (s390)                            (CONFIG_CRYPTO_AES_S390)
```

Figure 53. General architecture-specific kernel configuration menu options 2 of 2. The └ symbols indicate dependencies on preceding options. Options with more complex dependencies are marked with an asterisk (*).

The following is an alphabetically sorted list with details on the general architecture-specific options summarized in Figure 52 on page 252. For device driver specific options see "Device driver-related options" on page 258.

CONFIG_APPLDATA_BASE

This provides a kernel interface for creating and updating z/VM APPLDATA monitor records. The monitor records are updated at certain time intervals, once the timer is started. Writing 1 or 0 to /proc/appldata/timer starts(1) or stops(0) the timer, i.e. enables or disables monitoring on the Linux side. A custom interval value (in seconds) can be written to /proc/appldata/interval.

Defaults are 60 seconds interval and timer off. The /proc entries can also be read from, showing the current settings.

Depends on PROC_FS && VIRT_TIMER=y.

PROC_FS is a common code option.

CONFIG_APPLDATA_MEM

This provides memory management related data to the Linux - VM Monitor Stream, like paging/swapping rate, memory utilisation, etc. Writing 1 or 0 to /proc/appldata/memory creates(1) or removes(0) a z/VM APPLDATA monitor record, i.e. enables or disables monitoring this record on the z/VM side.

Default is disabled. The /proc entry can also be read from, showing the current settings.

This can also be compiled as a module, which will be called appldata_mem.o.

Depends on APPLDATA_BASE.

CONFIG_APPLDATA_NET_SUM

This provides network related data to the Linux - VM Monitor Stream, currently there is only a total sum of network I/O statistics, no per-interface data. Writing 1 or 0 to /proc/appldata/net_sum creates(1) or removes(0) a z/VM APPLDATA monitor record, i.e. enables or disables monitoring this record on the z/VM side.

Default is disabled. This can also be compiled as a module, which will be called appldata_net_sum.o.

Depends on APPLDATA_BASE.

CONFIG_APPLDATA_OS

This provides OS related data to the Linux - VM Monitor Stream, like CPU utilisation, etc. Writing 1 or 0 to /proc/appldata/os creates(1) or removes(0) a z/VM APPLDATA monitor record, i.e. enables or disables monitoring this record on the z/VM side.

Default is disabled. This can also be compiled as a module, which will be called appldata_os.o.

Depends on APPLDATA_BASE.

CONFIG_BINFMT_ELF32

This allows you to run 32-bit Linux/ELF binaries on your zSeries in 64 bit mode. Everybody wants this; say Y.

CONFIG_CHECK_STACK

This option enables the compiler option -mstack-guard and -mstack-size if they are available. If the compiler supports them it will emit additional code to each function prolog to trigger an illegal operation if the kernel stack is about to overflow.

Say N if you are unsure.

CONFIG_CMM

Select this option, if you want to enable the kernel interface to reduce the memory size of the system. This is accomplished by allocating pages of memory and put them ″on hold″. This only makes sense for a system running under VM where the unused pages will be reused by VM for other guest systems. The interface allows an external monitor to balance memory of many systems. Everybody who wants to run Linux under VM should select this option.

CONFIG_CMM_IUCV

Select this option to enable the special message interface to the cooperative memory management.

Depends on CMM && (SMSGIUCV=y || CMM=SMSGIUCV).

CONFIG_CMM_PROC

Select this option to enable the /proc interface to the cooperative memory management.

Depends on CMM.

CONFIG_CRYPTO_AES_S390

This is the s390 hardware accelerated implementation of the AES cipher algorithms (FIPS-197). AES uses the Rijndael algorithm.

Rijndael appears to be consistently a very good performer in both hardware and software across a wide range of computing environments regardless of its use in feedback or non-feedback modes. Its key setup time is excellent, and its key agility is good. Rijndael's very low memory requirements make it very well suited for restricted-space environments, in which it also demonstrates excellent performance. Rijndael's operations are among the easiest to defend against power and timing attacks.

On s390 the System z9-109 currently only supports the key size of 128 bit.

Depends on CRYPTO && S390.

CRYPTO is a common code option.

CONFIG_CRYPTO_DES_S390

DES cipher algorithm (FIPS 46-2), and Triple DES EDE (FIPS 46-3).

Depends on CRYPTO && S390.

CRYPTO is a common code option.

CONFIG_CRYPTO_SHA1_S390
This is the s390 hardware accelerated implementation of the SHA-1 secure hash standard (FIPS 180-1/DFIPS 180-2).

Depends on CRYPTO && S390.

CRYPTO is a common code option.

CONFIG_CRYPTO_SHA256_S390
This is the s390 hardware accelerated implementation of the SHA256 secure hash standard (DFIPS 180-2).

This version of SHA implements a 256 bit hash with 128 bits of security against collision attacks.

Depends on CRYPTO && S390.

CRYPTO is a common code option.

CONFIG_DEFAULT_MIGRATION_COST
No helptext found.

CONFIG_HOTPLUG_CPU
Say Y here to be able to turn CPUs off and on. CPUs can be controlled through /sys/devices/system/cpu/cpu#. Say N if you want to disable CPU hotplug.

Depends on SMP.

CONFIG_IPL
If you want to use the produced kernel to IPL directly from a device, you have to merge a bootsector specific to the device into the first bytes of the kernel. You will have to select the IPL device.

CONFIG_IPL_TAPE | CONFIG_IPL_VM
Select ″tape″ if you want to IPL the image from a Tape.

Select ″vm_reader″ if you are running under VM/ESA and want to IPL the image from the emulated card reader.

Depends on IPL.

CONFIG_MACHCHK_WARNING
Select this option if you want the machine check handler on IBM S/390 or zSeries to process warning machine checks (e.g. on power failures). If unsure, say ″Y″.

CONFIG_MARCH_G5 | CONFIG_MARCH_Z900 | CONFIG_MARCH_Z990
Select MARCH_G5 to build a 31 bit kernel that works on all S/390 and zSeries machines. MARCH_G5 is not available if option 64BIT is selected.

Select MARCH_Z900 to optimize for zSeries machines. This will enable some optimizations that are not available on older 31 bit only CPUs.

Select MARCH_Z990 to enable optimizations for model z890/z990. This will be slightly faster but does not work on older machines such as the z900.

CONFIG_NO_IDLE_HZ
Switches the regular HZ timer off when the system is going idle. This helps z/VM to detect that the Linux system is idle. VM can then ″swap-out″ this guest which reduces memory usage. It also reduces the overhead of idle systems.

The HZ timer can be switched on/off via /proc/sys/kernel/hz_timer. hz_timer=0 means HZ timer is disabled. hz_timer=1 means HZ timer is active.

CONFIG_NO_IDLE_HZ_INIT

The HZ timer is switched off in idle by default. That means the HZ timer is already disabled at boot time.

Depends on NO_IDLE_HZ.

CONFIG_NR_CPUS

This allows you to specify the maximum number of CPUs which this kernel will support. The maximum supported value is 64 and the minimum value which makes sense is 2.

This is purely to save memory - each supported CPU adds approximately sixteen kilobytes to the kernel image.

Depends on SMP.

CONFIG_OPROFILE

OProfile is a profiling system capable of profiling the whole system, include the kernel, kernel modules, libraries, and applications.

If unsure, say N.

Depends on PROFILING.

CONFIG_PACK_STACK

This option enables the compiler option -mkernel-backchain if it is available. If the option is available the compiler supports the new stack layout which dramatically reduces the minimum stack frame size. With an old compiler a non-leaf function needs a minimum of 96 bytes on 31 bit and 160 bytes on 64 bit. With -mkernel-backchain the minimum size drops to 16 byte on 31 bit and 24 byte on 64 bit.

Say Y if you are unsure.

CONFIG_PFAULT

Select this option, if you want to use PFAULT pseudo page fault handling under VM. If running native or in LPAR, this option has no effect. If your VM does not support PFAULT, PAGEEX pseudo page fault handling will be used. Note that VM 4.2 supports PFAULT but has a bug in its implementation that causes some problems. Everybody who wants to run Linux under VM != VM4.2 should select this option.

CONFIG_PREEMPT

This option reduces the latency of the kernel when reacting to real-time or interactive events by allowing a low priority process to be preempted even if it is in kernel mode executing a system call. This allows applications to run more reliably even when the system is under load.

Say N if you are unsure.

CONFIG_PROCESS_DEBUG

Say Y to print all process fault locations to the console. This is a debugging option; you probably do not want to set it unless you are an S390 port maintainer.

CONFIG_PROFILING

Say Y here to enable profiling support mechanisms used by profilers such as readprofile or OProfile.

CONFIG_STATISTICS

The statistics infrastructure provides a debug-fs based user interface for statistics of kernel components, that is, usually device drivers. Statistics are available for components that have been instrumented to feed data into the statistics infrastructure. This feature is useful for performance measurements or performance debugging. If in doubt, say ″N″.

Depends on the common code option DEBUG_FS.

CONFIG_QDIO

This driver provides the Queued Direct I/O base support for IBM mainframes.

For details please refer to the documentation provided by IBM at

`http://www.software.ibm.com/developerworks/opensource/linux390`

To compile this driver as a module, choose M here: the module will be called qdio.

If unsure, say Y.

CONFIG_QDIO_DEBUG

Say Y here to get extended debugging output in /sys/kernel/debug/s390dbf/ qdio... Warning: this option reduces the performance of the QDIO module.

If unsure, say N.

Depends on QDIO.

CONFIG_QDIO_PERF_STATS

Say Y here to get performance statistics in /proc/qdio_perf

If unsure, say N.

Depends on QDIO.

CONFIG_S390_HYPFS_FS

This is a virtual file system intended to provide accounting information in an S/390 hypervisor environment.

CONFIG_SHARED_KERNEL

Select this option, if you want to share the text segment of the Linux kernel between different VM guests. This reduces memory usage with lots of guests but greatly increases kernel size. You should only select this option if you know what you are doing and want to exploit this feature.

CONFIG_SMALL_STACK

If you say Y here and the compiler supports the -mkernel-backchain option the kernel will use a smaller kernel stack size. For 31 bit the reduced size is 4kb instead of 8kb and for 64 bit it is 8kb instead of 16kb. This allows to run more thread on a system and reduces the pressure on the memory management for higher order page allocations.

Say N if you are unsure.

Depends on PACK_STACK.

CONFIG_SMP

This enables support for systems with more than one CPU. If you have a system with only one CPU, like most personal computers, say N. If you have a system with more than one CPU, say Y.

If you say N here, the kernel will run on single and multiprocessor machines, but will use only one CPU of a multiprocessor machine. If you

say Y here, the kernel will run on many, but not all, singleprocessor machines. On a singleprocessor machine, the kernel will run faster if you say N here.

See also the <file:Documentation/smp.txt> and the SMP-HOWTO available at <http://www.tldp.org/docs.html#howto>.

Even if you don't know what to do here, say Y.

CONFIG_STACK_GUARD

This allows you to specify the size of the guard area at the lower end of the kernel stack. If the kernel stack points into the guard area on function entry an illegal operation is triggered. The size needs to be a power of 2. Please keep in mind that the size of an interrupt frame is 184 bytes for 31 bit and 328 bytes on 64 bit. The minimum size for the stack guard should be 256 for 31 bit and 512 for 64 bit.

Depends on CHECK_STACK.

CONFIG_VIRT_CPU_ACCOUNTING

Select this option to use CPU timer deltas to do user process accounting.

Depends on VIRT_TIMER.

CONFIG_VIRT_TIMER

This provides a kernel interface for virtual CPU timers. Default is disabled.

CONFIG_WARN_STACK

This option enables the compiler options -mwarn-framesize and -mwarn-dynamicstack. If the compiler supports these options it will generate warnings for function which either use alloca or create a stack frame bigger then CONFIG_WARN_STACK_SIZE.

Say N if you are unsure.

CONFIG_WARN_STACK_SIZE

This allows you to specify the maximum frame size a function may have without the compiler complaining about it.

Depends on WARN_STACK.

Device driver-related options

Figure 54 on page 259 provides an overview of the device driver-related options in the order in which you find them in the kernel configuration menu. The following pages provide explanations for each option in alphabetical order. For architecture-specific options see "General architecture-specific options" on page 251.

```
        --- S/390 block device drivers ---
        XPRAM disk support                                              (CONFIG_BLK_DEV_XPRAM)*
        DCSSBLK support                                                 (CONFIG_DCSSBLK)
        Support for DASD devices                                        (CONFIG_DASD)
        ├─ Profiling support for dasd devices                           (CONFIG_DASD_PROFILE)
        ├─ Support for ECKD Disks                                       (CONFIG_DASD_ECKD)
        ├─ Support for FBA  Disks                                       (CONFIG_DASD_FBA)
        ├─ Support for DIAG access to Disks                             (CONFIG_DASD_DIAG)
        ├─ Extended error reporting (EER)                               (CONFIG_DASD_EER)
        └─ Compatibility interface for DASD channel measurement blocks  (CONFIG_DASD_CMB)
        ...
  Character device drivers --->
      Watchdog Cards --->
      --- Watchdog Device Drivers ---
          ...
          z/VM Watchdog Timer                                          (CONFIG_ZVM_WATCHDOG)*
      ...
      --- S/390 character device drivers ---
      Support for locally attached 3270 terminals                     (CONFIG_TN3270)
      ├─ Support for tty input/output on 3270 terminals               (CONFIG_TN3270_TTY)
      ├─ Support for fullscreen applications on 3270 terminals         (CONFIG_TN3270_FS)
      └─ Support for console on 3270 terminal                         (CONFIG_TN3270_CONSOLE)*
      Support for 3215 line mode terminal                             (CONFIG_TN3215)
      └─ Support for console on 3215 line mode terminal               (CONFIG_TN3215_CONSOLE)
      Support for SCLP                                                (CONFIG_SCLP)
      ├─ Support for SCLP line mode terminal                          (CONFIG_SCLP_TTY)
      │  └─ Support for console on SCLP line mode terminal            (CONFIG_SCLP_CONSOLE)
      ├─ Support for SCLP VT220-compatible terminal                   (CONFIG_SCLP_VT220_TTY)
      │  └─ Support for console on SCLP VT220-compatible terminal     (CONFIG_SCLP_VT220_CONSOLE)
      └─ Control-Program Identification                               (CONFIG_SCLP_CPI)
      S/390 tape device support                                       (CONFIG_S390_TAPE)
      --- S/390 tape interface support ---
      ├─ Support for tape block devices                               (CONFIG_S390_TAPE_BLOCK)
      --- S/390 tape hardware support ---
      └─ Support for 3480/3490 tape hardware                          (CONFIG_S390_TAPE_34XX)
      Support for 3590 tape hardware                                  (CONFIG_S390_TAPE_3590)*
      Support for the z/VM recording system services (VM only)        (CONFIG_VMLOGRDR)*
      Support for the z/VM CP interface (VM only)                     (CONFIG_VMCP)
      API for reading z/VM monitor service records                    (CONFIG_MONREADER)*
      API for writing z/VM monitor service records                    (CONFIG_MONWRITER)
  Cryptographic devices --->
|     Support for PCI-attached cryptographic adapters                 (CONFIG_ZCRYPT)
|     Support for monolithic cryptographic device driver              (CONFIG_ZCRYPT_MONOLITHIC)
  Network device support --->
      ...
    S/390 network device drivers (Depends on  NETDEVICES && S390) --->
      Lan Channel Station Interface                                   (CONFIG_LCS)*
      CTC device support                                              (CONFIG_CTC)*
      IUCV support (VM only)                                          (CONFIG_IUCV)
      ├─ IUCV network device support (VM only)                        (CONFIG_NETIUCV)
      └─ IUCV special message support (VM only)                       (CONFIG_SMSGIUCV)
      CLAW device support                                             (CONFIG_CLAW)*
      MPC SNA device support                                          (CONFIG_MPC)*
      Gigabit Ethernet device support                                 (CONFIG_QETH)*
      --- Gigabit Ethernet default settings ---
      IPv6 support for gigabit ethernet                               (CONFIG_QETH_IPV6)*
      VLAN support for gigabit ethernet                               (CONFIG_QETH_VLAN)*
      Performance statistics in /proc                                 (CONFIG_QETH_PERF_STATS)*
```

Figure 54. Kernel configuration menu options. The └ *symbols indicate dependencies on preceding options. Options with more complex dependencies are marked with an asterisk (*).*

The following is an alphabetically sorted list with details on the device driver-related options summarized in Figure 54 on page 259. For architecture-specific options see "General architecture-specific options" on page 251.

CONFIG_BLK_DEV_XPRAM

Select this option if you want to use your expanded storage on S/390 or zSeries as a disk. This is useful as a _fast_ swap device if you want to access more than 2G of memory when running in 31 bit mode. This option is also available as a module which will be called xpram. If unsure, say ″N″.

Depends on S390.

CONFIG_CLAW

This driver supports channel attached CLAW devices. CLAW is Common Link Access for Workstation. Common devices that use CLAW are RS/6000s, Cisco Routers (CIP) and 3172 devices. To compile as a module choose M here: The module will be called claw.ko to compile into the kernel choose Y

Depends on the common code option NETDEVICES.

Note: The CLAW device driver is deprecated.

CONFIG_CTC

Select this option if you want to use channel-to-channel networking on IBM S/390 or zSeries. This device driver supports real CTC coupling using ESCON. It also supports virtual CTCs when running under VM. It will use the channel device configuration if this is available. This option is also available as a module which will be called ctc.ko. If you do not know what it is, it's safe to say ″Y″.

Depends on the common code option NETDEVICES.

Note: The CTC device driver is deprecated.

CONFIG_DASD

Enable this option if you want to access DASDs directly utilizing S/390s channel subsystem commands. This is necessary for running natively on a single image or an LPAR.

CONFIG_DASD_CMB

This driver provides an additional interface to the channel measurement facility, which is normally accessed though sysfs, with a set of ioctl functions specific to the dasd driver. This is only needed if you want to use applications written for linux-2.4 dasd channel measurement facility interface.

Depends on DASD.

CONFIG_DASD_DIAG

Select this option if you want to use Diagnose250 command to access disks under VM. If you are not running under VM or unsure what it is, say ″N″.

Depends on DASD.

CONFIG_DASD_ECKD

ECKD devices are the most commonly used devices. You should enable this option unless you are very sure to have no ECKD device.

Depends on DASD.

CONFIG_DASD_EER

This driver provides a character device interface to the DASD extended error reporting. This is only needed if you want to use applications written for the EER facility.

Depends on DASD.

CONFIG_DASD_FBA

Select this option to be able to access FBA devices. It is safe to say ″Y″.

Depends on DASD.

CONFIG_DASD_PROFILE

Enable this option if you want to see profiling information in /proc/dasd/statistics.

Depends on DASD.

CONFIG_DCSSBLK

Support for dcss block device

CONFIG_IUCV

Select this option if you want to use inter-user communication under VM or VIF. If unsure, say ″Y″ to enable a fast communication link between VM guests. At boot time the user ID of the guest needs to be passed to the kernel. Note that both kernels need to be compiled with this option and both need to be booted with the user ID of the other VM guest.

CONFIG_LCS

Select this option if you want to use LCS networking on IBM S/390 or zSeries. This device driver supports Token Ring (IEEE 802.5), FDDI (IEEE 802.7) and Ethernet. This option is also available as a module which will be called lcs.ko. If you do not know what it is, it's safe to say ″Y″.

Depends on NETDEVICES && (NET_ETHERNET || TR || FDDI).

NETDEVICES, NET_ETHERNET, TR, and FDDI are common code options.

CONFIG_MONREADER

Character device driver for reading z/VM monitor service records

Depends on IUCV.

CONFIG_MONWRITER

Character device driver for writing z/VM monitor service records

CONFIG_MPC

This driver supports channel-to-channel MPC SNA devices. MPC is a SNA protocol device used by Comm Server for Linux. If you don't have Comm Server for Linux you don't need the device. To compile as a module choose M here: The module will be called ctcmpc.ko to compile into the kernel choose Y If you do not need SNA MPC device just say N

Depends on the common code option NETDEVICES.

CONFIG_NETIUCV

Select this option if you want to use inter-user communication vehicle networking under VM or VIF. It enables a fast communication link between VM guests. Using ifconfig a point-to-point connection can be established to the Linux for zSeries and S7390 system running on the other VM guest. This option is also available as a module which will be called netiucv.ko. If unsure, say ″Y″.

Depends on IUCV && NETDEVICES.

NETDEVICES is a common code option.

Note: The NETIUCV device driver is deprecated.

CONFIG_QETH

This driver supports the IBM S/390 and zSeries OSA Express adapters in QDIO mode (all media types), HiperSockets interfaces and VM GuestLAN interfaces in QDIO and HIPER mode.

For details please refer to the documentation provided by IBM at

`http://www.software.ibm.com/developerworks/opensource/linux390`

To compile this driver as a module, choose M here: the module will be called qeth.ko.

Depends on NETDEVICES && IP_MULTICAST && QDIO.

NETDEVICES and IP_MULTICAST are common code options.

CONFIG_QETH_IPV6

If CONFIG_QETH is switched on, this option will include IPv6 support in the qeth device driver.

Depends on (QETH = IPV6) || (QETH && IPV6 = 'y').

IPV6 is a common code option.

CONFIG_QETH_PERF_STATS

When switched on, this option will add a file in the proc-fs (/proc/qeth_perf_stats) containing performance statistics. It may slightly impact performance, so this is only recommended for internal tuning of the device driver.

Depends on QETH.

CONFIG_QETH_VLAN

If CONFIG_QETH is switched on, this option will include IEEE 802.1q VLAN support in the qeth device driver.

Depends on (QETH = VLAN_8021Q) || (QETH && VLAN_8021Q = 'y').

VLAN_8021Q is a common code option.

CONFIG_S390_TAPE

Select this option if you want to access channel-attached tape devices on IBM S/390 or zSeries. If you select this option you will also want to select at least one of the tape interface options and one of the tape hardware options in order to access a tape device. This option is also available as a module. The module will be called tape390 and include all selected interfaces and hardware drivers.

CONFIG_S390_TAPE_34XX

Select this option if you want to access IBM 3480/3490 magnetic tape subsystems and 100% compatibles. It is safe to say ″Y″ here.

Depends on S390_TAPE.

CONFIG_S390_TAPE_3590

Select this option if you want to access IBM 3590 magnetic tape subsystems and 100% compatibles. It is safe to say ″Y″ here.

Depends on S390_TAPE.

CONFIG_S390_TAPE_BLOCK

Select this option if you want to access your channel-attached tape devices using the block device interface. This interface is similar to CD-ROM devices on other platforms. The tapes can only be accessed read-only when using this interface. Have a look at `file:Documentation/s390/TAPE` for further information about creating volumes for and using this interface. It is safe to say ″Y″ here.

Depends on S390_TAPE.

CONFIG_SCLP

Include support for the SCLP interface to the service element.

CONFIG_SCLP_CONSOLE

Include support for using an IBM HWC line-mode terminal as the Linux system console.

Depends on SCLP_TTY.

CONFIG_SCLP_CPI

This option enables the hardware console interface for system identification. This is commonly used for workload management and gives you a nice name for the system on the service element. Please select this option as a module since built-in operation is completely untested. You should only select this option if you know what you are doing, need this feature and intend to run your kernel in LPAR.

Depends on SCLP.

CONFIG_SCLP_TTY

Include support for IBM SCLP line-mode terminals.

Depends on SCLP.

CONFIG_SCLP_VT220_CONSOLE

Include support for using an IBM SCLP VT220-compatible terminal as a Linux system console.

Depends on SCLP_VT220_TTY.

CONFIG_SCLP_VT220_TTY

Include support for an IBM SCLP VT220-compatible terminal.

Depends on SCLP.

CONFIG_SMSGIUCV

Select this option if you want to be able to receive SMSG messages from other VM guest systems.

Depends on IUCV.

CONFIG_TN3215

Include support for IBM 3215 line-mode terminals.

CONFIG_TN3215_CONSOLE

Include support for using an IBM 3215 line-mode terminal as a Linux system console.

Depends on TN3215.

CONFIG_TN3270

Include support for IBM 3270 terminals.

CONFIG_TN3270_CONSOLE

Include support for using an IBM 3270 terminal as a Linux system console. Available only if 3270 support is compiled in statically.

Depends on TN3270=y && TN3270_TTY=y.

CONFIG_TN3270_FS

Include support for fullscreen applications on an IBM 3270 terminal.

Depends on TN3270.

CONFIG_TN3270_TTY

Include support for using an IBM 3270 terminal as a Linux tty.

Depends on TN3270.

CONFIG_VMCP

Select this option if you want to be able to interact with the control program on z/VM

CONFIG_VMLOGRDR

Select this option if you want to be able to receive records collected by the z/VM recording system services, eg. from *LOGREC, *ACCOUNT or *SYMPTOM. This driver depends on the IUCV support driver.

Depends on IUCV.

CONFIG_ZVM_WATCHDOG

IBM S/390 and zSeries machines running under z/VM 5.1 or later provide a virtual watchdog timer to their guest that cause a user define Control Program command to be executed after a timeout.

To compile this driver as a module, choose M here. The module will be called vmwatchdog.

Depends on WATCHDOG && S390.

WATCHDOG is a common code option.

CONFIG_ZCRYPT

Select this option if you want to use a PCI-attached cryptographic adapter like the PCI Cryptographic Accelerator (PCICA) or the PCI Cryptographic Coprocessor (PCICC).

CONFIG_ZCRYPT_MONOLITHIC

Select this option if you want to build a monolithic cryptographic kernel module. The monolithic kernel module is called **z90crypt.ko**.

Part 6. Commands and kernel parameters

This part describes commands for configuring and booting Linux on System z. It also describes kernel parameters that are not specific to a particular device driver. Device driver-specific kernel parameters are described in the "Setting up" section of the respective device driver chapter.

> **Note**
>
> For prerequisites and restrictions refer to the kernel 2.6 October 2005 stream pages on developerWorks at:
>
> `ibm.com/developerworks/linux/linux390/october2005_recommended.html`

Chapter 27. Useful Linux commands

This chapter describes commands to configure and work with the Linux on System z device drivers and features.

- chccwdev
- dasdfmt
- dasdview
- fdasd
- lscss
- lsdasd
- lsqeth
- lstape
- lszfcp
- osasnmpd
- qetharp
- qethconf
- san_disc
- scsi_logging_level
- snipl
- tape390_display
- tunedasd
- vmcp
- zipl

You can obtain these commands on developerWorks at:

`ibm.com/developerworks/linux/linux390/index.shtml.`

snipl is provided as a separate package under "Useful add-ons".

san_disc is provided with the zfcp HBA API library.

All other commands are included in the s390-tools package for the Linux 2.6 October 2005 stream.

Note: For tools related to taking and analyzing system dumps, see *Linux on System z9 and zSeries Using the Dump Tools*.

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:

chccwdev

```
  ┌─ Common command options ───────────────────────────────────────────┐
  │                                                                     │
  ►►──<command>──┬─┤ Other command options ├──┬──────────────────────►◄ │
  │              ├──-h──────────────────────┤ │                         │
  │              ├──--help──────────────────┤ │                         │
  │              └──--version───────────────┘ │                         │
  │                                                                     │
  └─────────────────────────────────────────────────────────────────────┘
```
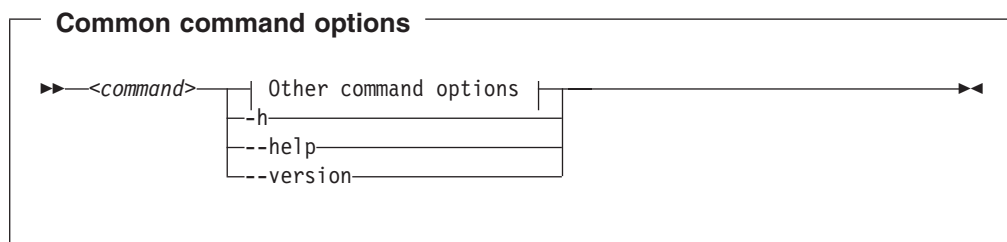
where command can be any of the commands described in this section.

See "Understanding syntax diagrams" on page xix for general information on reading syntax diagrams.

chccwdev - Set a CCW device online

Scenario

This command is used to set CCW devices (See "Device categories" on page 9) online or offline.

Format

```
  chccwdev syntax

  ►►──chccwdev──┬── -e ────────────────┬──┬─<device_bus_id>──────────────────────┬──►◄
               ├── -d ────────────────┤  └─<from_device_bus_id>-<to_device_bus_id>─┘
               ├── -f ────────────────┤
               └── -a <name> = <value>─┘
```

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
```

dasdfmt - Format a DASD

Scenario

This tool is used to give a low-level format to ECKD-type direct access storage devices (DASD). Note that this is a software format. To give a hardware format to raw DASD you must use another System z9 or zSeries device support facility such as ICKDSF, either in stand-alone mode or through another operating system.

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.

Before you start: You must have root permissions.

Format

```
 dasdfmt syntax

                            ┌── -d cdl ──┐        ┌── -l (default) ──┐ (1)
►►──dasdfmt── -b <blocksize>┤            ├────────┤ -l <volser> ─────├──►
                            └── -d ldl ──┘        └── -k ────────────┘
                                    └─ -L ─┘

►──┬──────────────────────────┬──┬── -y ──┬─┬── -F ──┬─┬── -v ──┬─┬── -t ──┬── <node> ──►◄
   └─ -p ─────────────────────┘
   └─ -m ─┬── 10 ────────┬─────┘
          └─ <hashstep> ─┘
```

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.
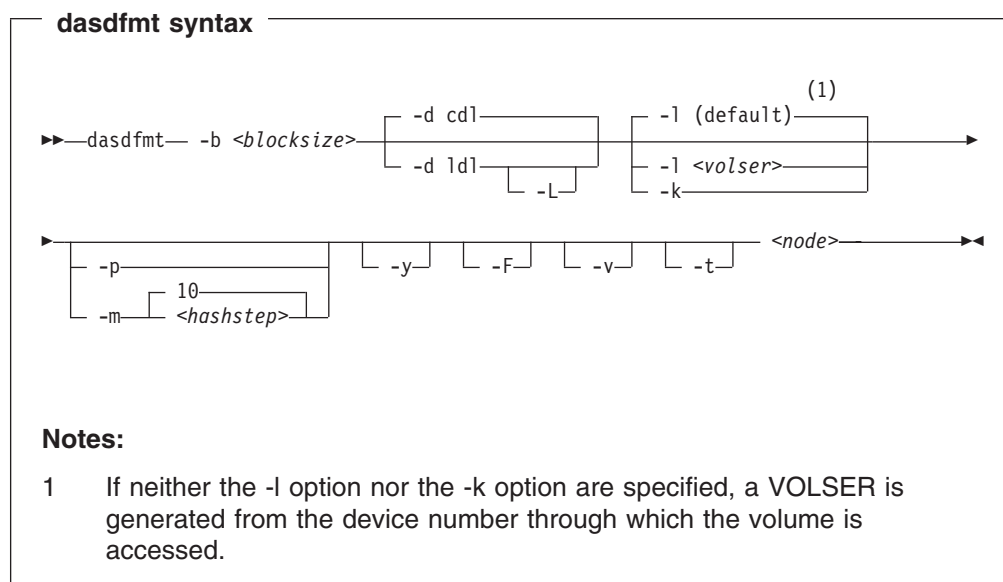
Where:

-b *<block_size>* or **--blocksize=***<block_size>*
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 30 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).

-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 27) 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_serial**

> Keeps the volume serial number when writing the volume 5 Label (see "VOLSER" on page 27). This is useful, for example, if the volume serial number has been written with a VM tool and should not be overwritten.

-p or **--progressbar**

> Prints 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).

-m *<hashstep>* or **--hashmarks=***<hashstep>*

> Prints 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   Prints out extra information messages.

-t or **--test**

> Runs the command in test mode. Analyzes parameters and prints out what would happen, but does not modify the disk.

-V or **--version**

> Prints the version number of **dasdfmt** and exits.

-h or **--help**

> Prints out an overview of the syntax. Any other parameters are ignored.

Examples

- To format a 100 cylinder 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

Scenario

dasdview displays this DASD information on the system console:

- The volume label.
- VTOC details (general information, and FMT1, FMT4, FMT5 and FMT7 labels).
- The content of the DASD, by specifying:
  - Starting point
  - Size

  You can display these values in hexadecimal, EBCDIC, and ASCII format.

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.)

Before you start: You need root permissions.

Format
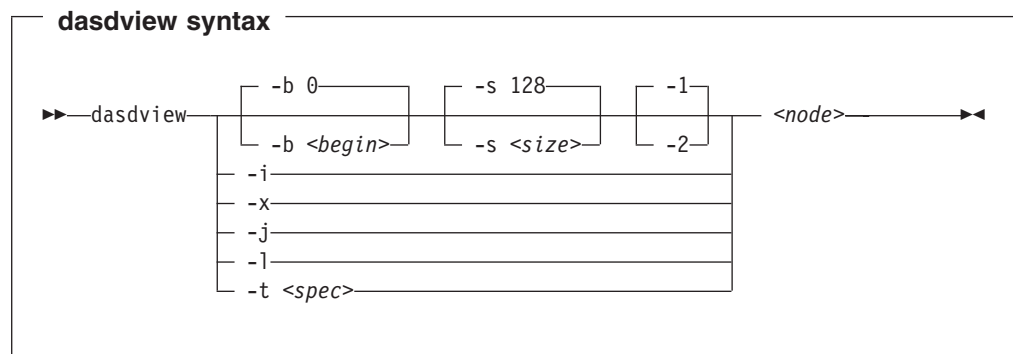
Where:

-b *<begin>* or **--begin=***<begin>*

> Display 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>*

Display 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   Display 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   Display 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**

Display basic information such as device node, device bus-id, device type, or geometry data.

-x or **--extended**

Display the information obtained by using **-i** option, but also open count, subchannel identifier, and so on.

-j   Print volume serial number (volume identifier).

-l or **--label**

Display the volume label.

-t *<spec>* or **--vtoc=***<spec>*

Display the VTOC's table-of-contents, or a single VTOC entry, on the console. The variable *<spec>* can take these values:

info   Display overview information about the VTOC, such as a list of the data set names and their sizes.

f1   Display the contents of all *format 1* data set control blocks (DSCBs).

f4   Display the contents of all *format 4* DSCBs.

f5   Display the contents of all *format 5* DSCBs.

f7   Display the contents of all *format 7* DSCBs.

all   Display the contents of *all* DSCBs.

<node>

Specifies the device node of the device for which you want to display

dasdview

information, for example, `/dev/dasdzzz`. See "DASD naming scheme" on page 30 for more details on device nodes).

-h or **--help**
Display short usage text on console. To view the man page, enter **man dasdview**.

-v or **--version**
Display version number on console, and exit.

Examples

- To display basic information about a DASD:

```
# dasdview -i -f /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 include extended information:

```
# dasdview -x -f /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 -f /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
#
```

dasdview

- To display partition information:

```
# dasdview -t info -f /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 -f /dev/dasdzzz
```

This displays:

```
--- VTOC format 4 label -------------------------------------------------------
DS4KEYCD    : 040404040404040404040404040404040404040404040404040404040404040404...
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 01000000000100000001
              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
#
```

dasdview

- To print the contents of a disk to the console starting at block 2 (volume label):

```
# dasdview -b 2b -s 128 -f /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 -f /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 -f /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 -f /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

Scenario

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 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:
- You must have root permissions.
- 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─────────────────────────────────┬───────┤
    │                              <node>─┘
    │          (1)
    ├──-a──┬──-k──────────────┬──┤
    │      └──-l <volser>──────┘
    ├──-c <conf_file>─────────┤
    ├──-i─────────────────────┤
    └──-p─────────────────────┘

 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**
  Prints additional messages that are normally suppressed.

-a or **--auto**
  Auto-create 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 27). This is useful, for example, if the volume serial number has been written with a VM tool and should not be overwritten.

-l *<volser>* or **--label=***<volser>*
  Specifies the volume serial number (see "VOLSER" on page 27).

  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 (OS/390).

  Omitting this parameter causes **fdasd** to prompt for it, if it is needed.

-c *<conf_file>* or **--config** *<conf_file>*
  This option enables you to create 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**
> Prints the volume serial number and exits.

-p or **--table**
> Prints the partition table and exits.

<node>
> Is the device node of the DASD you want to partition, for example, /dev/dasdzzz. See "DASD naming scheme" on page 30 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 = filesystem, 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 27 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 VM minidisk, change the type of one of the partitions, save the changes and check the results.

In this example, we will format a 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
```

lscss - List subchannels

Scenario

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>─┘
```
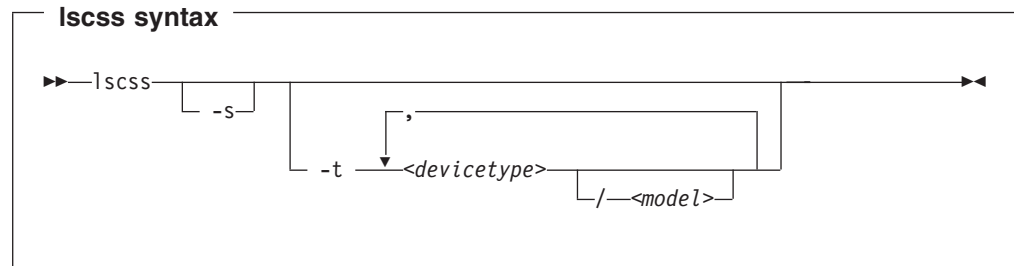
Where:

-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
```

```
# lscss -s -t 3480/04,3590
```

lsdasd - List DASD devices

Scenario

This command is used to gather information on DASD devices from sysfs and display it in a summary format.

Format

```
 lsdasd syntax
┌─────────────────────────────────────────────────────────────────┐
│  ►►──lsdasd──────┬────┬──────┬────┬──────┬────┬──────────────────────────►◄
│                  └─-a─┘      └─-s─┘      └─-v─┘      └─<device_bus_id>─┘
│
└─────────────────────────────────────────────────────────────────┘
```

Where:

-a or **--offline**
> includes devices that are currently offline.

-s or **--short**
> strips the "0.n." from the device bus IDs in the command output.

-v or **--verbose**
> prints additional messages while the command is running.

<device_bus_id>
> limits the output to information on the specified device only.

Examples

- The following command lists all DASD (the sample output shows only five):

```
# lsdasd
0.0.b104(ECKD) at ( 94: 0) is dasda : active at blocksize: 4096, 601020 blocks, 2347 MB
0.0.b105(ECKD) at ( 94: 4) is dasdb : active at blocksize: 4096, 601020 blocks, 2347 MB
0.0.b106(ECKD) at ( 94: 8) is dasdc : active at blocksize: 4096, 601020 blocks, 2347 MB
0.0.b107(ECKD) at ( 94:12) is dasdd : active at blocksize: 4096, 601020 blocks, 2347 MB
0.0.b108(ECKD) at ( 94:16) is dasde : active at blocksize: 4096, 601020 blocks, 2347 MB
```

- The following command shows information only for the DASD with device number 0xb106 and strips the "0.n." from the bus IDs in the output:

```
# lsdasd -s 0.0.b106
b106(ECKD) at ( 94: 8) is dasdc : active at blocksize: 4096, 601020 blocks, 2347 MB
```

lsqeth - List qeth based network devices

Scenario

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 305). You install **qethconf** and **lsqeth** with the same packet.

Format

```
┌── lsqeth syntax ──────────────────────────────────────────────┐
│                                                                │
│  ▶▶──lsqeth─────────────────────────────────────────────▶◀    │
│              ├─ -p ─┤   ┌─ <interface> ─┐                      │
│              └─ -c ─┘                                          │
│                                                                │
└────────────────────────────────────────────────────────────────┘
```
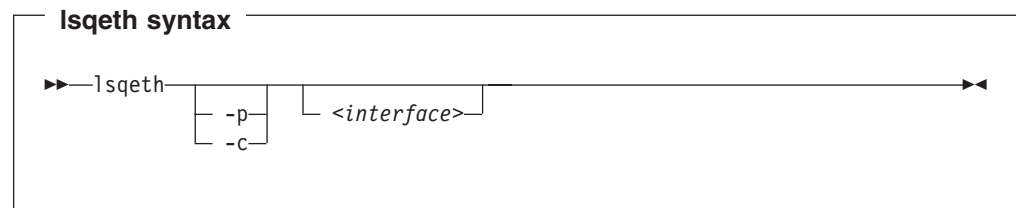
Where:

-p or **--proc**
    displays the interface information in the same format as `cat /proc/qeth`. This option can generate input to tools that expect qeth information in `/proc/qeth` format.

-c or **--ccw**
    displays the interface information in the `/etc/ccwgroup.conf` format. This option can capture a current qeth configuration in a format that conforms to the `/etc/ccwgroup.conf` syntax. If your distribution uses `/etc/ccwgroup.conf`, adding this information to it makes the configuration persistent across reboots.

<interface>
    limits the output to information on the specified interface only.

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              :
        portno                : 0
        route4                : no
        route6                : no
        checksumming          : sw checksumming
        state                 : UP (LAN ONLINE)
        priority_queueing     : always queue 2
        detach_state          : 0
        fake_ll               : 0
        fake_broadcast        : 0
        buffer_count          : 16
        add_hhlen             : 0
```

- The following command lists information on all qeth-based interfaces in /proc/qeth format:

```
# lsqeth -c
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
```

- The following command lists information on all qeth-based interfaces in /etc/ccwgroup.conf format:

```
# lsqeth -c
# Definitions for HiperSockets interface hsi0
group qeth 0.0.833f 0.0.8340 0.0.8341

# Definitions for OSD_100 interface eth0
group qeth 0.0.f5a2 0.0.f5a3 0.0.f5a4

# Definitions for OSD_100 interface eth1
group qeth 0.0.fba2 0.0.fba3 0.0.fba4
ipa_takeover/add4 10.0.0.4/24
ipa_takeover/add6 fe80:0000:0000:0000:0000:0000:0000:0001/10
```

lstape - List tape devices

Scenario

This command is used to gather information on tape devices from sysfs (see "Displaying tape information" on page 83) and display it in a summary format.

Format

```
┌─ lstape syntax ──────────────────────────────────────────────────────┐
│                                                                       │
│  ►►─lstape─┬────┬─┬───────────────────────┬─┬──────────────┬──────►   │
│           └─-s─┘ │         ┌─,─┐          │ ├─ --online ───┤        │
│                  └─-t─▼─<devicetype>─┴────┘ └─ --offline ──┘        │
│                                                                       │
│                                                                       │
│        ┌──────,───────┐                                               │
│   ►─▼──┴──────────────┴───────────────────────────────────────►◄      │
│                          (1)                                           │
│      └─<device_bus_id>─┘                                              │
│                                                                       │
│   Notes:                                                              │
│                                                                       │
│   1    specify the first device bus-ID with a leading blank.          │
│                                                                       │
└───────────────────────────────────────────────────────────────────────┘
```
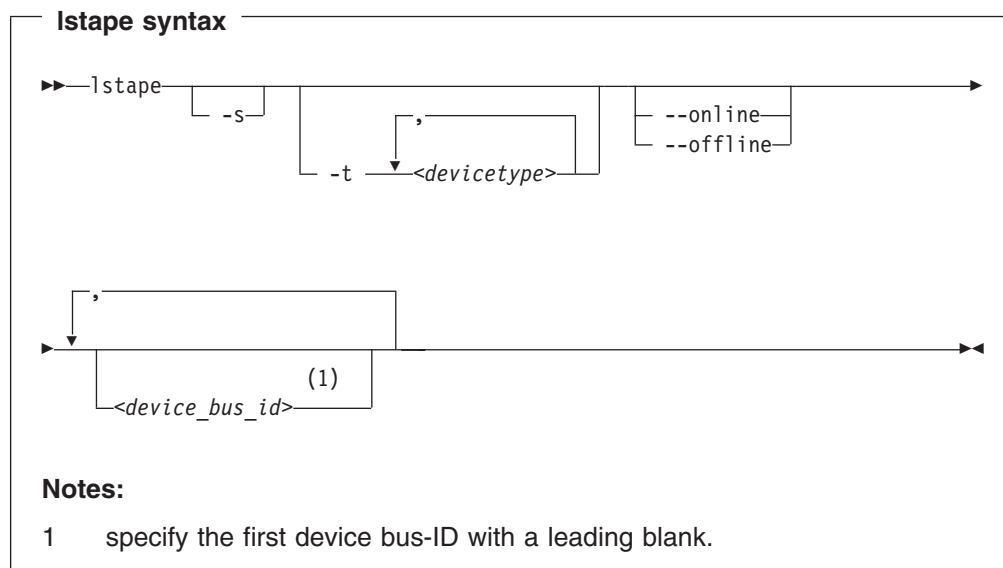
Where:

-s or **--shortid**

   strips the "0.n." from the device bus-IDs in the command output.

-t or **--type**

   limits the output to information on the specified type or types of tape devices only.

--online | --offline

   limits the output to information on online or offline tape devices only.

<device_bus_id>

   limits the output to information on the specified tape device or devices only.

-h or **--help**

   prints a short help text.

Examples

- This command displays information on all available tapes.

```
# lstape
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

```
# 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
```

lszfcp - List zfcp devices

Scenario

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

```
┌─ lszfcp syntax ─────────────────────────────────────────────┐
│                                                              │
│                 ┌──────────────────────────────────┐        │
│  ►►─ lszfcp ────┤                                   ├────►◄  │
│                 │ └─-H─┘ └─-P─┘ └─-D─┘ └─-a─┘ └─-V─┘ │        │
│                 ├─ -b ─── <device_bus_id> ──         │        │
│                 ├─ -p ─── <port_name> ──             │        │
│                 └─ -l ─── <lun_id> ──                │        │
│                                                      │        │
└──────────────────────────────────────────────────────────────┘
```
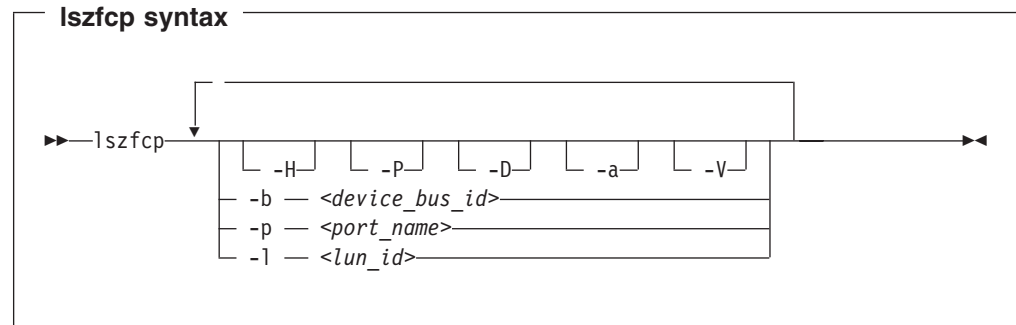
Where:

-H or **--hosts**
:   shows information about hosts.

-P or **--ports**
:   shows information about ports.

-D or **--devices**
:   shows information about SCSI 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.

-a or **--attributes**
:   shows all attributes (implies -V).

-V or **--verbose**
:   shows sysfs paths of associated class and bus devices.

-v or **--version**
:   prints version information.

-h or **--help**
:   prints a short help text.

Examples

- This command displays information on all available hosts, ports, and SCSI devices.

lszfcp

```
# 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 0.0.03d0c:

```
# 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
```

osasnmpd – Start OSA-Express SNMP subagent

Scenario

The **osasnmpd** command is used to start the OSA-Express Simple Network Management Protocol (SNMP) subagent (osasnmpd).

See Chapter 23, "OSA-Express SNMP subagent support," on page 237 for information on SNMP agent and osasnmpd subagent setup and usage.

Format

```
  osasnmpd syntax

                          ┌─ -l /var/log/osasnmpd.log ─┐
  ►►── osasnmpd ──────────┼─ -l <logfile> ─────────────┼─────────────────►
                          └─ -L ───────────────────────┘         └─ -A ─┘

                                      ┌─ -x /var/agentx/master ─┐
  ►────────────────────────────────────────────────────────────────────►◄
        └─ -f ─┘  └─ -P <pidfile> ─┘  └─ -x <agentx_socket> ───┘
```
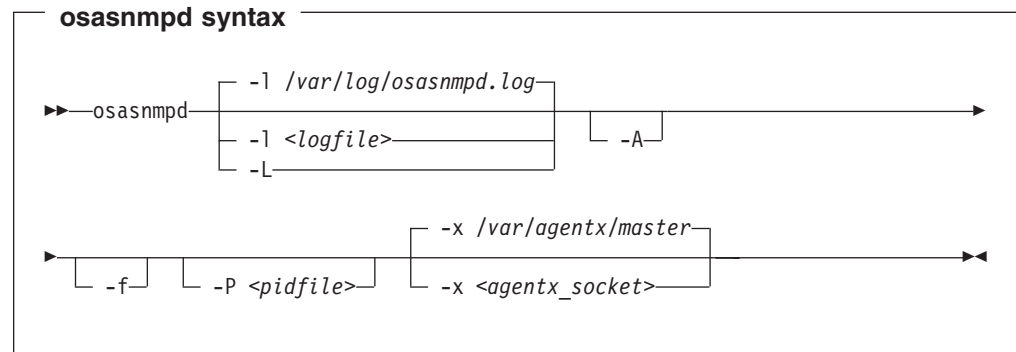
-l or **--logfile** *<logfile>*
  specifies a file for logging all subagent messages and warnings, including stdout and stderr. If no path is specified, the log file is created in the current directory. The default log file is `/var/log/osasnmpd.log`.

-L or **--stderrlog**
  print messages and warnings to stdout/stderr.

-A or **--append**
  appends to an existing log file rather than replacing it.

-f or **--nofork**
  prevents forking from the calling shell.

-P or **--pidfile** *<pidfile>*
  saves the process ID of the subagent in a file *<pidfile>*. If a path is not specified, the current directory is used.

-x or **--sockaddr** *<agentx_socket>*
  specifies the socket to be used for the AgentX connection. The default socket is `/var/agentx/master`.

  The socket can either be a UNIX domain socket path, or the address of a network interface. If a network address of the form `inet-addr:port` is specified, the subagent uses the specified port. If a net address of the form `inet-addr` is specified, the subagent uses the default AgentX port, 705. The AgentX sockets of the snmpd daemon and osasnmpd must match.

-h or **--help**
  displays help information for the command.

-v or **--version**
  displays version information for the command.

osasnmpd

Examples

To start the osasnmpd subagent with all default settings:

```
# osasnmpd
```

qetharp - Query and purge OSA and HiperSockets ARP data

Scenario

The **qetharp** command is used 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 ─┘  ├─ -a <interface>── -i <ip_address>── -m <mac_address>─┤
                     ├─ -d <interface>── -i <ip_address>────────┤
                     └─ -p <interface>──────────────────────────┘
```
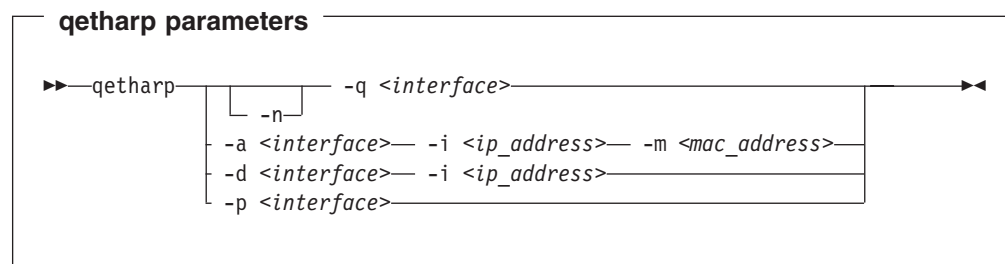
The meanings of the parameters of this command are as follows:

-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.

<interface>
> The qeth interface to which the command applies.

-a or **--add**
> Adds a static ARP entry to the OSA adapter card.

<ip_address>
> 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>
> 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.

qetharp

-v or **--verbose**
> 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

Scenario

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 95) 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
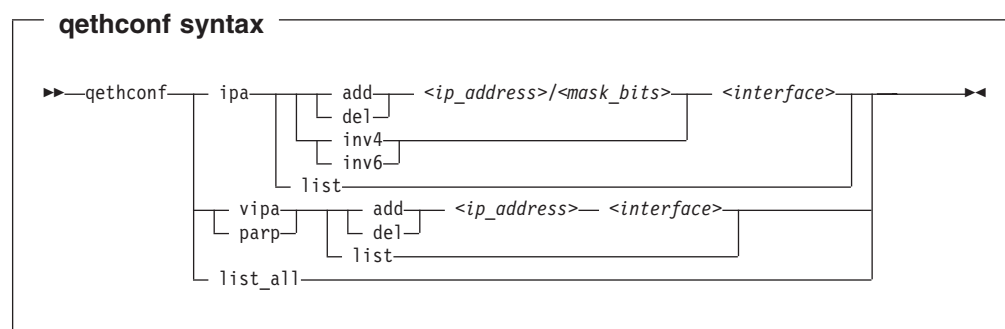
The meanings of the parameters of this command are as follows:

ipa
> Configure qeth for IP address takeover (IPA).

vipa
> Configure qeth for virtual IP address (VIPA).

parp
> Configure qeth for proxy ARP.

add
> Add an IP address or address range.

del
> Delete an IP address or address range.

inv4
> Invert 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
> Invert 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
> List existing definitions for specified qeth function.

list_all

List existing definitions for IPA, VIPA, and proxy ARP.

<ip_address>

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>

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>

Name of the interface associated with the specified address or address range.

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
```

san_disc - Discover ports and LUNs in Fibre Channel SANs

Scenario

Use the san_disc command to discover ports and scan LUNs in Fibre Channel Storage Area Networks (SANs). The tool depends on HBA API which is defined in FC-HBA. Some commands only work for a switched fabric topology.

Format

```
  ┌─────────────── san_disc syntax ───────────────────────────────────────┐
  │                                                                        │
  │                ┌──────────────────────┐                               │
  │  ▶▶──san_disc──┼──────────────────────┼──── -c ──── <cmd> ──────────▶◀│
  │                │   ┌─ -a── <ID> ──────┐│                               │
  │                │   ├─ -d── <DID> ─────┤│                               │
  │                │   ├─ -p── <name> ────┤│                               │
  │                │   ├─ -r── <revision>─┤│                               │
  │                │   └─ -V──────────────┘│                               │
  │                                                                        │
  └────────────────────────────────────────────────────────────────────────┘
```
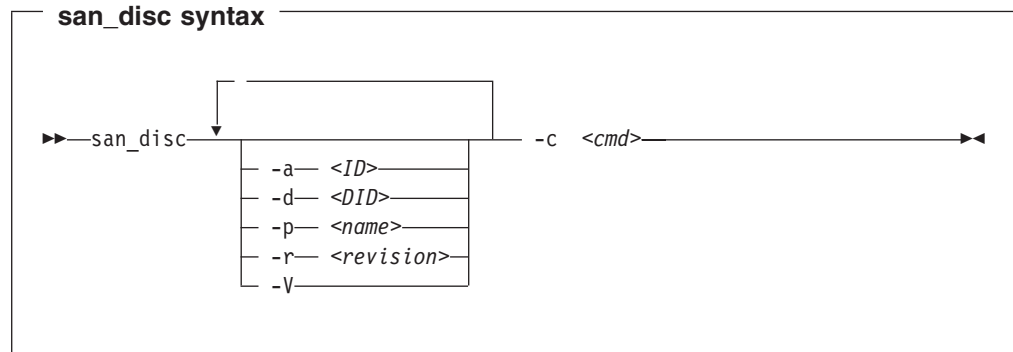
Where:

-a or **--adapter-id** *<ID>*
    specifies the ID of an adapter for commands PORT_LIST, GA_NXT, RNID, REPORT_LUNS. Valid adapter IDs are returned by the HBA_LIST command.

-d or **--did** *<DID>*
    specifies the destination ID for the GA_NXT command. The destination ID is a 3-byte hexadecimal value, for example, `0x65013f`.

-p or **--port-name** *<name>*
    specifies the WWPN for the RNID and REPORT_LUNS commands. The port name is an 8-byte hexadecimal value, for example, `0x500500ab0012cd00`.

-r or **--revision** *<revision>*
    specifies the revision used for Generic Service Requests. The default value 1 corresponds to FC-GS-2. Other possible values are 2 for FC-GS-3 and 3 for FCGS- 4. Set the revision if a switch does not support FC-GS-2, but a later version of FCGS.

-V or **--verbose**
    generates verbose output. Currently this option affects only the output of the GA_NXT and PORT_LIST commands. When you use this option, an additional column in the output shows the associated type of a port.

-v or **--version**
    Prints the version number of **san_disc** and exits.

-h or **--help**
    Prints out an overview of the syntax.

-c or **--command** *<cmd>*
    specifies the command to be executed. See below for detailed command specification.

The **san_disc** command provides some Generic Service, Extended Link Service, and SCSI commands. A list of the supported commands and their required arguments is as follows:

Generic service commands

GA_NXT (Get All Next)
Arguments: `-c GA_NXT -a <ID> -d <DID>`

Return name server objects for the next registered port identifier greater than <DID> for the zone to whihc the specified adapter belongs.

Arguments: `-c GA_NXT -a <ID> -d <DID> -V` The same as above, but additionally an RNID ELS command is sent to the port to detect its associated type. The associated type is displayed in an additional column. If the port does not respond to the RNID ELS command the associated type is empty.

Extended link service commands

RNID (Request Node Identification Data)
Arguments: `-c RNID -a <ID> -p <name>`

Display information from the RNID payload (with General Topology Discovery format) for the specified remote port on the specified adapter.

SCSI commands

REPORT_LUNS (Report Logical Unit Numbers)
Arguments: `-c REPORT_LUNS -a <ID> -p <name>`

Generate a list of available logical unit numbers for the specified remote port on the specified adapter.

Other commands

HBA_LIST (List Available Host Bus Adapters)
Arguments: `-c HBA_LIST`

Generate a list of available host bus adapters. For each adapter an ID, port name, and node name of the adapter port, the serial number and a vendor-specific ID are given. The vendor ID is converted to a bus ID which is needed on Linux for zSeries to unambiguously identify a virtual adapter. This is because WWPN and WWNN might be the same for virtual adapters sharing the same physical adapter.

PORT_LIST (List Registered Ports)
Arguments: -c PORT_LIST -a <ID>

Generate a list of ports registered at the nameserver directory service of the fabric to which the specified adapter is connected. A remote port can only be discovered if it is in the same zone as the adapter port.

Arguments: -c PORT_LIST -a <ID> -V

Same as above, but additionally an RNID ELS command is sent to each port to detect the associated type of that port. The associated type is displayed in an additional column. If the port does not respond to the RNID ELS command the associated type is empty.

Scenario

This scenario shows how to configure an adapter and port using **san_disc**.
1. Load the zfcp_hbaapi module:

```
# modprobe zfcp_hbaapi
```

2. Check for available FC adapters:

```
# san_disc -c HBA_LIST
 Number of Adapters: 6
 No.   Port WWN           Node WWN           SerialNumber      Busid
   1 (adapter unavailable)
   2 (adapter unavailable)
   3 (adapter unavailable)
   4 (adapter unavailable)
   5 (adapter unavailable)
   6 (adapter unavailable)
```

This shows that there are no adapters available.

3. Set an adapter online:

```
# chccwdev 0.0.3d0c --online
```

Check again for available FC adapters:

```
# san_disc -c HBA_LIST
 Number of Adapters: 6
 No.   Port WWN           Node WWN           SerialNumber      Busid
   1 (adapter unavailable)
   2 (adapter unavailable)
   3 (adapter unavailable)
   4 (adapter unavailable)
   5 (adapter unavailable)
   6 0x5005076401c08f98 0x5005076400cd6aad IBM020000000D6AAD  0.0.3d0c
```

4. Check for available ports on adapter 6 (0.0.3d0c):

```
# san_disc -c PORT_LIST -a 6
 No.  Port WWN            Node WWN            DID      Type
    1 0x5005076401408f98 0x5005076400cd6aad 0x650513 N_Port
    2 0x500507640140863c 0x5005076400cd6aad 0x650613 N_Port
    3 0x50050764010087ef 0x5005076400cd6aad 0x650713 N_Port
    4 0x50050764014087f0 0x5005076400cd6aad 0x650913 N_Port
...
   22 0x500507630300c562 0x5005076303ffc562 0x652113 N_Port
...
   75 0x5005076401202fd8 0x5005076400c1ab8a 0x683f13 N_Port
   76 0x5005076401808f99 0x5005076400c1ab8a 0x684013 N_Port
   77 0x50050763031b0104 0x5005076303ffc104 0x684113 N_Port
   78 0x5005076300c393cb 0x5005076300c093cb 0x684213 N_Port
   79 0x50050764012022e4 0x5005076400c1ab8a 0x684313 N_Port
   80 0x5005076401408f98 0x5005076400cd6aad 0x650513 N_Port
```

5. Configure a port:

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

6. Scan for available LUNs on adapter 6 (0.0.3d0c), port 0x500507630300c562:

```
# san_disc -c REPORT_LUNS -a 6  -p 0x500507630300c562
 Number of LUNs: 268
    No.  LUN
      1 0x4010400000000000
      2 0x4010400100000000
      3 0x4010400200000000
      4 0x4010400300000000
      5 0x4010400400000000
...
    267 0x4011401900000000
    268 0x4011401a00000000
```

scsi_logging_level - Set and get the SCSI logging level

Scenario

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──┬─────────────────────────┬──┬─ -s ─┬──►◄
                        │  ┌──────────────────┐   │  ├─ -g ─┤
                        └──┤ -a─ <level> ─────┤───┘  └─ -c ─┘
                           │ -E─ <level> ─────│
                           │ -T─ <level> ─────│
                           │ -S─ <level> ─────│
                           │ -M─ <level> ─────│
                           │ --mlqueue─ <level> ─│
                           │ --mlcomplete─ <level> ─│
                           │ -L─ <level> ─────│
                           │ --llqueue─ <level> ─│
                           │ --llcomplete─ <level> ─│
                           │ -H─ <level> ─────│
                           │ --hlqueue─ <level> ─│
                           │ --hlcomplete─ <level> ─│
                           └ -I─ <level> ─────┘
```
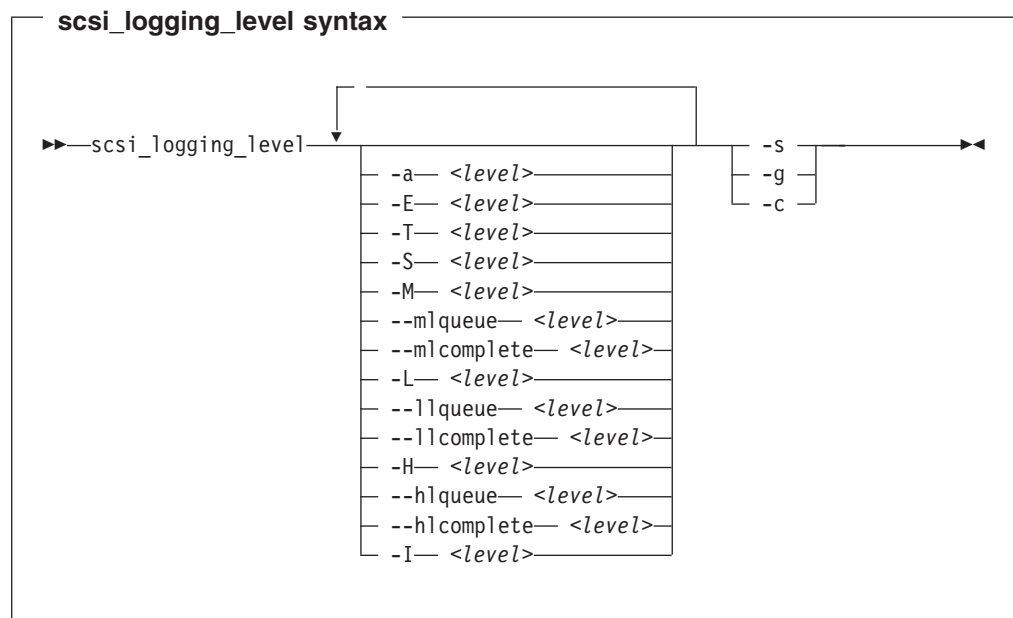
Where:

-a or **--all** *<level>*
    specifies 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**
    prints version information.

-h or **--help**
    prints 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

```
#> 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
```

snipl – Simple network IPL (Linux image control for LPAR and VM)

Scenario

snipl (**s**imple **n**etwork **IPL**) is a command line tool for *remotely controlling Linux images* using either:

- Basic System z support element (SE) functions for systems running in **LPAR mode,** or
- Basic z/VM system management functions for systems running as a **z/VM guest** (z/VM 4.4 or higher).

Note: Be aware that incautious use of **snipl** can result in loss of data.

LPAR mode

In LPAR mode, **snipl** allows you to:

- *Load* an LPAR.
- *Send* and *retrieve* operating system messages.
- *Activate*, *reset*, or *deactivate* an LPAR for I/O-fencing purposes.

Using **snipl** in LPAR mode allows you to overcome the limitations of the SE graphical interface when **snipl** is used for I/O-fencing from within a clustered environment of Linux systems that run in LPAR mode.

snipl uses the network management application programming interfaces (API) provided by the SE, which establishes an SNMP network connection and uses the SNMP protocol to send and retrieve data. The API is called "hwmcaapi". It has to be available as shared library.

To establish a connection (using a valid community):

- In the SE *SNMP configuration* task, configure the IP address of the initiating system and the community.
- In the SE *settings* task, configure SNMP support.
- In your firewall settings, ensure that UDP port 161 and TCP port 3161 are enabled.

If **snipl** in LPAR mode repeatedly reports a timeout, the target SE is most likely inaccessible or not configured properly. For details on how to configure the SE, refer to *zSeries Application Programming Interfaces*, SB10-7030, or *S/390 Application Programming Interfaces*, SC28-8141, which is obtainable from the following Web site:

```
ibm.com/servers/resourcelink/
```

z/VM mode

In z/VM mode, **snipl** allows you to remotely control basic z/VM system management functions. You can:

- *Activate*, *reset*, or *deactivate* an image for I/O-fencing purposes.

snipl in z/VM mode uses the system management application programming interfaces (APIs) of z/VM (version 4.4 or higher). To communicate with the z/VM host, **snipl** establishes a network connection and uses the RPC protocol to send and retrieve data.

To establish a connection to the VM host, the VSMSERVE server must be configured and the vmsapi service must be registered on the target VM host. Also,

snipl

there has to be an account for the specified user ID on the host. If **snipl** in VM
mode repeatedly reports ″RPC: Port mapper failure - RPC timed out″, it is most
likely that the target z/VM host is inaccessible, or the service is not registered, or
the configuration of the VSMSERVE server is not correct.

Note: The configuration of VSMSERVE requires DIRMAINT authorization.

For details about configuration of the VSMSERVE server on z/VM refer to *z/VM:
Systems Management Application Programming*, SC24-6063 obtainable from the
following Web site:

    ibm.com/vm/

Usage

Command line syntax (LPAR mode)

```
┌─ snipl command (LPAR mode) ──────────────────────────────────────────────┐
│                                                                          │
│                  ┌──────────────────┐                                    │
│                  ▼                  │                                    │
│  ▶▶──snipl───────┬─<image_name>─┬──────┬──────────────────────┬─────────▶ │
│                  └──────────────┘      └─ -L <ip_address> ─────┘          │
│                                                                          │
│      ┌─ -p public ──────┐   ┌─ -f <defaultfile> ─┐         (1)           │
│  ▶───┼──────────────────┼───┼────────────────────┼──────────────────────▶ │
│      ├─ -p <community>──┤   └─ -f <filename> ─────┘                       │
│      └─ -P ─────────────┘                                                 │
│                                                                          │
│      ┌─ --timeout 10000 ────┐                                            │
│  ▶───┼──────────────────────┼────────────────────────────────────────────▶ │
│      └─ --timeout <timeout>─┘                                            │
│                                                                          │
│  ▶───┬──────────────────────────────────────────────────────┬──────────▶◀ │
│      │        ┌─ --profilename <defaultprofile> ─┐   (2)      │           │
│      ├─ -a ───┼───────────────────────────────────┤           │           │
│      │   └-F-┘└─ --profilename <filename> ────────┘           │           │
│      ├─ -d ──────────┐                                        │           │
│      │       └-F-┘                                            │           │
│      ├─ -r ──────────┐                                        │           │
│      │       └-F-┘                                            │           │
│      ├─ -l ─┤ loadparms ├─────────────────────────────────────┤           │
│      ├─ -x ──────────────────────────────────────────────────┤           │
│      │      ┌─ --msgtimeout 5000 ──┐                          │           │
│      └─ -i ─┼──────────────────────┼──────────────────────────┘           │
│             └─ --msgtimeout <interval>─┘                                  │
│                                                                          │
│  **loadparms:**                                                          │
│                                                                          │
│  ├──┬──────┬─┬──────────────────────┬─┬────────────────────────────────┬─▶ │
│     └-F-┘  └─ -A <load_address> ─────┘ └─ --parameters_load <string> ───┘  │
│                                                                          │
│      ┌─ --load_timeout 60 ──────┐                                        │
│  ▶───┼──────────────────────────┼─┬──────────┬─┬──────────────┬──────────┤ │
│      └─ --load_timeout <timeout>─┘ └--noclear-┘ └--storestatus-┘          │
│                                                                          │
│  **Notes:**                                                              │
│  1   See description of the **-f** option.                               │
│  2   See description of the **--profilename** option.                    │
└──────────────────────────────────────────────────────────────────────────┘
```
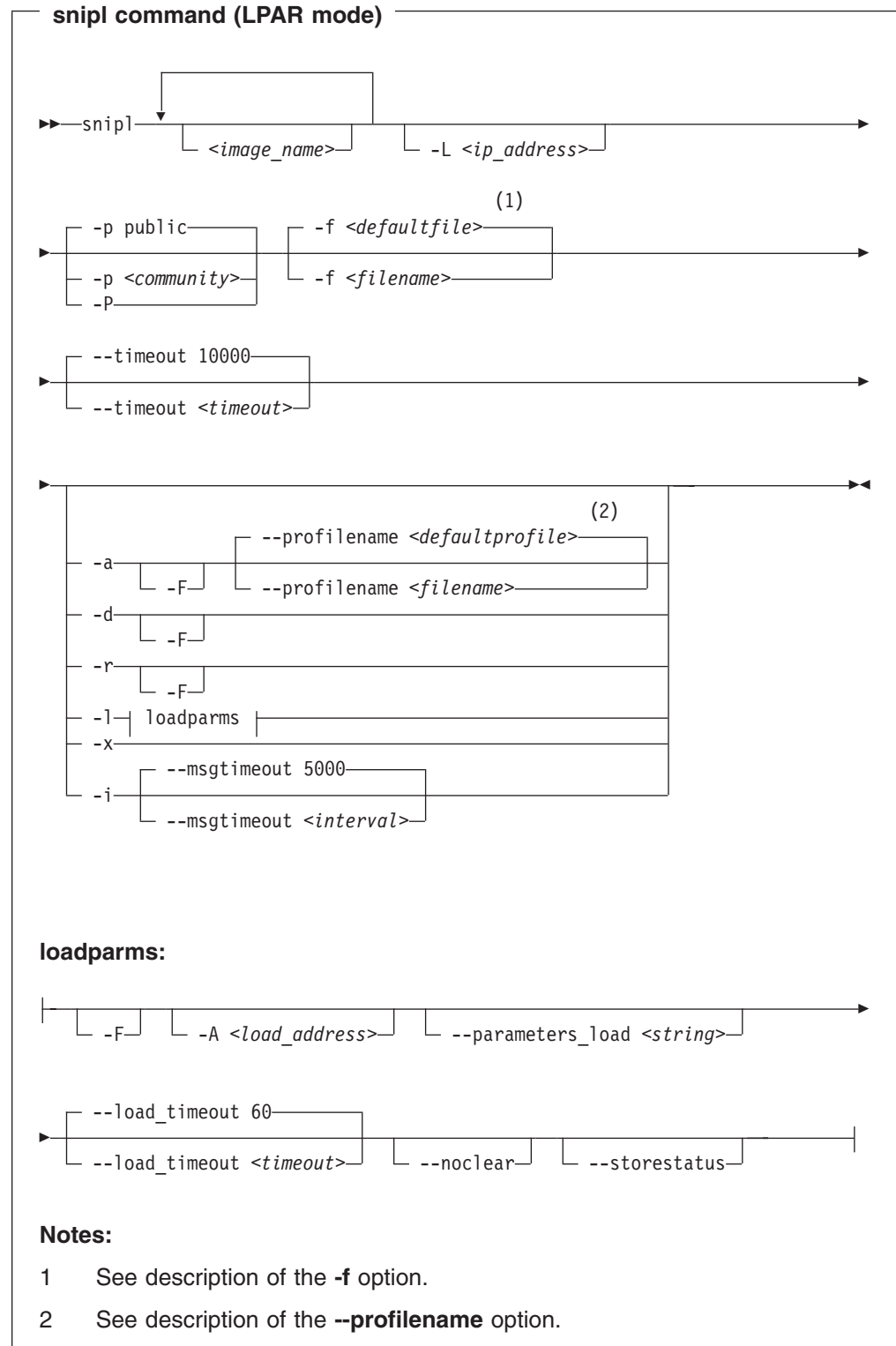
Command line syntax (VM mode)

```
 snipl command (VM mode)

 ►►─snipl─┬──────────────────┬─┬──────────────────┬──────────────────►
          │  ┌◄──────────┐   │ │  -V <ip_address> │
          └──┴─<image_name>──┴─┘ └──────────────────┘

 ►─┬─────────────────────────────────────────┬─┬────────────────┬──────►
   │ -u <userid>─┬─ -p <password> ─┬─        │ │ -f <filename>  │
   │             └─ -P ────────────┘         │ └────────────────┘
   └─────────────────────────────────────────┘

 ►─┬─ -a ─────────────┬────────────────────────────────────────────►◄
   ├─ -d ─┬───────┬───┤
   │      └─ -F ──┘   │
   ├─ -r ─────────────┤
   └─ -x ─────────────┘
```
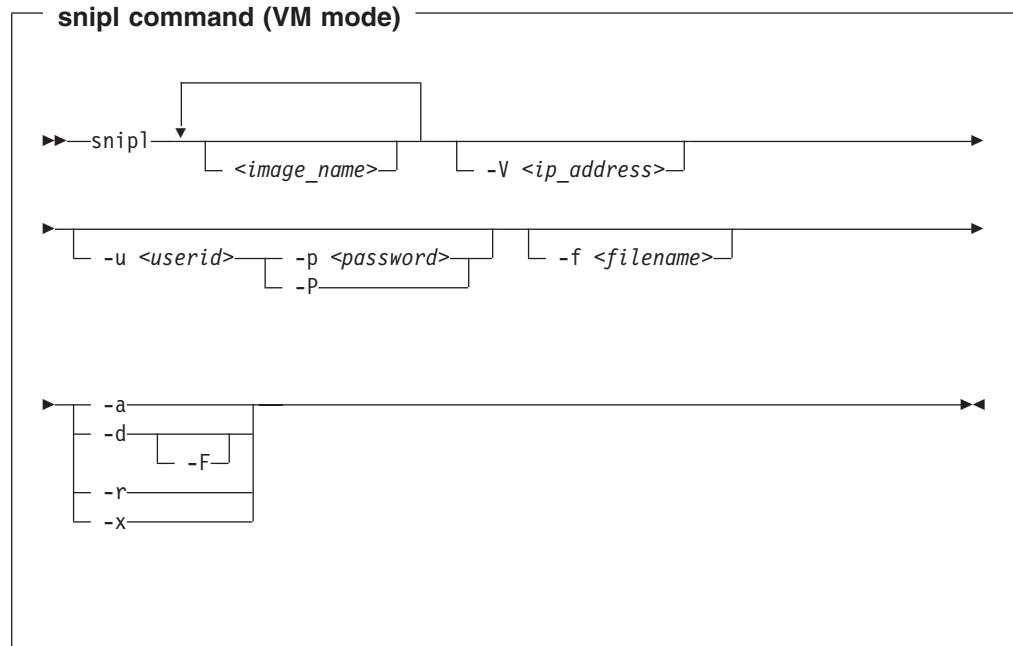
Options and Parameters

<image_name>
Specifies the name of the targeted LPAR or z/VM guest. This parameter is required for *--activate*, *--deactivate*, *--reset*, *--load*, and *--dialog*. If the same command is to be performed on more than one image of a given server, more than one *<image_name>* can be specified. Exception: A *--dialog* can only be started with one image.

-V *<ip_address>* or **--vmserver** *<ip_address>*
Specifies the server to be of type VM. Use this option if the system is running in VM mode. Also specifies the IP-address/host-name of targeted VM-host. This option can also be defined in the configuration file and thus may also be omitted.

-L *<ip_address>* or **--lparserver** *<ip_address>*
Specifies the server to be of type LPAR. Use this option if the system is running in LPAR mode. Specifies the IP-address/hostname of targeted SE. This option can also be defined in the configuration file and thus may also be omitted.

-u *<userid>* or **--userid** *<userid>*
VM only: Specifies the userid used to access the VM-host. If none is given, the configuration file can be used to determine the userid for a given IP-address or VM-guest-name.

-p *<community>* | *<password>* or **--password** *<community>* | *<password>*
• For LPAR mode, the option specifies the *<community>* (HMC term for password) of the initiating host system. The default for *<community>* is "public". The value entered here must match the entry contained in the SNMP configuration settings on the SE.
• For VM mode, specifies the password for the given user ID.

If no password is given, the configuration file can be used to determine the password for a given IP address, LPAR, or VM guest name.

-P or **--promptpassword**
Lets **snipl** prompt for a password in protected entry mode.

-f *<filename>* or **--configfilename** *<filename>*
Specifies the name of a configuration file containing HMC/SE IP-addresses together with their community (=password) and VM IP-address together with their userid and password followed by a list of controlled `LPARnames` or `VM-guest-names`. Default *user-specific filename* is `$HOME/.snipl.conf` and *default system-wide filename* is `/etc/snipl.conf`. Without available configuration file all required options have to be specified with the command. The structure of the configuration file is described below.

-x or **--listimages**
Lists all available images for the specified server.
- For VM this may be specified with *image*, *server*, *server+user* or *image+user* according to the uniqueness in the configuration file. In case of VM the returned list is retrieved from the configuration file only.
- For LPAR just the *server name* is used to retrieve the actual images. The information is directly retrieved from the SE.

-a or **--activate**
Issues an activate command for the targeted LPAR or VM guest.

-d or **--deactivate**
Issues a deactivate command for the target LPAR or VM guest.

-r or **--reset**
Issues a reset command for the targeted LPAR(s) or VM guest(s).

-l or **--load**
LPAR only: Issues a load command for the target LPAR.

-i or **--dialog**
LPAR only: This option starts an operating system message dialog with the targeted LPAR. It allows the user to enter arbitrary commands, which are sent to the targeted LPAR. In addition, `dialog` starts a background process, which continuously retrieves operating system messages. The output of this polling process is sent to `stdout`. The operating system messages dialog is aborted by pressing `CTRL-D`. This also kills the polling process. After the dialog is terminated, **snipl** exits.

-t *<timeout>* or **--timeout** *<timeout>*
LPAR only: Specifies the timeout in milliseconds for general management API calls. The default is 10000 ms.

-m *<interval>* or **--msgtimeout** *<interval>*
LPAR only: Specifies – in conjunction with `--dialog` – the interval in milliseconds for management API calls that retrieve operating system messages. The default value is set to 5000 ms.

-F or **--force**
Forces the imageoperation.
- VM: in conjunction with `--deactivate` non graceful deactivation of the image.
- LPAR: In conjunction with `--activate`, `--deactivate`, `--reset` and `--load` allows unconditional execution of the command regardless of the state of the image.

--profilename *<filename>*
> LPAR only: In conjunction with `--activate` the option specifies the profile name used on the activate command for LPAR mode. If none is provided, the HMC/SE default profile name for the given image is used.

-A *<loadaddress>* or **--address_load** *<loadaddress>*
> LPAR only: In conjunction with `--load` specifies the load address in 4 hexadecimal digits. If none is provided, the address of the previous load is used as load address.

--parameters_load *<string>*
> LPAR only: In conjunction with `--load` specifies a parameter string for loading. If none is given, the parameter string of the previous load is used. This parameter is used for instance for `IPL` of z/OS and z/VM.

--noclear
> LPAR only: In conjunction with `--load` denies memory clearing before loading. The memory is cleared by default.

--load_timeout *<timeout>*
> LPAR only: In conjunction with `--load` specifies the maximum time for load completion, in seconds. The value must be between 60 and 600 seconds. The default value is 60 seconds.

--storestatus
> LPAR only: In conjunction with `--load` requests status before loading. The status is not stored by default.

-v or **--version**
> Prints version of **snipl** and exits.

-h or **--help**
> Prints usage and exits.

Structure of the configuration file

A configuration file contains a list of addresses (IP-addresses of an SE or a z/VM host), and the host type (LPAR vs. VM). The configuration file also contains a list of image names available for control on the subswitch.

- For LPAR, the list of image names can also be retrieved from the SE.
- For z/VM the list can only be retrieved by users with appropriate z/VM access rights. Therefore, a local list must be available.

The following is an example for the structure of the **snipl** configuration file:

```
Server = <IP-address>
type = <host-type>
password = <password>
image = <imagename>
image = <imagename>
image = <imagename>
Server = <IP-address>
type = <host-type>
user = <username>
password = <password>
image = <imagename>
image = <imagename>
image = <imagename>
image = <imagename>
```

Blanks and n/ are separators. The keywords are not case-sensitive.

snipl command examples

LPAR mode: Activate:

```
# snipl LPARLNX1 -L 9.164.70.100 -a -P
Enter password: Warning : No default configuration file could be found/opened.
processing......
LPARLNX1: acknowledged.
```

LPAR mode: Load: Load using configuration file:

```
# snipl LPARLNX1 -f xcfg -l -A 5119
processing......
LPARLNX1: acknowledged.
```

z/VM mode: Activate using configuration file:

```
# snipl -f xcfg -a vmlnx2 vmlnx1
* ImageActivate : Image vmlnx1 Request Successful
* ImageActivate : Image vmlnx2 Image Already Active
```

Connection errors and exit codes

If a connection error occurs (e.g.*timeout*, or *communication failure*), **snipl** sends an *error code* of the management API and a *message* to `stderr`. For

- `snipl --vmserver` the shell exit code is set to "1000 + error code"
- `snipl --lparserver` the shell exit code is set to "2000 + error code"

Return codes like

```
LPARLNX1: not acknowledged — command was not successful — rc is 135921664
```

are described in "Appendix B" of the HWMCAAPI document *zSeries Application Programming Interfaces*, SB10–7030. You can obtain this publication from the following Web site: `ibm.com/servers/resourcelink/`.

Additionally, the following **snipl** error codes exist. They are accompanied by a short message on `stderr`:

1   An unknown option is specified.

2   An option with an invalid value is specified.

3   An option is specified more than once.

4   Conflicting options are specified.

5   No command option is specified.

6   Server is not specified and cannot be determined.

7   No image is specified.

8   User-ID is not specified and cannot be determined.

9   Password is not specified and cannot be determined.

10   A specified image name does not exist on the server used.

20   An error occurred while processing the configuration file.

22   Operation `--dialog`: More than one image name is specified.

30   An error occurred while loading one of the libraries *libhwmcaapi.so* or *libvmsmapi.so*

40   Operation `--dialog` encounters a problem while starting another process.

41	Operation `--dialog` encounters a problem with `stdin` attribute setting.
50	Response from HMC/SE is cannot be interpreted.
60	Response buffer is too small for HMC/SE response.
90	A storage allocation failure occurred.

If no error occurs, a shell exit code of 0 is returned upon completion of **snipl**.

Recovery

Currently, **snipl** does not

- recover connection failures.
- recover errors in API call execution.

In these cases, it is sufficient to *restart* the tool. Should the problem persist, a networking failure is most likely. In this case, increase the timeout values for `snipl --lparserver`.

STONITH support (snipl for STONITH)

The STONITH implementation is part of the Heartbeat framework of the High Availability Project (`http://linux-ha.org/`) and STONITH is generally used as part of this framework. It can also be used independently, however. A general description of the STONITH technology can be found at: `http://linux-ha.org/stonith.html`.

The STONITH support for **snipl** can be regarded as a driver for one or more virtual power switches controlling a set of Linux images located on LPARs or z/VM instances as z/VM guests. A single LPAR or z/VM host can be seen as a VPS subswitch. STONITH requires the availability of a list of the controllable images by a switch. For this Linux Image Control VPS, the set of controlled images is retrieved from different locations depending on access rights and configuration.

The format of the **snipl** for STONITH configuration file corresponds with the configuration file format of **snipl**, see "Structure of the configuration file" on page 320.

Before you start: The setup requirements for using the STONITH plug-in differ, depending on the environment into which you want to implement it.

- **snipl** for STONITH in LPAR mode:

  The SE must be configured to allow the initiating host system to access the network management API. Direct communication with the HMC is not supported.

  For details, refer to either of these publications, as applicable:

  > *zSeries Application Programming Interfaces*, SB10-7030

  > *S/390 Application Programming Interfaces*, SC28-8141

  You can obtain these publications from the following Web site: `ibm.com/servers/resourcelink/`

- **snipl** for STONITH in VM mode:

  To communicate with the z/VM host, **snipl** establishes a network connection and uses the Remote Procedure Call (RPC) protocol to send and retrieve data.

  Communication with z/VM requires prior configuration of the VSMSERVE server on z/VM. For details, refer to:

  > *z/VM: Systems Management Application Programming*, SC24–6063

  You can obtain this publication from the following Web site: `ibm.com/vm/`

tape390_display - display messages on tape devices and load tapes

Scenario

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 ──────────────────────────────────────────────

                            ┌─ -t standard ─┐
  ►►──tape390_display──┬────┬──┼─ -t ──┬─ load ───┬──────┬──────────┬──── <message1> ──── <node>──►◄
                       └─ -l ─┘       ├─ unload ─┤      └─ -b ─┘
                                      │        (1) │    └─ <message1>── <message2> ─┘
                                      ├─ reload ─┤
                                      └─ noop ───┘

  Notes:

  1    With reload, two messages must be specified and option -b is not
       permitted.
```
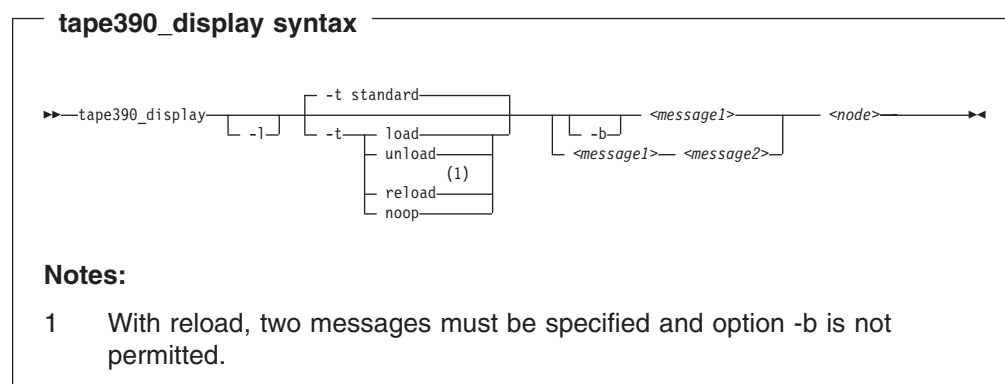
where:

-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

-h or **--help**
> prints help text

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

Scenario

tunedasd is used 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 42)

Before you start:

- You must have root permissions.
- For the performance statistics:
  - Your kernel needs to have been compiled with the kernel configuration option CONFIG_DASD_PROFILE (see "Building a kernel with the DASD device driver" on page 35).
  - Data gathering must have been switched on by writing "on" to `/proc/dasd/statistics`.

Format

```
  ┌─ tunedasd syntax ─────────────────────────────────────────────

                           ┌─ -h ─────────────────────────┐
    ▶▶── tunedasd ──────────┼──────────────────────────────┼──────▶◀
                           │                    ◄─────────┐ │
                           │                    ┌─ <node> ─┘ │
                           ├─ -g ──────────────┤          │
                           ├─ -c <mode> ───────┤          │
                           │      └─ -n <cylinders> ─┘     │
                           ├─ -S ───────────────┤
                           ├─ -L ───────────────┤
                           ├─ -O ───────────────┤
                           ├─ -R ───────────────┤
                           └─ -P ───────────────┤
                                └─ -I <row> ─┘
```
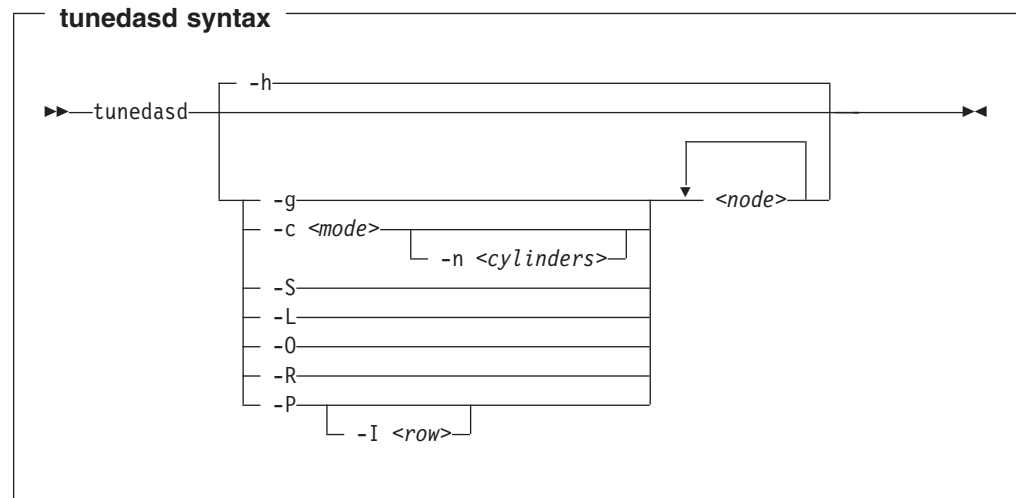
Where:

<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**
:   Unconditionally reserves the device. 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**
:   reset the profile information of the device.

-P or **--profile**
:   Prints a usage profile of the device.

-I *<row>* or **--profile_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 VM hypervisor

Scenario

vmcp is used to:
- Send control program (CP) commands to the VM hypervisor.
- Display VM's response.

The **vmcp** command expects the command line as a parameter and returns the response to stdout. Error messages are written to stderr.

You can use vmcp as a device node (usually /dev/vmcp, see Chapter 17, "z/VM CP interface device driver," on page 199) or as a user space tool. In both cases, the vmcp module must be loaded or compiled in the kernel. Distributors should consider providing a boot script that automatically loads the module if the guest runs under z/VM.

The vmcp tool can coexist with the cpint package.

Before you start:
- You must have root permissions.
- Ensure that vmcp is loaded by issuing: `modprobe vmcp`

Format

```
  vmcp syntax

►►──vmcp──┬──-h──────────────────────────────┬──►◄
          ├──-v──────────────────────────────┤
          │          ┌─ 8 KB ─┐              │
          └──┬────┬──┴──┬──────┬──<command>──┘
             └─-k─┘     └─-b─┘
```
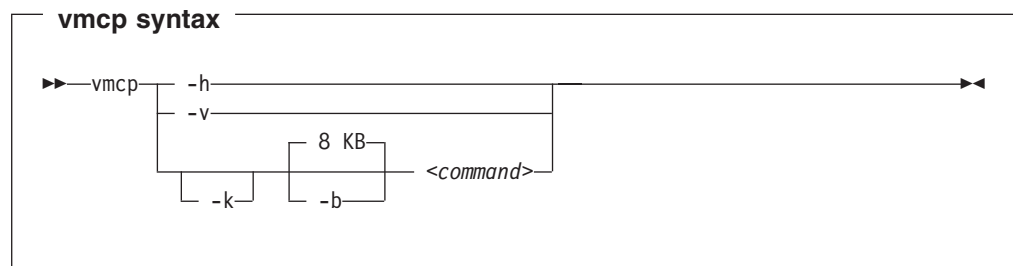
Where:

-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 VM's 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
```

zipl – zSeries initial program loader

Scenario

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 *Linux on System z9 and zSeries Using the Dump Tools*.
- 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 332).

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 32. 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 333 for details.	-i	--image	image=
Prepare a DASD or tape dump device See "Preparing a DASD or tape dump device" on page 335 for details.	-d	--dumpto	dumpto=
Prepare a SCSI dump device See "Preparing a dump device on a SCSI disk" on page 337 for details.	-D	--dumptofs	dumptofs=
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 339 for details.	-s	--segment	segment=

Table 32. zipl base functions (continued)

Base function	Command line short option	Command line long option	Configuration file option
Install a menu configuration	-m	--menu	(None)
See "Installing a menu configuration" on page 348 for details.			

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 348), the entire configuration must be defined using command-line parameters.

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.

zipl syntax overview
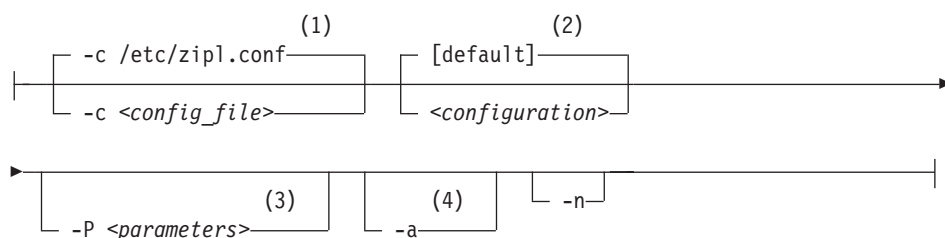
```
>>--zipl--+------+--+-----------+--+-| parameters when omitting base function |-+--><
          +- -V -+  +- --dry-run -+ +- -i --i_parameters---------------------+
                                    +- -D --D_parameters---------------------+
                                    +- -s --s_parameters---------------------+
                                    +- -d --d_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 343).

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 335 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 339 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 "" on page 340 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──────┬─ ─┬─ -t <directory>─┬──────►
                    └──,<image_addr>─────┘   └─ -T <tape_node>─┘

►─┬─────────────────────────────────────────┬──────────────────────►
  └─ -r <ramdisk>─┬──────,0x800000──────┬────┘
                  └──,<initrd_addr>──────┘

►─┬───────────────────────────────────┬─┬─ -P <parameters>─┬─┬─ -a─┬─►◄
  └─ -p <parmfile>─┬──,0x1000──┬───────┘ └──────────────────┘ └─────┘
                   └─,<parm_addr>─┘
```
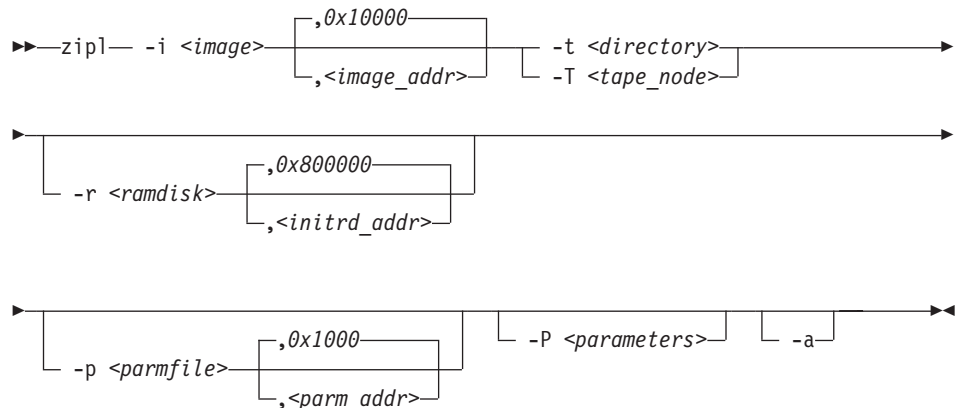
To prepare a device as a boot device you must specify:

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 335 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 "" on page 340 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 boot configuration within a configuration file section. Required specifications are shown in bold. See "" on page 343 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 55. 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 with a parameter file specification.

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 *Linux on System z9 and zSeries Using the Dump Tools* for details on how to process these dumps.

See "" on page 340 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 DASD or tape dump configuration in a configuration file. See "" on page 343 for a more comprehensive discussion of the configuration file.

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

Figure 56. 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 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 *Linux on System z9 and zSeries Using the Dump Tools* 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 339 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 "" on page 340 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 "" on page 343 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 and automatic deletion of the oldest dump if there is not enough space to 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.

Figure 58 summarizes how you can specify a file to be loaded to a DCSS within a configuration file section. See "" on page 343 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
```

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
-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.
-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 333 for details.
-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.

Command line short option Command line long option Configuration file option	Explanation
-p *<parmfile>*[,*<parm_addr>*] **--parmfile**=*<parmfile>*[,*<parm_addr>*]	In conjunction with a boot configuration, specifies the location of a kernel parameter file.
parmfile=*<parmfile>*[,*<parm_addr>*]	In conjunction with 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 337). 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 339 and "How kernel parameters from different sources are combined" on page 335 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 conjunction with a boot configuration, specifies kernel parameters. In conjunction with a SCSI dump configuration, specifies SCSI system dumper parameters (see "Preparing a dump device on a SCSI disk" on page 337) 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 339 and "How kernel parameters from different sources are combined" on page 335 for more information.
-d *<dump_device>*[,*<size>*] **--dumpto**=*<dump_device>*[,*<size>*]	Specifies the DASD partition or tape device to which a dump is to be written after IPL.
dumpto=*<dump_device>*[,*<size>*]	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 335 and *Linux on System z9 and zSeries Using the Dump Tools* for details.

zipl

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 337 and *Linux on System z9 and zSeries Using the Dump Tools* for details.
-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 339 for details.
-t *<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).
-T *<tape_node>* **--tape**=*<tape_node>* **tape**=*<tape_node>*	Specifies the tape device where **zipl** installs the boot loader code.
-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 344).
-n **--noninteractive** n/a	Suppresses all confirmation prompts (for example, when preparing a DASD or tape dump device).
-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.
-V **--verbose** n/a	Provides more detailed command output.
-v **--version** n/a	Prints version information.

Command line short option Command line long option	Explanation
Configuration file option	
-h --help	Displays help information.
n/a	

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 333 for details.

dumpto=<dump_device>
> Defines a DASD or tape dump configuration. See "Preparing a DASD or tape dump device" on page 335 for details.

dumptofs=<dump_partition>
> Defines a SCSI dump configuration. See "Preparing a dump device on a SCSI disk" on page 337 for details.

segment=<segment_file>
> Defines a DCSS load configuration. See "Installing a loader to initialize a discontiguous named saved segment (DCSS)" on page 339 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 346 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 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 348):

```
# 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 VM" on page 369 and "Example for a DASD menu configuration (LPAR)" on page 375 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.

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.                                                     │
│                                                                                 │
└─────────────────────────────────────────────────────────────────────────────┘
```
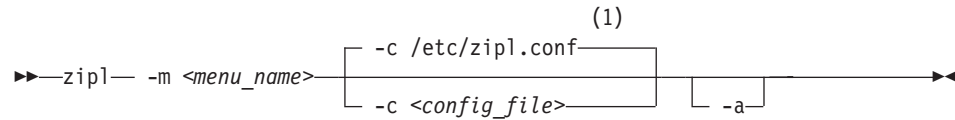
Where:

-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 345, you
could install a menu configuration with:

```
# zipl -m menu1
```

Chapter 28. Selected kernel parameters

There are two different ways of passing parameters to Linux:

- Passing parameters to your kernel at startup time (the parameter line)
- Configuring your boot loader to always pass those parameters

The kernel can only handle a parameter line file that is no larger than 896 bytes.

Device driver-specific kernel parameters are described in the setting up section of the respective device driver chapter. The following parameters affect Linux on System z in particular and are beyond the scope of an individual device driver:

- additional_cpus
- cio_ignore
- cio_msg
- ipldelay
- maxcpus
- mem
- noinitrd
- possible_cpus
- ramdisk_size
- ro
- root
- vmhalt
- vmpoff

additional_cpus

Usage

Specifies the number of CPUs that Linux can add to the system. For example, if there were two CPUs present at IPL, then specifying `additional_cpus=1` will set the number of possible CPUs to three. If the possible_cpus parameter is set, additional_cpus is ignored. See also "maxcpus" on page 356 and "possible_cpus" on page 359.

Format

```
┌─ additional_cpus syntax ──────────────────────────────────────────────

►►──additional_cpus=<number>─────────────────────────────────────►◄

```

Examples

```
additional_cpus=1
```

cio_ignore

Usage

When a Linux on System z instance boots, it senses and analyses all available devices. You can use the cio_ignore kernel parameter to specify a list of 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.

See also "Changing the list of devices to be ignored" on page 352.

Format

cio_ignore syntax

```
▶▶──cio_ignore=──┬─all───────────┬──────────────────────────────────▶◀
                 └─<device_spec>─┘  ┌────,────┐
                                 └,─▼─┬─────┬─<device_spec>─┘
                                      └─!───┘
```

<device_spec>:

```
├──┬─<device_bus_id>────────────────────────┬──┤
   └─<from_device_bus_id>-<to_device_bus_id>─┘
```
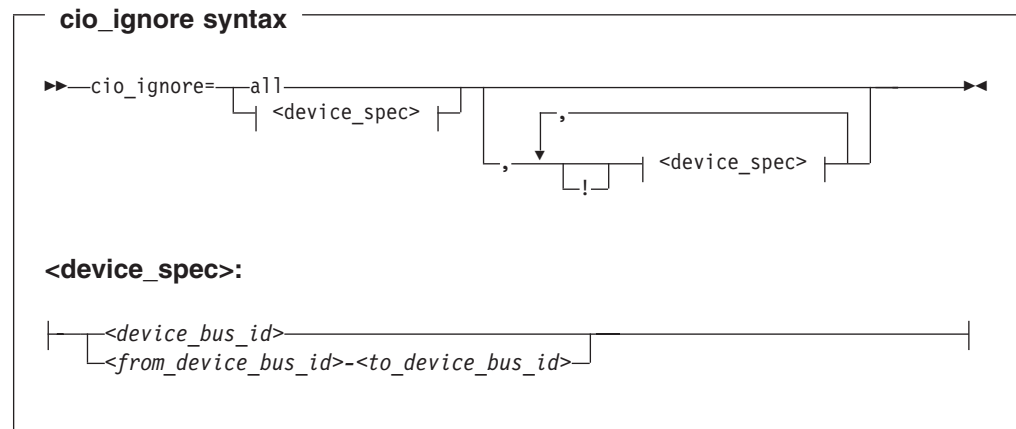
Where:

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 list of devices to be ignored

When Linux boots, it senses the available devices and analyses them. You can use the cio_ignore kernel parameter (see "cio_ignore" on page 351) to provide a list of devices that are to be excluded from sensing and analyzing.

After booting Linux you can display the list of devices to be ignored by issuing:

```
# cat /proc/cio_ignore
```

You can add devices to this list or remove devices from the list:

- When you remove a device from the list, the device is sensed and analyzed and, where possible, the corresponding device driver is informed. The device then becomes available to the system.
- When you add a device that has already been sensed and analyzed, there is no immediate effect. However, if such a device disappears with a machine check, it is ignored when it reappears.

To remove devices from the list of devices to be ignored issue a command of this form:

```
# echo free <device_list> > /proc/cio_ignore
```

To add devices to the list of devices to be ignored issue a command of this form:

```
# echo add <device_list> > /proc/cio_ignore
```

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>──┘
```
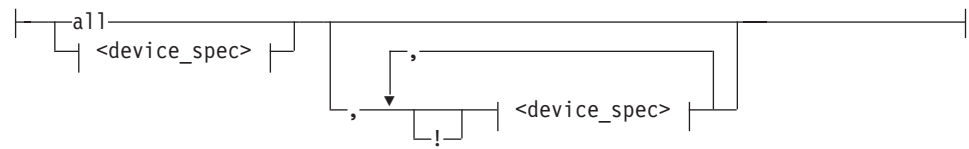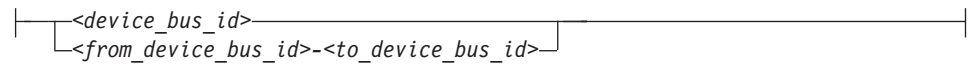
Where the keywords and variables have the same meaning as in "Format" on page 351.

Examples:

- This command removes all devices from the list of devices to be ignored.

```
# 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 list of devices to be ignored.

```
# 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 list of devices to be ignored.

```
# 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 list of devices to be ignored.

```
# echo free 0.0.c104 > /proc/cio_ignore
```

- This command adds the device with bus ID 0.0.c104 to the list of devices to be ignored.

```
# echo add 0.0.c104 > /proc/cio_ignore
```

cio_msg

Usage

Specifies whether I/O messages are to be sent to the console on boot-up.

These messages are usually suppressed (`cio_msg=no`) because on large machines with many attached devices the I/O layer generates a large number of these messages which can flood the console for a significant period of time. If you do need those messages (for example for debugging), you can switch them on manually using `cio_msg=yes`.

Format

```
  cio_msg syntax

              ┌─cio_msg=no──┐
  ▶▶──────────┴─cio_msg=yes─┴────────────────────────────────────────▶◀
```
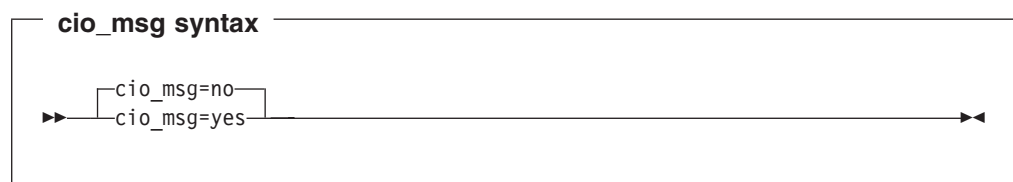
Examples

This example switches I/O messages to the console on boot:

```
cio_msg=yes
```

ipldelay

Usage

When you do a power on reset (POR), some activation and loading is done. This can cause Linux not to find the OSA-2 card. If you have problems with your OSA-2 card after booting, you might want to insert a delay to allow the POR, microcode load and initialization to take place in the OSA-2 card. The recommended delay time is two minutes. For example, `30s` means a delay of thirty seconds between the boot and the initialization of the OSA-2 card, `2m` means a delay of two minutes. The value *<time>* must be a number followed by either `s` or `m`.

Format

```
┌─ ipldelay syntax ──────────────────────────────────────────────────┐
│                                                                     │
│   ►►──ipldelay=<time>──┬─m─┬──────────────────────────────────►◄    │
│                        └─s─┘                                         │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

Examples

This example delays the initialization of the card by 2 minutes:

```
ipldelay=2m
```

This example delays the initialization of the card by 30 seconds:

```
ipldelay=30s
```

maxcpus

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 "additional_cpus" on page 350 and "possible_cpus" on page 359.

Format

```
┌─ maxcpus syntax ──────────────────────────────────────────────┐
│                                                                │
│  ►►──maxcpus=<number>──────────────────────────────────►◄      │
│                                                                │
└────────────────────────────────────────────────────────────┘
```

Examples

```
maxcpus=2
```

mem

Usage

Restricts memory usage to the size specified. The specified size must be suffixed either with M for megabyte or K for kilobyte.

Format

```
┌─ mem syntax ─────────────────────────────────────────────────┐
│                                                               │
│  ►►──mem=<size>──┬─M─┬──────────────────────────────────►◄    │
│                  └─K─┘                                         │
│                                                               │
└───────────────────────────────────────────────────────────────┘
```
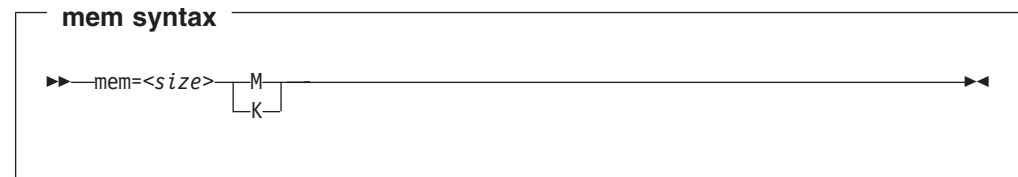
Examples

```
mem=64M
```

Restricts the memory Linux can use to 64 MB.

```
mem=123456K
```

Restricts the memory Linux can use to 123456 KB.

noinitrd

Usage

The noinitrd statement is required when the kernel was compiled with initial RAM disk support enabled. This command bypasses using the initial ramdisk.

This can be useful if the kernel was used with a RAM disk for the initial startup, but the RAM disk is not required when booted from a DASD.

Format

```
noinitrd syntax

►►──noinitrd──────────────────────────────────────────────────►◄
```

possible_cpus

Usage

Specifies the number of maximum possible and usable CPUs that Linux can add to the system. If the possible_cpus parameter is set, the additional_cpus parameter is ignored. See also "additional_cpus" on page 350 and "maxcpus" on page 356.

Format

```
┌─ possible_cpus syntax ──────────────────────────────────────────┐
│                                                                   │
│  ►►──possible_cpus=<number>────────────────────────────────►◄    │
│                                                                   │
└───────────────────────────────────────────────────────────────┘
```

Examples

```
possible_cpus=8
```

ramdisk_size

Usage

Specifies the size of the ramdisk in kilobytes.

Format

---

ramdisk_size syntax

```
►►──ramdisk_size=<size>──────────────────────────────────────◄
```

Examples

```
ramdisk_size=32000
```

ro

Usage

Mounts the root file system read-only.

Format

```
┌─ ro syntax ──────────────────────────────────────────────┐
│                                                          │
│  ▶▶──ro──────────────────────────────────────────▶◀     │
│                                                          │
└──────────────────────────────────────────────────────────┘
```

root

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
```

vmhalt

Usage

Specifies a command to be issued to CP after a system halt. This command is only applicable if the system runs as a VM guest.

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.

vmpoff

Usage

Specifies a command to be issued to CP after a system power off. This command is only applicable if the system runs as a VM guest.

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.

Appendix A. Booting Linux

This chapter provides a general overview of how to boot Linux in an LPAR or as a z/VM guest.

IPL and booting

On System z, you usually start booting Linux by performing an Initial Program Load (IPL). Figure 60 summarizes the main steps.
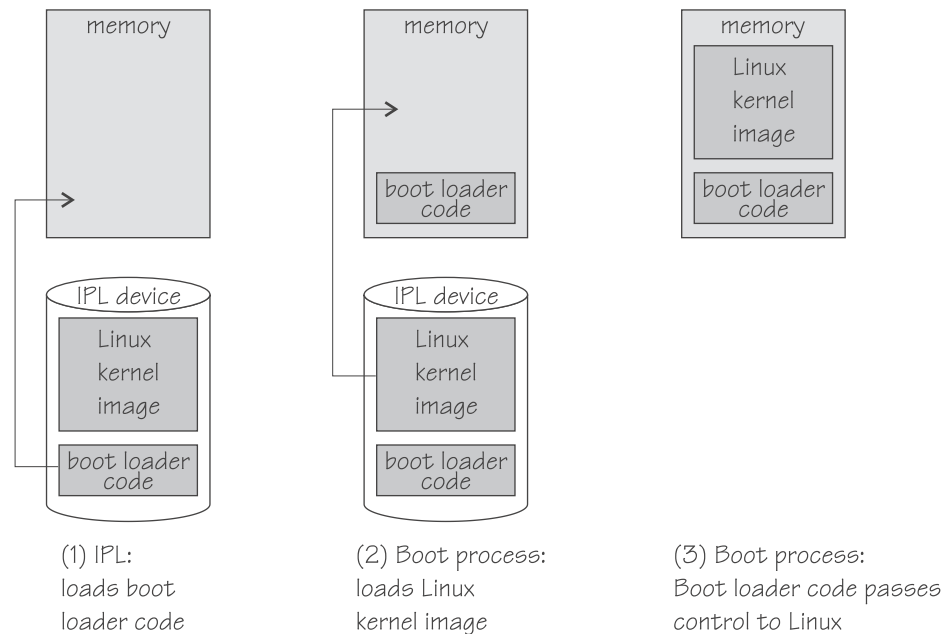
Figure 60. IPL and boot process

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 CD-ROM or from an FTP server" on page 376).

Apart from starting a boot process, an IPL can also be used for:

- Writing out system storage (dumping)

  Refer to *Linux on System z9 and zSeries Using the Dump Tools* 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* for more information on DCSSs.

You can find the latest copies of these documents on developerWorks at:

`ibm.com/developerworks/linux/linux390/october2005_documentation.html`

365

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 "zipl – zSeries initial program loader" on page 330 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 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 is to run as a VM guest, 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 33 provides an overview of the possibilities:

Table 33. Boot media

	DASD	tape	SCSI	VM reader	CD-ROM/FTP
VM guest	✔	✔	✔	✔	
LPAR	✔	✔	✔		✔

DASDs, tapes on channel-attached tape devices, and SCSI disks that are attached through an FCP channel can be used for both LPAR and VM guests. The VM reader is available only in a 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

If you use **zipl** to prepare a DASD or SCSI boot device, you can define a menu configuration. A boot device with a menu configuration can hold the code for multiple boot configurations. For SCSI devices, 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 VM" on page 369 and "Example for a DASD menu configuration (LPAR)" on page 375). For menu configurations on SCSI devices, you need to know the configuration numbers without being able to display the menus.

See "Menu configurations" on page 344 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 (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

You can obtain the kernel image from a Linux 2.6 distribution for System z.

Alternatively, you can compile your own kernel. You can find the System z specific patches and OCO modules on developerWorks at:

`ibm.com/developerworks/linux/linux390/october2005_recommended.html`

Important

Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Boot loader code

A kernel image is usually compiled to contain boot loader code for a particular boot device. For example, there are Linux configuration menu options to compile boot loader code for tape or for the VM reader into the kernel image.

If your kernel image does not include any boot loader code or 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.

You can 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 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 "zipl – zSeries initial program loader" on page 330 or refer to the **zipl** and zipl.conf man pages for details.

If you are using a menu configuration on a DASD boot device, you can display the menu and provide additional kernel parameters as you select a boot configuration.

The following kernel parameters are typically used for booting Linux on System z:

conmode=*<mode>*, **condev=***<cuu>*, and **console=***<name>*
> to set up the Linux console. See "Console kernel parameter syntax" on page 214 for details.

dasd=<devices>
>
> to set specific DASDs online during the boot process. You need to specify this parameter if a DASD is required for the boot process (for example, as the boot device).

noinitrd
>
> to suppress an initial RAM disk. Specify this parameter if your boot configuration includes an initial RAM disk but you do not want to use it.

ramdisk_size=<size>
>
> to specify the size of the initial RAM disk.

ro to mount the root file system read-only.

root=<rootdevice>
>
> to specify the device to be mounted as the root file system.

zfcp.device=<device_bus_id>,<wwpn>,<fcp_lun>
>
> is required if a SCSI device is required for IPL (for example, as the root disk). See "Device driver kernel parameters" on page 54 for 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. If your image contains all files, programs, and modules that are needed for booting, you do not need an initial RAM disk.

Distributions often provide specific RAM disk images to go with their kernel images.

Booting a z/VM Linux guest

You boot a z/VM Linux guest by issuing CP commands from a guest CMS session.

This section provides summary information for booting Linux in a VM guest. For more detailed information on z/VM guest environments for Linux refer to Redpaper *Building Linux Systems under IBM VM* at `ibm.com/redbooks/redpapers/pdfs/redp0120.pdf`.

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 (optional — required only if the kernel image has not been compiled for booting from tape)

   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 session with the VM guest where you want to boot Linux.
2. Ensure that the boot device is accessible to your VM guest.
3. Ensure that the correct tape is inserted and rewound.
4. Issue a command of this form:

```
#cp i <devno>
```

where *<devno>* is the device number of the boot device as seen by the guest.

Using DASD

Before you start:

- You need a DASD boot device prepared with **zipl** (see "Preparing a boot device" on page 333).

Perform these steps to start the boot process:

1. Establish a CMS session with the VM guest where you want to boot Linux.
2. Ensure that the boot device is accessible to your VM guest.
3. Issue a command of this form:

```
#cp i <devno> loadparm <n>
```

where:

i *<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 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 344 for more details on menu configurations.

Example for a DASD menu configuration on VM

This example illustrates how menu2 in the sample configuration file in Figure 59 on page 346 displays on the VM console:

```
00: zIPL v1.3.0 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 SCSI

Before you start:

• You need a SCSI boot device prepared with **zipl** (see "Preparing a boot device" on page 333).

Perform these steps to start the boot process:

1. Establish a CMS session with the VM guest where you want to boot Linux.
2. Ensure that the FCP channel that provides access to the SCSI boot disk is accessible to your VM guest.
3. Specify the SCSI boot disk's target port and LUN for the LOADDEV environment variable. In conjunction with a menu configuration, you can also specify the boot configuration (boot program in VM terminology) to be used. Issue a command of this form:

```
#cp set loaddev portname <wwpn> lun <lun> bootprog <n>
```

where:

<wwpn>
    is the world wide port name (WWPN) of the target port. Specify the WWPN in hexadecimal format with a blank separating the first 8 from the final 8 digits.

<lun>
    is the LUN of the SCSI boot disk. Specify the LUN in hexadecimal format with a blank separating the first 8 from the final 8 digits.

<n>
    in conjunction with a menu configuration, *<n>* is the configuration number that identifies which boot configuration is to be used. Omitting the bootprog parameter or specifying the value "0" selects the menu's default configuration.

See "Menu configurations" on page 344 for more details on menu configurations.

Examples:

- For a WWPN 0x5005076300c20b8e and a LUN 0x5241000000000000:

```
#cp set loaddev portname 50050763 00c20b8e lun 52410000 00000000
```

- To select a configuration with configuration number "2" from a menu configuration for a WWPN 0x5005076300c20b8e and a LUN 0x5242000000000000:

```
#cp set loaddev portname 50050763 00c20b8e lun 52420000 00000000 bootprog 2
```

4. Issue a command of this form:

   Example:

   ```
   #cp i <devno>
   ```

   where *<devno>* is the device number of the FCP channel that provides access to the SCSI boot disk.

Using the VM reader

This section provides a summary of how to boot Linux from a VM reader. For more details refer to Redpaper *Building Linux Systems under IBM VM* at `ibm.com/redbooks/redpapers/pdfs/redp0120.pdf`.

Before you start:

You need the following files, all in record format "fixed 80":
- Linux kernel image with built-in VM reader boot loader code
- Kernel parameters (optional)
- Initial RAM disk image (optional)

Proceed like this to boot Linux from a 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 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 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:

   ```
   #cp spool pun * rdr
   ```

   c. Use the punch device 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:

```
#cp pun <file_name> <file_type> <file_mode> (noh
```

d. Optionally, ensure that the contents of the reader remain fixed.

```
#cp 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:

```
#cp i 000c clear
```

where 0x000c is the device number of the reader.

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 333).
- 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. Click the Groups icon in the "Views" area of the HMC to display the "Groups Work Area" (Figure 61 on page 373).
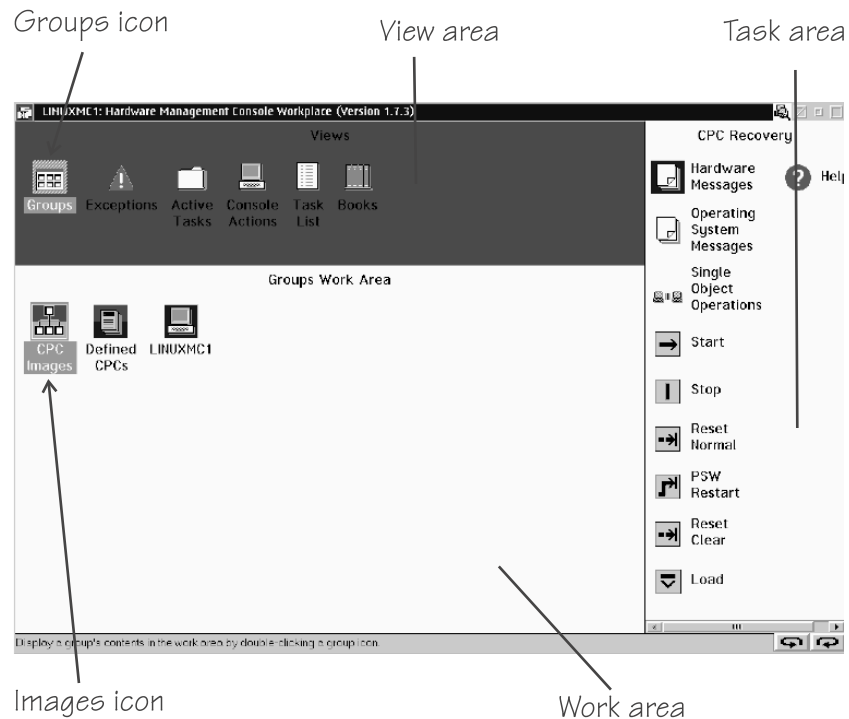
Figure 61. Groups Work Area on the HMC

2. Click the Images icon in the "Groups Work Area" to display the "CPC Images Work Area" with all defined images (Figure 62).
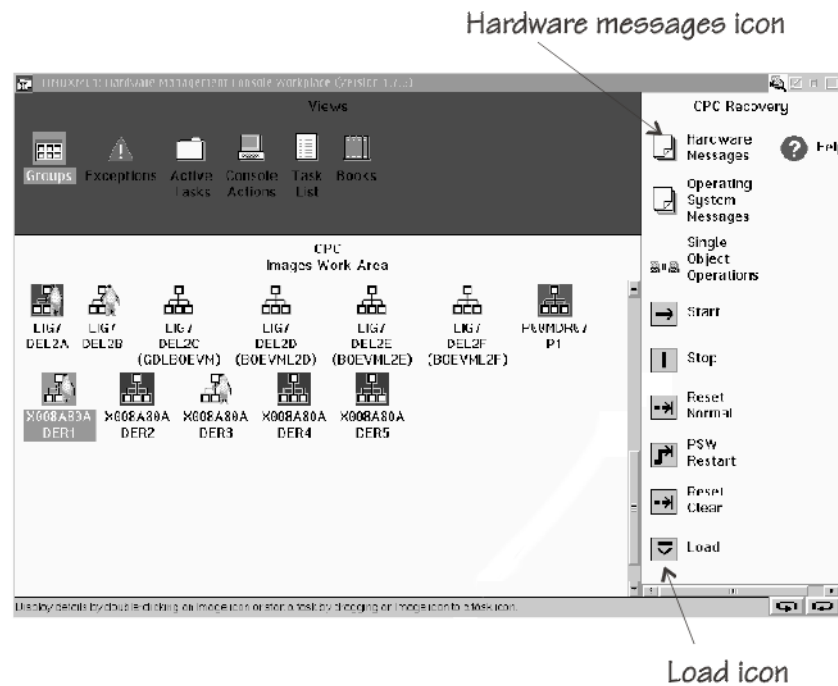
Figure 62. CPC Images Work Area on the HMC

3. Select the image you want to boot.
4. Click the Load icon in the task area to display the Load panel.
5. Proceed according to your boot device.

For booting from tape:

    a. Select **Load type** "Normal".
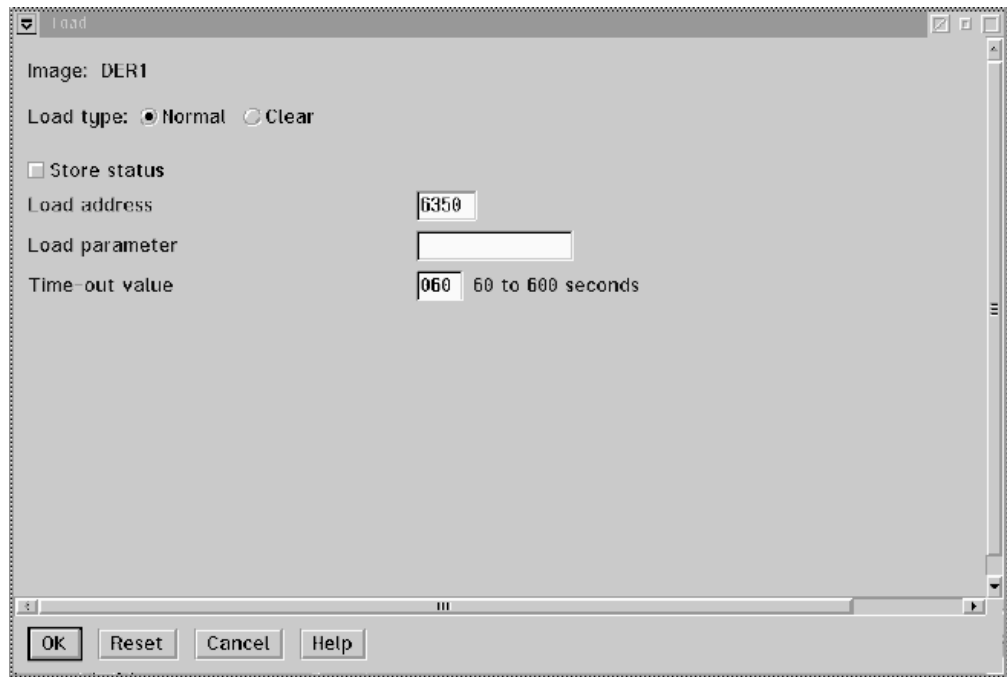
Figure 63. Load panel for booting from DASD or tape

    b. Type the device number of the tape boot device in the **Load address** field.

For booting from DASD:

    a. Select **Load type** "Normal" (see Figure 63).

    b. Type 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, type 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 375). 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 344 for more details on menu configurations.

For booting from a SCSI disk:

    a. Select **Load type** "SCSI".
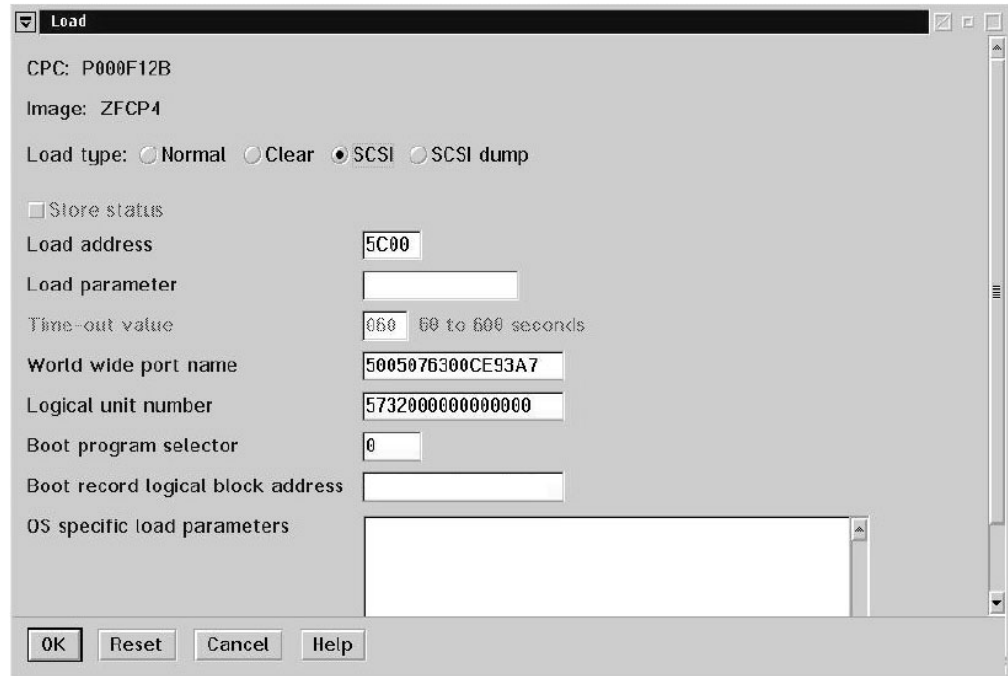
Figure 64. Load panel with SCSI feature enabled — for booting from a SCSI disk

    b. Type the device number of the FCP channel through which the SCSI disk is accessed in the **Load address** field.

    c. Type the WWPN of the SCSI disk in the **World wide port name** field.

    d. Type the LUN of the SCSI disk in the **Logical unit number** field.

    e. If the boot configuration is part of a **zipl** created menu configuration, type the configuration number that identifies your SCSI boot configuration within the menu in the **Boot program selector** field. Configuration number "0" specifies the default configuration.

       See "Menu configurations" on page 344 for more details on menu configurations.

    f. Leave the fields **Load parameter**, **Boot record logical block address**, and **OS specific load parameters** blank.

6. Click **OK** to start the boot process.

Check the output on the preferred console (see "console parameter" on page 215) 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 346 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 CD-ROM 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 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.

Before you start: You need installation data that includes a special file with installation information (with extension "ins") either:

- On a CD-ROM that is inserted in the SE's CD-ROM drive or in the CD-ROM drive of the system where the HMC runs
- In the file system of an FTP server to which you have access

The "ins-file" contains a mapping of the location of installation data in the file system of the CD-ROM or FTP server and the memory locations where the data is to be copied.

The following description is based on accessing the SE remotely from an HMC. If you are working directly from an SE, skip step 4.

1. Click the Groups icon in the "Views" area of the HMC to display the "Groups Work Area" (Figure 65 on page 377).

Groups icon   View area   Task area

Images icon   Work area
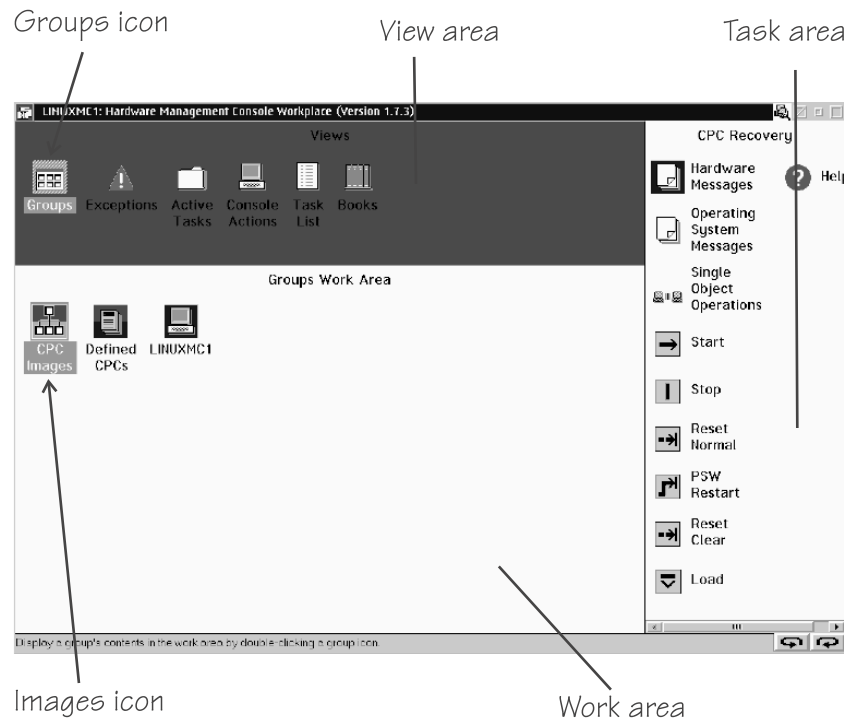
Figure 65. Groups Work Area on the HMC

2. Click the Images icon in the "Groups Work Area" to display the "Defined CPCs Work Area" with all defined images (Figure 66).

Single Object Operations icon
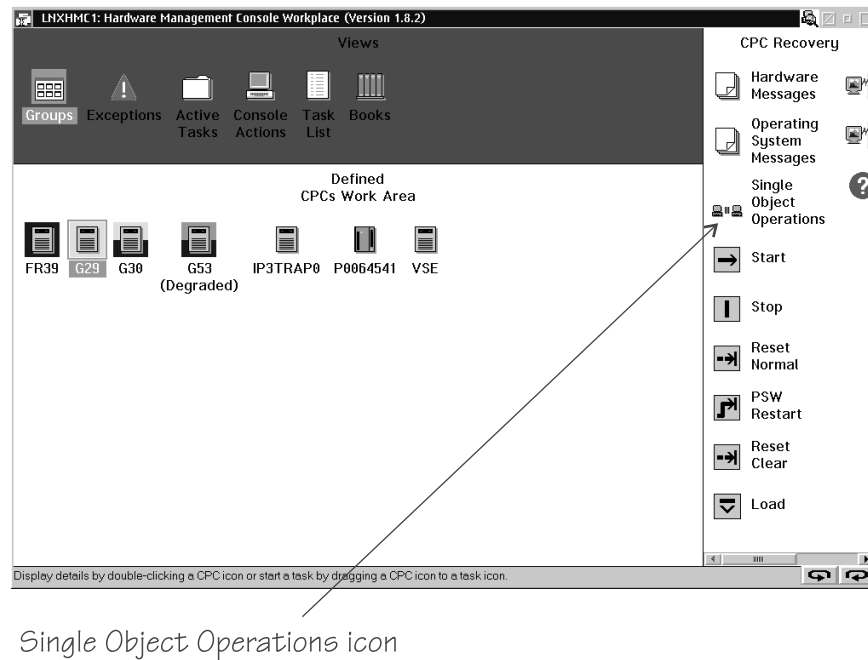
Figure 66. Defined CPCs Work Area on the HMC

3. Select the image you want to IPL.

4. If you are working from an HMC, click the "Single Object Operation" icon (Figure 66 on page 377). This gives you remote access to the SE that controls the image (Figure 67).

Skip this step if you are working directly form the SE.
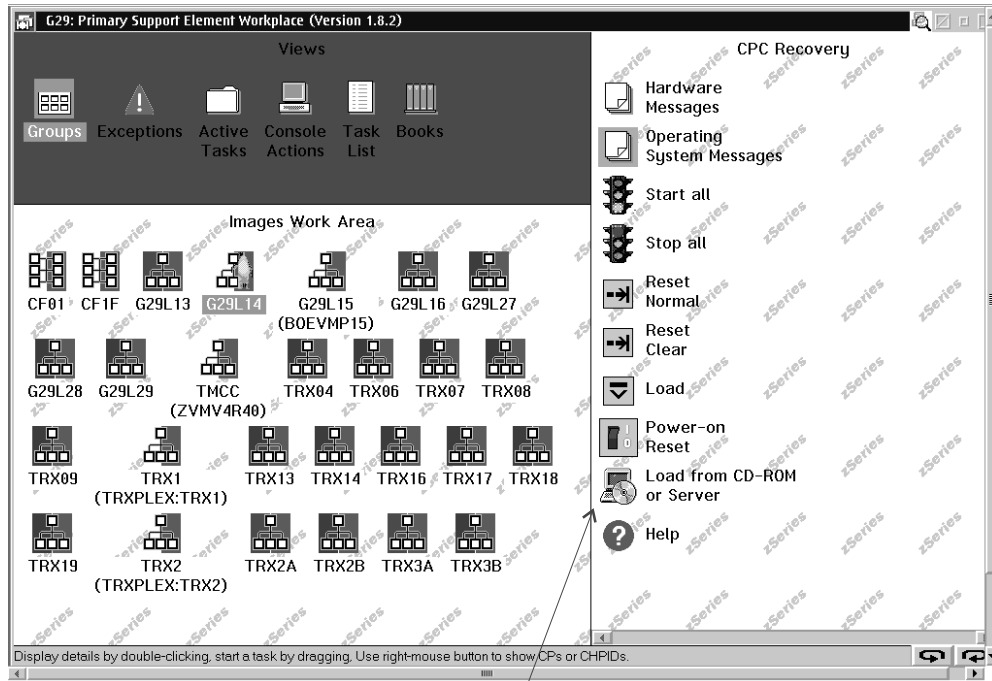
Load from CD ROM or Server icon

Figure 67. Images Work Area on the SE

5. Click the "Load from CD-ROM or Server" icon in the Task Area to display the "Load from CD-ROM or Server" panel (Figure 68 on page 379).
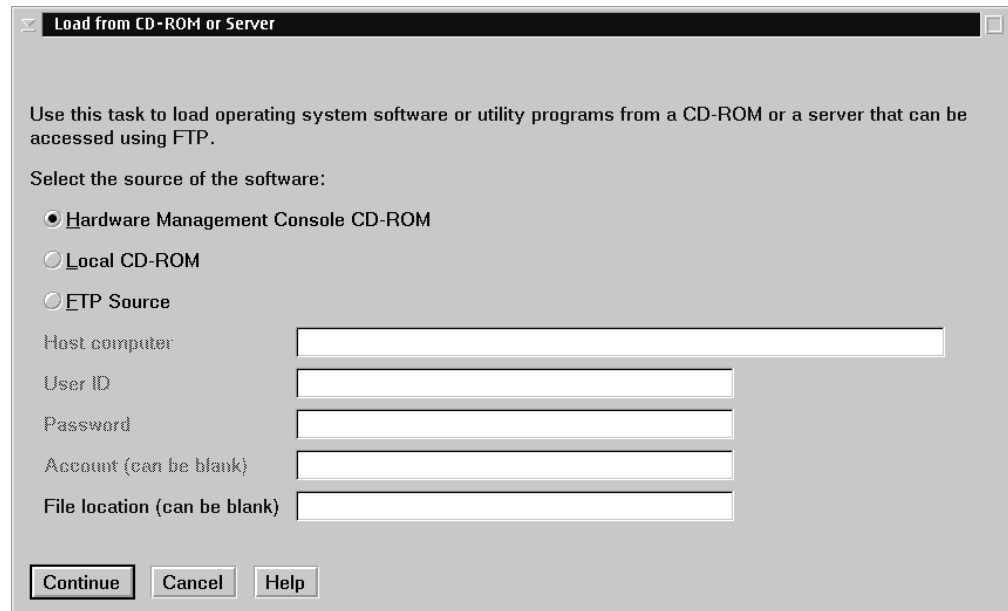
Figure 68. Load from CD-ROM or Server panel

6. Specify the source of the code to be loaded.

   For loading from a CD-ROM drive:

   a. Select the radio button for the CD-ROM you want to use. Select either of:

      - **Hardware Management Console CD-ROM** for the CD-ROM drive on the system where the HMC runs
      - **Local CD-ROM** for the SE's CD-ROM drive

      The CD-ROM drive for the HMC is not available if you are working directly from the SE.

   b. Type the path for the directory where the "ins-file" resides in the **File location** field. You can leave this field blank if the "ins-file" is located in the root directory of the file system on the CD-ROM.

   For loading from an FTP server:

   a. Select the **FTP Source** radio button.

   b. Type the IP address or host name of the FTP server where the install code resides in the **Host computer** entry field.

   c. Type your user ID for the FTP server in the **User ID** entry field.

   d. Type your password for the FTP server in the **Password** entry field.

   e. If required by your FTP server, type your account information in the **Account** entry field.

   f. Type the path for the directory where the "ins-file" resides in the file location entry field. You can leave this field blank if the file resides in the FTP server's root directory.

7. Click **Continue** to display the "Select the software to load" panel (Figure 69 on page 380).
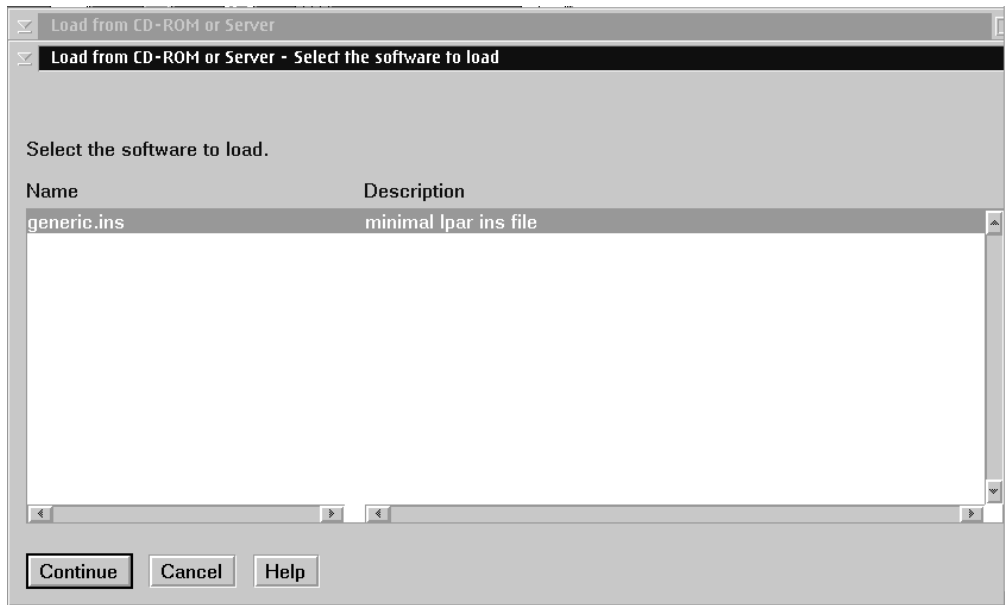
Figure 69. Select the software to load panel

8. Select the "ins-file" to be used.

9. Click **Continue** to start loading Linux.

10. When the load process has completed click the PSW Restart icon.

At this point distribution-specific configuration scripts take over, if present.

Appendix B. Linux diagnose code use

Linux on System z issues several diagnose instructions to the hypervisor (LPAR or z/VM). Table 34 lists all diagnoses which are used by the Linux kernel or a kernel module.

Table 34. Linux diagnoses

Number	Description	Linux use	z/VM	LPAR
0x008	VM/CP command console interface	• The **vmcp** command • The 3215 and 3270 console drivers • The z/VM recording device driver (vmlogrdr) • smsgiucv	X	
0x010	Release pages	CMM	X	
0x044	Voluntary time-slice end	In the kernel for spinlock and udelay	X	X
0x060	Virtual storage size	SALIPL specific boot code.	X	
0x064	Allows Linux to attach a DCSS	The DCSS block device driver (dcssblk), xip, and the MONITOR record device driver (monreader).	X	
0x09c	Voluntary time slice yield	Spinlock.	X	
0x0dc	Monitor stream	The APPLDATA monitor record and the MONITOR stream application support (monwriter).	X	
0x204	LPAR Hypervisor data	The hypervisor file system (hypfs).		X
0x210	Retrieve device information	The common I/O layer and the DASD driver DIAG access method.	X	
0x224	CPU type name table	The hypervisor file system (hypfs).		X
0x250	Block I/O	The DASD driver DIAG access method.	X	
0x258	Page-reference services	In the kernel, for pfault.	X	
0x288	Virtual machine time bomb	The watchdog device driver.	X	
0x308	Re-ipl	Re-ipl and dump code.		X

Linux can fail if you change the privilege class of these diagnoses using the MODIFY diag command in z/VM.

Appendix C. 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 Chapter 21, "Channel measurement facility," on page 231 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" on page 351 and "Changing the list of devices to be ignored" on page 352.

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.

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

asynchronous transfer mode (ATM). A transfer mode in which the information is organized into cells; it is asynchronous in the sense that the recurrence of cells containing information from an individual user is not necessarily periodic. ATM is specified in international standards such as ATM Forum UNI 3.1.

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 zSeries (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 zSeries 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

Glossary

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.

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.

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 on Multiprise 3000 or P/390 or emulated by VM.

FDDI. fiber distributed data interface. An American National Standards Institute (ANSI) standard for a 100-Mbps LAN using optical fiber cables.

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

G3, G4, G5 and G6. The generation names of the S/390 CMOS based product family.

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

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.

IPX. Internetwork Packet Exchange. (1) The network protocol used to connect Novell servers, or any workstation or router that implements IPX, with other workstations. Although similar to the Internet Protocol (IP), IPX uses different packet formats and terminology.

IPX address. The 10-byte address, consisting of a 4-byte network number and a 6-byte node address, that is used to identify nodes in the IPX network. The node address is usually identical to the medium access control (MAC) address of the associated LAN adapter.

IUCV. inter-user communication vehicle. A VM facility for passing data between virtual machines and 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

LAN. local area network.

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 **http://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 a System z9 or zSeries.

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.

Multiprise. An enterprise server of the S/390 family.

N

NIC. network interface card. The physical interface between the IBM mainframe and the network.

O

OCO. Object-code only. A loadable module supplied by IBM without the associated source code.

OS. operating system. (1) Software that controls the execution of programs. An operating system may provide services such as resource allocation, scheduling, input/output control, and data management. (2) A set of programs that control how the system works. (3) The software that deals with the most basic operations that a computer performs.

OSA-2. Open Systems Adapter-2. A common System z network interface feature

OSA-Express. Abbreviation for Open Systems Adapter-Express networking features. These include 10 Gigabit Ethernet, Gigabit Ethernet, Fast Ethernet, Token Ring, and ATM.

OSPF. open shortest path first. A function used in route optimization in networks.

Glossary

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

S/390. The predecessor of System z.

SA/SE. stand alone support element. See SE.

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

TCP. transmission control protocol. A communications protocol used in the Internet and in any network that follows the Internet Engineering Task Force (IETF) standards for internetwork protocol. TCP provides a reliable host-to-host protocol between hosts in packet-switched communications networks and in interconnected systems of such networks. It uses the Internet Protocol (IP) as the underlying protocol.

TCP/IP. transmission control protocol/internet protocol. (1) The Transmission Control Protocol and the Internet Protocol, which together provide reliable end-to-end connections between applications over interconnected networks of different types. (2) The suite of transport and application protocols that run over the Internet Protocol.

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.

Token Ring. (1) According to IEEE 802.5, network technology that controls media access by passing a token (special packet or frame) between media-attached stations. (2) A FDDI or IEEE 802.5 network with a ring topology that passes tokens from one attaching ring station (node) to another.

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 VM, a guest whose real memory (virtual from a VM perspective) corresponds to the real memory of VM.

V=V. In VM, a guest whose real memory (virtual from a VM perspective) corresponds to virtual memory of 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

zSeries. The family of IBM enterprise servers that demonstrate outstanding reliability, availability, scalability, security, and capacity in today's network computing environments.

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.

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

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:
- Common User Access
- developerWorks

389

- ECKD
- Enterprise Storage Server
- ESCON
- @server
- FICON
- HiperSockets
- IBM
- Multiprise
- OS/390
- RAMAC
- S/390
- System/390
- System z9
- TotalStorage
- VSE/ESA
- z9
- z/Architecture
- z/OS
- z/VM
- zSeries

Linux is a 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.

Other company, product, and service names may be trademarks or service marks of others.

Index

Special characters

/sys, mount point   xviii
*ACCOUNT, VM record   185
*LOGREC, VM record   185
*SYMPTOM, VM record   185

Numerics

10 Gigabit Ethernet   95
1000Base-T, Ethernet   95
1750, control unit   25
2105, control unit   25
2107, control unit   25
3088, control unit   145, 151
31-bit   xvii
   values for monitor records   176
   zcrypt   226
3215 line-mode terminal   207
3270 emulation   212
3270 line-mode terminal   207
3370, DASD   25
3480 tape drive   75
3490 tape drive   75
3590 tape drive   75
3592 tape drive   75
3880, control unit   25
3990, control unit   25
6310, control unit   25
64-bit   xvii
9336, DASD   25
9343, control unit   25
9345, DASD   25

A

access control
   FCP LUN   53
access_denied
   zfcp attribute (port)   64
   zfcp attribute (SCSI device)   67, 68
access_shared
   zfcp attribute   68
ACCOUNT, VM record   185
add_hhlen, qeth attribute   116
add, DCSS attribute   160
additional_cpus=, kernel parameter   350
Address Resolution Protocol
   See ARP
AgentX protocol   237
alias
   DASD attribute   45
alias device   45
all_rings, value for qeth broadcast_mode attribute   115
AP
   devices   9
API
   FC-HBA   51

appldata_mem, kernel module   173
appldata_net_sum, kernel module   173
appldata_os, kernel module   173
APPLDATA, monitor stream   173
ARP   100
   proxy ARP   123
   query/purge OSA-Express ARP cache   303
ATM   95
attributes
   device   11
   for CCW devices   11
   for subchannels   14
   qeth   105
auto-detection
   DASD   37
   LCS   145
   qeth   97
autoconfiguration, IPv6   103
autopurge, z/VM recording attribute   189
autorecording, z/VM recording attribute   188
availability
   common CCW attribute   11
   DASD attribute   42
avg_*, cmf attributes   233

B

base device   45
base name
   network interfaces   5
block device
   tape   75
blocksize, tape attribute   83
boot devices   366
   preparing   330
boot loader code   367
booting Linux   365
broadcast_mode, qeth attribute   115
buffer_count, qeth attribute   116
bus ID   11

C

canonical_macaddr, qeth attribute   114
card_type, qeth attribute   117
case conversion   211
CCW
   channel measurement facility   231
   common attributes   11
   devices   9
   group devices   9
   hotplug events   16
   setting devices online/offline   269
CD-ROM, loading Linux   376
CEX2A (Crypto Express2)   219
CEX2C (Crypto Express2)   219
channel measurement facility   231

Readers' Comments — We'd Like to Hear from You

Linux on System z
Device Drivers, Features, and Commands
November, 2006
Linux Kernel 2.6 - October 2005 stream

Publication No. SC33-8289-02

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.

If you would like a response from IBM, please fill in the following information:

Name   Address

Company or Organization

Phone No.   E-mail address

IBM®

Fold and Tape   **Please do not staple**   Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Information Development
Department 3248
Schoenaicher Strasse 220
71032 Boeblingen
Germany

Fold and Tape   **Please do not staple**   Fold and Tape

Readers' Comments — We'd Like to Hear from You

SC33-8289-02

IBM ®