Device Drivers, Features, and Commands as available with SUSE Linux Enterprise Server 11
Device Drivers, Features, and Commands as available with SUSE Linux Enterprise Server 11
Before using this document, be sure to read the information in "Notices" on page 451.
Contents

About this document ........................................ vii

Part 1. General concepts .................................. 1

  Chapter 1. How devices are accessed by Linux .......... 3
  Chapter 2. Devices in sysfs .......................... 7

Part 2. Storage .............................................. 17

  Chapter 3. DASD device driver .......................... 19
  Chapter 4. SCSI-over-Fibre Channel device driver .... 41
  Chapter 5. Channel-attached tape device driver ....... 65
  Chapter 6. XPRAM device driver ........................ 75

Part 3. Networking ............................ 79

  Chapter 7. qeth device driver for OSA-Express (QDIO) and HiperSockets 81
  Chapter 8. OSA-Express SNMP subagent support .......... 129
  Chapter 9. LAN channel station device driver .......... 139
  Chapter 10. CTCM device driver ........................ 143
  Chapter 11. NETIUCV device driver .................... 153
  Chapter 12. CLAW device driver ........................ 161

Part 4. z/VM virtual server integration .................. 167

  Chapter 13. z/VM concepts ............................. 169
  Chapter 14. Writing kernel APPLDATA records .......... 173
  Chapter 15. Writing application APPLDATA records .... 179
  Chapter 16. Reading z/VM monitor records ............. 183
  Chapter 17. z/VM recording device driver .......... 189
  Chapter 18. z/VM unit record device driver .......... 197
  Chapter 19. z/VM DCSS device driver .............. 199
  Chapter 20. Shared kernel support .................. 209
  Chapter 21. Watchdog device driver ........... 213

© Copyright IBM Corp. 2000, 2009
Glossary ........................................ 453
Index .......................................... 457
About this document

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


This document describes the device drivers, features, and commands available to SUSE Linux Enterprise Server 11 for the control of IBM® System z devices and attachments. Unless stated otherwise, in this book the terms device drivers and features are understood to refer to device drivers and features for SUSE Linux Enterprise Server 11 for System z.

In this document, System z is taken to include IBM System z10™ and System z9®.

Unless stated otherwise, all z/VM® related information in this book is based on the assumption that z/VM 5.2 or later is used.

For more specific information about the device driver structure, see the documents in the kernel source tree at

/usr/src/linux-<version>/Documentation/s390

For what is new, known issues, and frequently asked questions, see the SUSE Linux Enterprise Server 11 release notes at


Using sysfs and YaST

This document describes how to change settings and options for mainframe computers in sysfs. In most cases, changes in sysfs are not persistent. If you need to make your changes persistent, use YaST or other configuration programs supplied with SUSE Linux Enterprise Server. See SUSE Linux Enterprise Server 11 Deployment Guide and SUSE Linux Enterprise Server 11 Administration Guide for details.

How this document is organized

The first part of this document contains general and overview information for the System z device drivers for SUSE Linux Enterprise Server 11 for System z.

Part two contains chapters specific to individual storage device drivers.

Part three contains chapters specific to individual network device drivers.

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

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

Part six contains chapters about device drivers and features that support security aspects of SUSE Linux Enterprise Server 11 for System z.
Part seven contains chapters about device drivers and features that are used in the context of booting and shutting down Linux.

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

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

Who should read this document

Most of the information in this document is intended for system administrators who want to configure SUSE Linux Enterprise Server 11 for System z.

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

The following general assumptions are made about your background knowledge:

- You have an understanding of basic computer architecture, operating systems, and programs.
- You have an understanding of Linux, System z terminology.
- You are familiar with Linux device driver software.
- You are familiar with the System z devices attached to your system.

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 297.

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.
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 \textbf{\[\rightarrow\]} symbol indicates the beginning of a syntax diagram.
- The \textbf{\[\rightarrow\]} symbol, at the end of a line, indicates that the syntax diagram continues on the next line.
- The \textbf{\[\rightarrow\]} symbol, at the beginning of a line, indicates that a syntax diagram continues from the previous line.
- The \textbf{\[\rightarrow\]} symbol indicates the end of a syntax diagram.

Syntax items (for example, a keyword or variable) may be:

- Directly on the line (required)
- Above the line (default).
- Below the line (optional)

If defaults are determined by your system status or settings, they are not shown in the diagram. Instead the rule is described together with the option, keyword, or variable in the list following the diagram.

Case sensitivity

Unless otherwise noted, entries are case sensitive.

Symbols

You must code these symbols exactly as they appear in the syntax diagram

- Asterisk
- Colon
- Comma
- Equal sign
- Hyphen
- Double slash
- Parentheses
- Period
- Add
- Dollar sign

For example:

dasd=0.0.7000-0.0.7fff
Variables
An italicized lowercase word indicates a variable that you must substitute with specific information. For example:

```
- p <interface>
```

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

```
vmhalt=<COMMAND>
```

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

```
<repeat>
```

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

```
<repeat>
```

Defaults
Defaults are above the line. The system uses the default unless you override it. You can override the default by coding an option from the stack below the line. For example:

```
A
    B
    C
```

In this example, `A` is the default. You can override `A` by choosing `B` or `C`.

Required Choices
When two or more items are in a stack and one of them is on the line, you must specify one item. For example:

```
A
    B
    C
```

Here you must enter either `A` or `B` or `C`.

Optional Choice
When an item is below the line, the item is optional. Only one item may be chosen. For example:

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

Finding IBM books

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

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

http://www.ibm.com/shop/publications/order
Part 1. General concepts

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

**Newest version:** You can find the newest version of this book at [ibm.com/developerworks/linux/linux390/documentation_novell_suse.html](http://ibm.com/developerworks/linux/linux390/documentation_novell_suse.html)

**Restrictions:** For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 release notes at [www.novell.com/linux/releasenotes/s390x/SUSE-SLES/11/](http://www.novell.com/linux/releasenotes/s390x/SUSE-SLES/11/)

---

**Chapter 1. How devices are accessed by Linux**
- Device name, device nodes, and major/minor numbers
- Network interfaces

**Chapter 2. Devices in sysfs**
- Device categories
- Devices and device attributes
- Device views in sysfs
- Channel path measurement
- Channel path ID information
- CCW hotplug events
Chapter 1. How devices are accessed by Linux

User space programs access devices through:

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

Device name, device nodes, and major/minor numbers

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

Some major numbers are reserved for particular device drivers, others are dynamically assigned to a device driver when Linux boots. For example, major number 94 is always the major number for DASD devices while the device driver for channel-attached tape devices has no fixed major number. A major number can also be shared by multiple device drivers.

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

Device drivers assign device names to their devices, according to a device driver-specific naming scheme (see, for example, “DASD naming scheme” on page 25). Each device name is associated with a minor number.

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.

SUSE Linux Enterprise Server uses udev to create device nodes for you. There is always a device node that matches the device name used by the kernel and additional nodes might be created by special udev rules. See SUSE Linux Enterprise Server 11 Administration Guide and the udev man page for more details.
Network interfaces

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

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 9).

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

Interface names

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

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

<table>
<thead>
<tr>
<th>Base name</th>
<th>Interface type</th>
<th>Device driver module</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>eth</td>
<td>Ethernet</td>
<td>qeth, lcs</td>
<td>OSA-Express, OSA-Express2</td>
</tr>
<tr>
<td>eth</td>
<td>Ethernet</td>
<td>qeth</td>
<td>OSA-Express3</td>
</tr>
<tr>
<td>osn</td>
<td>ESCON/CDLC bridge</td>
<td>qeth</td>
<td>OSA-Express2, OSA-Express3</td>
</tr>
<tr>
<td>ctc</td>
<td>Channel-to-Channel</td>
<td>ctcm</td>
<td>ESCON® channel card, FICON® channel card</td>
</tr>
<tr>
<td>mpc</td>
<td>Channel-to-Channel</td>
<td>ctcm</td>
<td>ESCON channel card</td>
</tr>
<tr>
<td>claw</td>
<td>CLAW</td>
<td>claw</td>
<td>ESCON channel card</td>
</tr>
</tbody>
</table>

Figure 3. Interfaces
Table 2 summarizes the base names used for the network device drivers for interfaces that are associated with virtual hardware:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Interface type</th>
<th>Device driver module</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsi</td>
<td>HiperSockets™, Guest LAN</td>
<td>qeth</td>
<td>Real HiperSockets or HiperSockets guest LAN</td>
</tr>
<tr>
<td>eth</td>
<td>Guest LAN</td>
<td>qeth</td>
<td>QDIO guest LAN</td>
</tr>
<tr>
<td>ctc</td>
<td>virtual Channel-to-Channel</td>
<td>ctc</td>
<td>virtual CTC/A</td>
</tr>
<tr>
<td>mpc</td>
<td>virtual Channel-to-Channel</td>
<td>ctc</td>
<td>virtual CTC/A</td>
</tr>
<tr>
<td>iucv</td>
<td>IUCV</td>
<td>netiucv</td>
<td>IUCV must be enabled for the VM guest</td>
</tr>
</tbody>
</table>

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

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

Matching devices with the corresponding interfaces

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

How you can keep track of the mapping yourself differs depending on the network device driver. For qeth, you can use the `lsqeth` command (see "lsqeth - List qeth based network devices" on page 374) to obtain a mapping.

After setting a device online, read `/var/log/messages` or issue `dmesg` to find the associated interface name in the messages that are issued in response to the device being set online.

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

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

- Define a network device.
  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 10).
  For some devices, there are attributes that can or need to be set later when the device is online or when the connection is active.

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

- Activate the interface.
  This adds interface properties like IP addresses, MTU, and netmasks to a network interface and makes the network interface available to user space programs.
Chapter 2. Devices in sysfs

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

Device categories

Figure 4 illustrates a part of sysfs.

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

AP devices
are adjunct processors used for cryptographic operations.

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

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

IUCV devices
are devices for virtual connections within an IBM mainframe. IUCV devices do not use the channel subsystem.
Table 3 lists the device drivers that have representation in sysfs:

<table>
<thead>
<tr>
<th>Device driver</th>
<th>Category</th>
<th>sysfs directories</th>
</tr>
</thead>
<tbody>
<tr>
<td>3215 console</td>
<td>CCW</td>
<td>/sys/bus/ccw/drivers/3215</td>
</tr>
<tr>
<td>3270 console</td>
<td>CCW</td>
<td>/sys/bus/ccw/drivers/3270</td>
</tr>
<tr>
<td>DASD</td>
<td>CCW</td>
<td>/sys/bus/ccw/drivers/dasd-eckd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/sys/bus/ccw/drivers/dasd-fba</td>
</tr>
<tr>
<td>SCSI-over-Fibre Channel</td>
<td>CCW</td>
<td>/sys/bus/ccw/drivers/zfcp</td>
</tr>
<tr>
<td>Tape</td>
<td>CCW</td>
<td>/sys/bus/ccw/drivers/tape_34xx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/sys/bus/ccw/drivers/tape_3590</td>
</tr>
<tr>
<td>Cryptographic</td>
<td>AP</td>
<td>/sys/bus/ap/drivers/cex2a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/sys/bus/ap/drivers/cex2c</td>
</tr>
<tr>
<td>DCSS</td>
<td>n/a</td>
<td>/sys/devices/dcssblk</td>
</tr>
<tr>
<td>XPARM</td>
<td>n/a</td>
<td>/sys/devices/system/xpram</td>
</tr>
<tr>
<td>z/VM recording device driver</td>
<td>IUCV</td>
<td>/sys/bus/iucv/drivers/vmlogrdr</td>
</tr>
<tr>
<td>OSA-Express,</td>
<td>CCW group</td>
<td>/sys/bus/ccwgroup/drivers/qeth</td>
</tr>
<tr>
<td>OSA-Express2,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSA-Express3,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HiperSockets (qeth)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCS</td>
<td>CCW group</td>
<td>/sys/bus/ccwgroup/drivers/lcs</td>
</tr>
<tr>
<td>CTCM</td>
<td>CCW group</td>
<td>/sys/bus/ccwgroup/drivers/ctcm</td>
</tr>
<tr>
<td>NETIUCV</td>
<td>IUCV</td>
<td>/sys/bus/iucv/drivers/netiucv</td>
</tr>
<tr>
<td>CLAW</td>
<td>CCW group</td>
<td>/sys/bus/ccwgroup/drivers/claw</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

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

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

**cmb_enable**
enables I/O data collection for the device. See “Enabling, resetting, and switching off data collection” on page 320 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 33).
- **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:
Device views in sysfs

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

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

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

0.n.dddd

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

Device driver view

The device driver view is of the form:

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

where:

<bus> is the device category, for example, ccw or ccwgroup.
<driver> is a name that specifies an individual device driver or the device driver component that controls the device (see Table 3 on page 8).
<device_bus_id> identifies an individual device (see Devices and device attributes on page 9).

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 9).

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> groups all devices created by a particular device driver, and usually matches the name of this 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 is of the form:

/sys/devices/css0/<subchannel>

where:

<subchannel> is a subchannel number with a leading "0.n.", where n is the subchannel set ID.
I/O subchannels show the devices in relation to their respective subchannel sets and subchannels. An I/O subchannel is of the form:

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

where:

- `<subchannel>`
  
  is a subchannel number with a leading “0.n.”, where n is the subchannel set ID.

- `<device_bus_id>`
  
  is a device number with a leading “0.n.”, where n is the subchannel set ID (see “Devices and device attributes” on page 9).

**Examples:**

- This example shows a CCW device with device number 0xb100 that is associated with a subchannel 0x0001.
  
  `/sys/devices/css0/0.0.0001/0.0.b100`

- This example shows a CCW device with device number 0xb200 that is associated with a subchannel 0x0001 in subchannel set 1.
  
  `/sys/devices/css0/0.1.0001/0.1.b200`

- The entries for a group device show as separate subchannels. If a CCW group device uses three subchannels 0x0002, 0x0003, and 0x0004 the subchannel information could be:
  
  `/sys/devices/css0/0.0.0002/0.0.a100`
  `/sys/devices/css0/0.0.0003/0.0.a101`
  `/sys/devices/css0/0.0.0004/0.0.a102`

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

- This example lists the information available for a non-I/O subchannel with which no device is associated:
  
  `ls /sys/devices/css0/0.0.ff00/
  bus driver modalias subsystem type uevent`

**Subchannel attributes**

Subchannels have two common attributes:

**type**

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

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

**modalias**

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

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

Apart from the bus ID of the attached device, I/O subchannel directories typically contain these attributes:
chpids is a list of the channel-path identifiers (CHPIDs) through which the device is connected. See also "Channel path ID information" on page 14.

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

## Channel path measurement

In sysfs, an attribute is created for the channel subsystem:

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

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

- **0**: Deactivates the measurement facility and remove the measurement-related attributes for the channel paths. No action if measurements are not active.
- **1**: Attempts to activate the measurement facility and create the measurement-related attributes for the channel paths. No action if measurements are already active.

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

Two sysfs attributes are added for each channel path object:

- **cmg**: Specifies the channel measurement group or unknown if no characteristics are available.
- **shared**: Specifies whether the channel path is shared between LPARs or unknown if no characteristics are available.

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

- **measurement**: A binary sysfs attribute that contains the extended channel-path measurement data for the channel path. It consists of eight 32-bit values and must always be read in its entirety, or 0 will be returned.
- **measurement_chars**: A binary sysfs attribute that is either empty, or contains the channel measurement group dependent characteristics for the channel path, if the channel measurement group is 2 or 3. If not empty, it consists of five 32-bit values.

### Examples

- To turn measurements on issue:
  ```
  # echo 1 > /sys/devices/css0/cm_enable
  ```

- To turn measurements off issue:
  ```
  # echo 0 > /sys/devices/css0/cm_enable
  ```
Channel path ID information

All CHPIDs that are known to Linux are shown alongside the subchannels in the 
/sys/devices/css0 directory. The directories that represent the CHPIDs have the 
form:
/sys/devices/css0/chp0.<chpid>

where <chpid> is a two digit hexadecimal CHPID.

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

Setting a CHPID logically online or offline

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

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

To set a CHPID logically online, set its status attribute to “online” by writing the 
value “on” to it. To set a CHPID logically offline, set its status attribute to “offline” by 
writing “off” to it. Issue a command of this form:

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

where:

<CHPID> is a two digit hexadecimal CHPID.
<value> is either “on” or “off”.

Examples

- To set a CHPID 0x4a logically offline issue:

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

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

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

- To set the same CHPID logically online issue:

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

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

```
# cat /sys/devices/css0/chp0.4a/status
online
```
Configuring a CHPID on LPAR

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

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

To configure a CHPID, set its configure attribute by writing the value “1” to it. To deconfigure a CHPID, set its configure attribute by writing “0” to it. Issue a command of this form:

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

where:

- `<CHPID>` is a two digit hexadecimal CHPID.
- `<value>` is either “1” or “0”.

To query and set the configure value using commands, see “chchp - Change channel path status” on page 336 and “lschp - List channel paths” on page 367.

Examples

- To set a channel path with the ID 0x40 to standby issue:
  ```sh
  # echo 0 > /sys/devices/css0/chp0.40/configure
  ```

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

- To read the configure attribute to confirm that the channel path has been set to standby issue:
  ```sh
  # cat /sys/devices/css0/chp0.40/configure
  0
  ```

- To set the same CHPID to configured issue:
  ```sh
  # echo 1 > /sys/devices/css0/chp0.40/configure
  ```

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

- To read the status attribute to confirm that the CHPID has been set to configured issue:
  ```sh
  # cat /sys/devices/css0/chp0.40/configure
  1
  ```

CCW hotplug events

A hotplug event is generated when a CCW device appears or disappears with a machine check. The hotplug events provide the following variables:

- **CU_TYPE** for the control unit type of the device that appeared or disappeared.
**CU_MODEL** for the control unit model of the device that appeared or disappeared.

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

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

**MODALIAS** for the module alias of the device that appeared or disappeared. The module alias is the same value that is contained in `/sys/devices/css0/<subchannel_id>/<device_id>/modalias` and is of the format

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

Hotplug events can be used, for example, for:
- Automatically setting devices online as they appear
- Automatically loading driver modules for which devices have appeared

For information on the device driver modules see `/lib/modules/<kernel_version>/modules.ccwmap`. This file is generated when you install the Linux kernel (version `<kernel_version>`).
Part 2. Storage

This part describes the storage device drivers for SUSE Linux Enterprise Server 11 for System z.

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

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 release notes at
www.novell.com/linux/releasenotes/s390x/SUSE-SLES/11/

Chapter 3. DASD device driver ................................................. 19
Features .............................................................................. 19
What you should know about DASD ........................................ 20
Setting up the DASD device driver ........................................... 28
Working with the DASD device driver ....................................... 31

Chapter 4. SCSI-over-Fibre Channel device driver ...................... 41
Features .............................................................................. 41
What you should know about zfcp ......................................... 41
Setting up the zfcp device driver .......................................... 46
Working with the zfcp device driver ....................................... 48
Scenario ............................................................................. 61
API provided by the zfcp HBA API support .............................. 62

Chapter 5. Channel-attached tape device driver ......................... 65
Features .............................................................................. 65
What you should know about channel-attached tape devices .... 65
Setting up the tape device driver .......................................... 69
Working with the tape device driver ....................................... 69
Scenario: Using a tape block device ....................................... 73

Chapter 6. XPRAM device driver ............................................... 75
XPRAM features ..................................................................... 75
What you should know about XPRAM .................................... 75
Setting up the XPRAM device driver ....................................... 76
The DASD device driver provides access to all real or emulated Direct Access Storage Devices (DASD) that can be attached to the channel subsystem of an IBM mainframe. DASD devices include a variety of physical media on which data is organized in blocks or records or both. The blocks or records in a DASD can be accessed for read or write in random order.

Traditional DASD devices are attached to a control unit that is connected to a mainframe I/O channel. Today, these real DASD have been largely replaced by emulated DASD, such as the volumes of the IBM System Storage™ DS8000™ Turbo, or the volumes of the IBM System Storage DS6000™. These emulated DASD are completely virtual and the identity of the physical device is hidden.

SCSI disks attached through a System z FCP adapter are not classified as DASD. They are handled by the zfcp driver (see Chapter 4, “SCSI-over-Fibre Channel device driver,” on page 41).

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>
<thead>
<tr>
<th>Device format</th>
<th>Control unit type</th>
<th>Device type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECKD (Extended Count Key Data)</td>
<td>1750</td>
<td>3380 and 3390</td>
</tr>
<tr>
<td></td>
<td>2107</td>
<td>3380 and 3390</td>
</tr>
<tr>
<td></td>
<td>2105</td>
<td>3380 and 3390</td>
</tr>
<tr>
<td></td>
<td>3990</td>
<td>3380 and 3390</td>
</tr>
<tr>
<td></td>
<td>9343</td>
<td>9345</td>
</tr>
<tr>
<td></td>
<td>3880</td>
<td>3390</td>
</tr>
<tr>
<td>FBA (Fixed Block Access)</td>
<td>6310</td>
<td>9336</td>
</tr>
<tr>
<td></td>
<td>3880</td>
<td>3370</td>
</tr>
</tbody>
</table>

All models of the specified control units and device types listed in Table 4 work with the DASD device driver. Check the storage support statement for what works with SUSE Linux Enterprise Server 11 for System z.

- The DASD device driver is also known to work with these devices:
  - RAMAC
  - RAMAC RVA
- SUSE Linux Enterprise Server 11 for System z provides a disk format with up to three partitions per disk. See "System z compatible disk layout" on page 21 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 used for DASD devices.

The IBM label partitioning scheme

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

Traditional mainframe operating systems (such as, z/OS®, z/VM, and z/VSE™) expect a standard DASD format. In particular, the format of the first two tracks of a DASD is defined by this standard and includes System z IPL, label, and for some layouts VTOC records. Partitioning schemes for platforms other than System z generally do not preserve these mainframe specific records.

SUSE Linux Enterprise Server 11 for System z includes the IBM label partitioning scheme that preserves the System z IPL, label, and VTOC records. This partitioning scheme allows Linux to share a disk with other mainframe operating systems. For example, a traditional mainframe operating system could handle backup and restore for a partition that is used by Linux.

The following sections describe the layouts that are supported by the IBM label partitioning scheme:

- "System z compatible disk layout" on page 21
- "Linux disk layout" on page 23
- "CMS disk layout" on page 24

DASD partitions

A DASD partition is a contiguous set of DASD blocks that is treated by Linux as an independent disk and by the traditional mainframe operating systems as a data set. The compatible disk layout allows for up to three partitions on a DASD. The Linux disk layout and the CMS disk layout both permit a single partition only.

There are several reasons why you might want to have multiple partitions on a DASD, for example:

- **Limit data growth.** Runaway processes or undisciplined users can consume disk space to an extend that the operating system runs short of space for essential operations. Partitions can help to isolate the space that is available to particular processes.
- **Encapsulate your data.** If a file system gets damaged, this damage is likely to be restricted to a single partition. Partitioning can reduce the scope of data damage.

Recommendations:

- Use `fdasd` to create or alter partitions. If you use another partition editor, it is your responsibility to ensure that partitions do not overlap. If they do, data damage will occur.
- Leave no gaps between adjacent partitions to avoid wasting space. Gaps are not reported as errors, and can only be reclaimed by deleting and recreating one or more of the surrounding partitions and rebuilding the file system on them.
A disk need not be partitioned completely. You may begin by creating only one or two partitions at the start of your disk and convert the remaining space to a partition later (perhaps when performance measurements have given you a better value for the block size).

There is no facility for moving, enlarging or reducing partitions, because fdasd has no control over the file system on the partition. You only can delete and recreate them. Changing the partition table results in loss of data in all altered partitions. It is up to you to preserve the data by copying it to another medium.

**System z compatible disk layout**

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

*Figure 5* illustrates a DASD with the compatible disk layout.

<table>
<thead>
<tr>
<th>IPL records</th>
<th>VOL1</th>
<th>VTOC</th>
<th>/dev/dasd&lt;x&gt;1</th>
<th>/dev/dasd&lt;x&gt;2</th>
<th>/dev/dasd&lt;x&gt;3</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/dasd&lt;x&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 5. 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 25).

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 347) to format a disk with the compatible disk layout. You use the `fdasd` command (see “fdasd – Partition a DASD” on page 359) 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**

Like other System z operating systems, SUSE Linux Enterprise Server 11 for System z uses a Volume Table Of Contents (VTOC). The VTOC contains pointers to the location of every data set on the volume. These data sets form the Linux partitions.

The VTOC is located in the second track (cylinder 0, track 1). It contains a number of 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, z/VM or z/VSE.

The VTOC can be displayed with standard System z tools such as VM/DITTO. A Linux DASD with physical device number 0x0193, volume label “LNX001”, and three partitions might be displayed like this:
In Linux, this DASD might appear so:

```bash
# ls -l /dev/dasda*
brw-rw---- 1 root disk 94, 0 Jan 27 09:04 /dev/dasda
brw-rw---- 1 root disk 94, 1 Jan 27 09:04 /dev/dasda1
brw-rw---- 1 root disk 94, 2 Jan 27 09:04 /dev/dasda2
brw-rw---- 1 root disk 94, 3 Jan 27 09:04 /dev/dasda3
```

where dasda represent the whole DASD and dasda1, dasda2, and dasda3 represent the individual partitions.

**Linux disk layout**

You can only format ECKD-type DASD with the Linux disk layout. Figure 6 illustrates a disk with the Linux disk layout.

![Figure 6. Linux disk layout](image)

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/dasdx`, where `<x>` can be one to four letters that identify the individual DASD (see "DASD naming scheme" on page 25). Linux can access the partition as `/dev/dasdx1`.

You use the `dasdfmt` command (see "dasdfmt - Format a DASD" on page 347) 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 7 illustrates two variants of the CMS disk layout.

The variant in the upper part of Figure 7 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 7 illustrates a CMS reserved volume. DASD like this have been reserved by a CMS RESERVE fn ft fm command. In addition to the IPL records and the volume label, DASD with the CMS disk layout also have CMS metadata. The CMS reserved file serves as the Linux partition.

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

Addressing the device and partition is the same for both variants. Linux can address the device as a whole as /dev/dasd<x>, where <x> can be one to four letters that identify the individual DASD (see "DASD naming scheme" on page 25). Linux can access the partition as /dev/dasd<x>1.

"Enabling DIAG calls to access DASDs" on page 34 describes how you can enable DIAG.
Disk layout summary

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

Table 5. Disk layout summary

<table>
<thead>
<tr>
<th>Disk Layout</th>
<th>Device format</th>
<th>DIAG call support (z/VM only)</th>
<th>Maximum number of partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ECKD</td>
<td>FBA</td>
<td></td>
</tr>
<tr>
<td>CDL</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>LDL</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CMS (z/VM only)</td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

DASD naming scheme

The DASD device driver uses the major number 94. For each configured device it uses 4 minor numbers:

- The first minor number always represents the device as a whole, including IPL, VTOC and label records.
- The remaining three minor numbers represent the up to three partitions.

With 1,048,576 (20-bit) available minor numbers, the DASD device driver can address 262,144 devices.

The DASD device driver uses a device name of the form dasd<x> for each DASD. In the name, <x> is one to four lowercase letters. Table 6 shows how the device names map to the available minor numbers.

Table 6. Mapping of DASD names to minor numbers

<table>
<thead>
<tr>
<th>Name for device as a whole</th>
<th>Minor number for device as a whole</th>
<th>Number of devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td>To</td>
<td>From</td>
</tr>
<tr>
<td>dasda</td>
<td>dasdz</td>
<td>0</td>
</tr>
<tr>
<td>dasdaa</td>
<td>dasdzz</td>
<td>104</td>
</tr>
<tr>
<td>dasdaaa</td>
<td>dasdzzz</td>
<td>2808</td>
</tr>
<tr>
<td>dasdaaaa</td>
<td>dasdnwtl</td>
<td>73112</td>
</tr>
<tr>
<td>Total number of devices:</td>
<td>262,144</td>
<td></td>
</tr>
</tbody>
</table>

The DASD device driver also uses a device name for each partition. The name of the partition is the name of the device as a whole with a 1, 2, or 3 appended to identify the first, second, or third partition. The three minor numbers following the minor number of the device as a whole are the minor number for the first, second, and third partition.

Examples:

- “dasda” refers to the whole of the first disk in the system and “dasda1”, “dasda2”, and “dasda3” to the three partitions. The minor number for the whole device is 0. The minor numbers of the partitions are 1, 2, and 3.
- “dasdz” refers to the whole of the 101st disk in the system and “dasdz1”, “dasdz2”, and “dasdz3” to the three partitions. The minor number for the whole device is 100. The minor numbers of the partitions are 101, 102, and 103.
"dasdaa" refers to the whole of the 102nd disk in the system and "dasdaa1", "dasdaa2", and "dasdaa3" to the three partitions. The minor number for the whole device is 104. The minor numbers of the partitions are 105, 106, and 107.

**DASD device nodes**

SUSE Linux Enterprise Server uses udev to create multiple device nodes for each DASD that is online.

**Device nodes based on device names**

udev creates device nodes that match the device names used by the kernel. These standard device nodes have the form /dev/<name>.

The mapping between standard device nodes and the associated physical disk space can change, for example, when you reboot Linux. To ensure that you access the intended physical disk space, you need device nodes that are based on properties that identify a particular DASD.

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

**Device nodes based on bus IDs**

udev creates device nodes of the form

/dev/disk/by-path/ccw-<device_bus_id>

for whole DASD and

/dev/disk/by-path/ccw-<device_bus_id>-part<n>

for the <n>th partition.

**Device nodes based on VOLSERs**

udev creates device nodes of the form

/dev/disk/by-id/ccw-<volser>

for whole DASD and

/dev/disk/by-path/ccw-<volser>-part<n>

for the <n>th partition.

**Device nodes based on file system information**

udev creates device nodes of the form

/dev/disk/by-uuid/<uid>

where <uid> is the UUID for the file system in a partition.

If a file system label has been assigned, udev also creates a node of the form

/dev/disk/by-label/<label>

There are no device nodes for the whole DASD that are based on file system information.

**Additional device nodes**

/dev/disk/by-id contains additional device nodes for the DASD and partitions, that are all based on a device identifier as contained in the uid attribute of the DASD.
Note: When using device nodes that are based on file system information and VOLSER be sure that they are unique for the scope of your Linux instance. This information can be changed by a user or it can be copied, for example when creating a backup disk. If two disks with the same VOLSER or UUID are online to the same Linux instance, the matching device node can point to either of these disks.

Example: For a DASD that is assigned the device name dasdzzz, has two partitions, a device bus-ID 0.0.b100 (device number 0xb100), VOLSER LNX001, and a UUID 6dd6c43d-a792-412f-a651-0031e631caed for the first and f45e955d-741a-4cf3-86b1-380ee5177ac3 for the second partition, udev creates the following device nodes:

For the whole DASD:
- /dev/dasdzzz (standard device node according to the DASD naming scheme)
- /dev/disk/by-path/ccw-0.0.b100
- /dev/disk/by-id/ccw-LNX001

For the first partition:
- /dev/dasdzzz1 (standard device node according to the DASD naming scheme)
- /dev/disk/by-path/ccw-0.0.b100-part1
- /dev/disk/by-id/ccw-LNX001-part1
- /dev/disk/by-uuid/6dd6c43d-a792-412f-a651-0031e631caed

For the second partition:
- /dev/dasdzzz2 (standard device node according to the DASD naming scheme)
- /dev/disk/by-path/ccw-0.0.b100-part2
- /dev/disk/by-id/ccw-LNX001-part2
- /dev/disk/by-uuid/f45e955d-741a-4cf3-86b1-380ee5177ac3

The sections that follow show how such nodes can be used to access a device by device bus-ID or VOLSER, 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 [DASD device nodes](#) on page 26. To assure that you are addressing a device with bus-ID 0.0.b100 you could make substitutions like the following.

Instead of issuing:

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

issue:

```
# fdasd /dev/disk/by-path/ccw-0.0.b100
```
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 21).

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 the Linux kernel needs to be forced to reread the partition tables from disks, for example, by issuing:

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

Examples

The examples in this section assume that udev provides device nodes as described in "DASD device nodes" on page 26. 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/disk/by-id/ccw-LNX001
```

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

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

with these specifications:

```
/dev/disk/by-id/ccw-LNX001-part1 /temp1 ext2 defaults 0 0
/dev/disk/by-id/ccw-LNX001-part2 /temp2 ext2 defaults 0 0
```

Setting up the DASD device driver

This section describes how to load and configure the DASD device driver modules.
**DASD module parameter syntax**

```
modprobe dasd_mod
insmod <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>
```

Where:

**dasd_mod**
loads the device driver base module.

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

You can use the `eer_pages` parameter to determine the number of pages used for internal buffering of error records.

**autodetect**
causes the DASD device driver to allocate device names and the corresponding minor numbers to all DASD devices and set them online during the boot process. See [“DASD naming scheme” on page 25](#) 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.

(erplog)
enables enhanced error recovery processing (ERP) related logging through
syslogd. If erplog is specified for a range of devices, the logging is switched
on during device initialization.

(failfast)
returns “failed” for an I/O operation when the last path to a DASD is lost.
Use this option with caution (see “Switching immediate failure of I/O
requests on or off” on page 37).

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 is
loaded first. 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
```

Table 7 shows the resulting allocation of device names:

<table>
<thead>
<tr>
<th>Name</th>
<th>To access</th>
</tr>
</thead>
<tbody>
<tr>
<td>dasda</td>
<td>device 0.7000 as a whole</td>
</tr>
<tr>
<td>dasda1</td>
<td>the first partition on 0.7000</td>
</tr>
<tr>
<td>dasda2</td>
<td>the second partition on 0.7000</td>
</tr>
<tr>
<td>dasda3</td>
<td>the third partition on 0.7000</td>
</tr>
<tr>
<td>dasdb</td>
<td>device 0.7001 as a whole</td>
</tr>
<tr>
<td>dasdb1</td>
<td>the first partition on 0.7001</td>
</tr>
<tr>
<td>dasdb2</td>
<td>the second partition on 0.7001</td>
</tr>
<tr>
<td>dasdb3</td>
<td>the third partition on 0.7001</td>
</tr>
<tr>
<td>dasdc</td>
<td>device 0.7002 as a whole</td>
</tr>
<tr>
<td>dasdc1</td>
<td>the first partition on 0.7002</td>
</tr>
<tr>
<td>dasdc2</td>
<td>the second partition on 0.7002</td>
</tr>
<tr>
<td>dasdc3</td>
<td>the third partition on 0.7002</td>
</tr>
<tr>
<td>dasdd</td>
<td>device 0.7005 as a whole</td>
</tr>
</tbody>
</table>
### Table 7. Example mapping of device names to devices (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>To access</th>
</tr>
</thead>
<tbody>
<tr>
<td>dasdd1</td>
<td>the first partition on 0.0.7005 (read-only)</td>
</tr>
<tr>
<td>dasdd2</td>
<td>the second partition on 0.0.7005 (read-only)</td>
</tr>
<tr>
<td>dasdd3</td>
<td>the third partition on 0.0.7005 (read-only)</td>
</tr>
<tr>
<td>dasde</td>
<td>device 0.0.7006 as a whole</td>
</tr>
<tr>
<td>dasde1</td>
<td>the first partition on 0.0.7006</td>
</tr>
<tr>
<td>dasde2</td>
<td>the second partition on 0.0.7006</td>
</tr>
<tr>
<td>dasde3</td>
<td>the third partition on 0.0.7006</td>
</tr>
</tbody>
</table>

### 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"
- "Preparing an FBA-type DASD for use" on page 33
- "Accessing DASD by force" on page 33
- "Enabling DIAG calls to access DASDs" on page 34
- "Working with extended error reporting for ECKD" on page 35
- "Switching extended error reporting on and off" on page 36
- "Setting a DASD online or offline" on page 36
- "Enable and disable logging" on page 37
- "Switching immediate failure of I/O requests on or off" on page 37
- "Displaying DASD information" on page 38

### Preparing an ECKD-type DASD for use

This section describes the main steps for enabling an ECKD-type DASD for use by SUSE Linux Enterprise Server 11 for System z.

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

**Before you start:**

- The modules for the base component and the ECKD component of the DASD device driver must have been loaded.
- The DASD device driver must have recognized the device as an ECKD-type device.
- You need to know the device node through which the DASD can be addressed.

Perform these steps to prepare the DASD:

1. **Format the device with the `dasdfmt` command** (see "dasdfmt - Format a DASD" on page 347 for details). The formatting process can take hours for large DASD.

   **Recommendations:**
   - Use the default `-d cd1` 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.
2. Proceed according to your chosen disk layout:
   - If you have formatted your DASD with the Linux disk layout, skip this step and continue with step 3. 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 359 for details).
     
     **Example:** To start the partitioning tool in interactive mode for partitioning a device `/dev/dasdzzz` issue:
     
     ```bash
     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 22).

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

   **Either:**

   Create a file system of your choice. For example, use the Linux `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:**
   
   ```bash
   # 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).

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

   **Example:** To mount a file system in a partition `/dev/dasdzzz1` to a mount point `/mnt` and to enable a swap partition `/dev/dasdzzz2` issue:
   
   ```bash
   # 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:`
Preparing an FBA-type DASD for use

This section describes the main steps for enabling an FBA-type DASD for use by SUSE Linux Enterprise Server 11 for System z.

Note: To access FBA devices, use the DIAG access method (see "Enabling DIAG calls to access DASDs" on page 34 for more information).

Before you start:

- The modules for the base component and the FBA component of the DASD device driver must have been loaded.
- The DASD device driver must have recognized the device as an FBA device.
- You need to know the device bus-ID or the device node through which the DASD can be addressed.

Perform these steps to prepare the DASD:

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

   Either:
   
   Create a file system of your choice. For example, use the Linux `mke2fs` command to create an ext2 file system (refer to the man page for details).
   
   Example: `mke2fs -b 4096 /dev/dasdzzz1`

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

2. Mount the file system to the mount point of your choice in Linux or enable your swap partition.

   Example: To mount a file system in a partition `/dev/dasdzzz1` issue:
   
   ```
   # mount /dev/dasdzzz1 /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 10).

Issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/availability
```
Example: This example shows that a DASD with device bus-ID 0.0.b110 (device number 0xb110) has been externally locked.

```
# cat /sys/bus/ccw/devices/0.0.b110/availability
boxed
```

If the DASD is an ECKD-type DASD and if you know the device bus-ID, you can break the external lock and set the device online. This means that the lock of the external system is broken with the “unconditional reserve” channel command.

**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 lock of a DASD that has already been analyzed see "tunedasd - Adjust DASD performance" on page 421.

**Enabling DIAG calls to access DASDs**

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

- The Linux instance runs as a VM guest.
- The device can be of type ECKD with either LDL or CMS disk layout, or it can be a device of type FBA.
- The module for the DIAG component must be loaded.
- The module for the component that corresponds to the DASD type (dasd_eckd_mod or dasd_fba_mod) must be loaded.
- The DASD is offline.
- The DASD does not represent a parallel access volume alias device.

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 36).

**Note:** When switching between enabled and disabled DIAG calls on FBA-type DASD, first re-initialize the DASD, for example, with CMS format or by overwriting any previous content. Switching without initialization might cause data-integrity problems.

For more details about DIAG see *z/VM CP Programming Services*, SC24-6084.

**Example**

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

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

1. Ensure that the driver is loaded:

   ```
   # modprobe dasd_diag_mod
   ```

2. Identify the sysfs CCW-device directory for the device in question and change to that directory:

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

3. Ensure that the device is offline:

   ```
   # echo 0 > online
   ```

4. Enable the DIAG access method for this device by writing ‘1’ to the use_diag sysfs attribute:

   ```
   # echo 1 > use_diag
   ```

5. Use the online attribute to set the device online:

   ```
   # echo 1 > online
   ```

**Working with extended error reporting for ECKD**

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

**open**

Multiple processes can open the node concurrently. Each process that opens the node has access to the records that are created from the time the node is opened. A process cannot access records that were created before the process opened the node.

**close**

You can close the node as usual.
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.

The poll operation is typically used in conjunction with non-blocking read.

**Switching extended error reporting on and off**

Extended error reporting is turned off by default. To turn extended error reporting on, issue a command of this form:

```
# echo 1 > /sys/bus/ccw/devices/<device bus-id>/eer_enabled
```

where `/sys/bus/ccw/devices/<device bus-id>` represents the device in sysfs.

When it is enabled on a device, a specific set of errors will generate records and may have further side effects. The records are made available via a character device interface.

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

```
# echo 0 > /sys/bus/ccw/devices/<device bus-id>/eer_enabled
```

**Setting a DASD online or offline**

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

Use the `chccwdev` command (["chccwdev - Set a CCW device online" on page 334](#)) 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 SUSE Linux Enterprise Server 11 for System z instance, for example, from VM.
When a DASD is attached, Linux attempts to initialize it according to the DASD device driver configuration. You can then set the device online. You can automate setting dynamically attached devices online by using CCW hotplug events (see “CCW hotplug events” on page 15).

Note
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.

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

- Enable logging during module load using the dasd= parameter. For example, to define a device range (0.0.7000-0.0.7005) and switch on logging, change the parameter line to contain:
  dasd=0.0.7000-0.0.7005(erplog)
- Use the sysfs attribute erplog to switch ERP-related logging on or off. Logging can be enabled for a specific device by writing "1" to the erplog attribute, for example:

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

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

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

Switching immediate failure of I/O requests on or off
By default, a DASD that has lost all paths waits for one of the paths to recover. I/O requests are blocked while the DASD is waiting.

If the DASD is part of a mirror setup, this blocking might cause the entire virtual device to be blocked. You can use the failfast attribute to immediately return I/O requests as failed while no path to the device is available.

Use this attribute with caution and only in setups where a failed I/O request can be recovered outside the scope of a single DASD.

- You can switch on immediate failure of I/O requests when you load the base module of the DASD device driver: For example, to define a device range (0.0.7000-0.0.7005) and enable immediate failure of I/O requests specify:
  dasd=0.0.7000-0.0.7005(failfast)
- You can use the sysfs attribute failfast of a DASD to switch immediate failure of I/O requests on or off. To switch on immediate failure of I/O requests, write "1" to the failfast attribute, for example:

  echo 1 > /sys/bus/ccw/devices/<device_bus_id>/failfast
To switch off immediate failure of I/O requests, write "0" to the failfast attribute, for example:

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

### 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 8. DASD device attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>“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 <em>How to Improve Performance with PAV</em> on developerWorks at <a href="http://ibm.com/developerworks/linux/linux390/documentation_dev.html">ibm.com/developerworks/linux/linux390/documentation_dev.html</a></td>
</tr>
<tr>
<td>discipline</td>
<td>Is the base discipline, ECKD or FBA, that is used to access the DASD. This attribute is read-only. If DIAG is enabled, this attribute might read DIAG instead of the base discipline.</td>
</tr>
<tr>
<td>eer_enabled</td>
<td>“1” if the DASD is enabled for extended error reporting or “0” if it is not enabled (see &quot;Switching extended error reporting on and off&quot; on page 36).</td>
</tr>
<tr>
<td>failfast</td>
<td>“1” if I/O operations are returned as failed immediately when the last path to the DASD is lost. “0” if a wait period for a path to return expires before an I/O operation is returned as failed. (see &quot;Switching immediate failure of I/O requests on or off&quot; on page 37).</td>
</tr>
<tr>
<td>online</td>
<td>“1” if the DASD is online or “0” if it is offline (see &quot;Setting a DASD online or offline&quot; on page 36).</td>
</tr>
<tr>
<td>readonly</td>
<td>“1” if the DASD is read-only “0” if it can be written to. This attribute is ignored for PAV alias devices.</td>
</tr>
</tbody>
</table>
| uid | A device identifier of the form `<vendor>.<serial>.<subsystem_id>.<unit_address>.<minidisk_identifier>` where  
- `<vendor>` is the specification from the vendor attribute.  
- `<serial>` is the serial number of the storage system.  
- `<subsystem_id>` is the ID of the logical subsystem to which the DASD belongs on the storage system.  
- `<unit_address>` is the address used within the storage system to identify the DASD.  
- `<minidisk_identifier>` is an identifier that the z/VM system assigns to distinguish between minidisks on the DASD. This part of the uid is only present if the Linux instance runs as a z/VM guest operating system and if the z/VM version and service level supports this identifier. This attribute is read-only. |
Table 8. DASD device attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use_diag</td>
<td>“1” if DIAG calls are enabled “0” if DIAG calls are not enabled (see &quot;Enabling DIAG calls to access DASDs&quot; on page 34). Do not enable DIAG calls for PAV alias devices.</td>
</tr>
<tr>
<td>vendor</td>
<td>A specification that identifies the manufacturer of the storage system that contains the DASD. This attribute is read-only.</td>
</tr>
</tbody>
</table>

Issue a command of this form to read an attribute:

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

where `<attribute>` is one of the attributes of Table 8 on page 38.

**Example**

The following sequence of commands reads the attributes for a DASD with a device bus-ID 0.0.b100:

```
# cat /sys/bus/ccw/devices/0.0.b100/alias
0
# cat /sys/bus/ccw/devices/0.0.b100/discipline
ECKD
# cat /sys/bus/ccw/devices/0.0.b100/eer_enabled
0
# cat /sys/bus/ccw/devices/0.0.b100/online
1
# cat /sys/bus/ccw/devices/0.0.b100/readonly
1
# cat /sys/bus/ccw/devices/0.0.b100/uid
IBM.75000000092461.e900.8a
# cat /sys/bus/ccw/devices/0.0.b100/use_diag
1
# cat /sys/bus/ccw/devices/0.0.b100/vendor
IBM
```
Chapter 4. SCSI-over-Fibre Channel device driver

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

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

Features

The zfcp device driver supports the following devices and functions:

- You can use most SAN-attached SCSI device types, for example, SCSI disks, tapes, CD-ROMs, and DVDs.
- SAN access through the following FCP adapters:
  - FICON Express™
  - FICON Express2
  - FICON Express4 (System z9 and later)
- The zfcp device driver supports switched fabric and point-to-point topologies.

What you should know about zfcp

The zfcp device driver is a low-level or host-bus adapter driver that supplements the Linux SCSI stack. Figure 8 illustrates how the device drivers work together.

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 53). 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 55). 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 9 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 383 for more details about the command.

See also “Mapping the representations of a SCSI device in sysfs” on page 56.
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.

SUSE Linux Enterprise Server provides udev to create device nodes for you that allow you to identify the corresponding actual device.

Device nodes based on device names
udev creates device nodes that match the device names used by the kernel. These standard device nodes have the form /dev/name.

The examples in this chapter use standard device nodes as assigned by the SCSI stack. These nodes have the form /dev/sd<x> for entire disks and /dev/sd<x><n> for partitions. In these node names <x> represents one or more letters and <n> is an integer. Refer to Documentation/devices.txt in the Linux source tree for more information on the SCSI device naming scheme.

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

Device nodes based on bus IDs
udev creates device nodes of the form
/dev/disk/by-path/ccw-<device_bus_id>-zfcp-<WWPN>:<LUN>

for whole SCSI device and
/dev/disk/by-path/ccw-<device_bus_id>-zfcp-<WWPN>:<LUN>-part<n>

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

Device nodes based on file system information
udev creates device nodes of the form
/dev/disk/by-uuid/<uuid>

where <uuid> is the UUID for the file system in a partition.

If a file system label has been assigned, udev also creates a node of the form
/dev/disk/by-label/<label>

There are no device nodes for the whole SCSI device that are based on file system information.

Additional device nodes
/dev/disk/by-id contains additional device nodes for the SCSI device and partitions, that are all based on a unique SCSI identifier generated by querying the device.
Example: For a SCSI device that is assigned the device name sda, has two partitions, a device bus-ID 0.0.3c1b (device number 0x3c1b), and a UUID 7eaf9c95-55ac-4e5e-8f18-065b313e63ca for the first and b4a818c8-747c-40a2-bfa2-acaa3ef70ead for the second partition, udev creates the following device nodes:

For the whole SCSI device:
- /dev/sda (standard device node according to the SCSI device naming scheme)
- /dev/disk/by-path/ccw-0.0.3c1b-zfcp-0x500507630300c562:0x401040ea00000000
- /dev/disk/by-id/scsi-36005076303ff5620000000000010ea

For the first partition:
- /dev/sda1 (standard device node according to the SCSI device naming scheme)
- /dev/disk/by-label/boot
- /dev/disk/by-path/ccw-0.0.3c1b-zfcp-0x500507630300c562:0x401040ea00000000-part1
- /dev/disk/by-id/scsi-36005076303ff56200000000000010ea-part1
- /dev/disk/by-uuid/7eaf9c95-55ac-4e5e-8f18-065b313e63ca

For the second partition:
- /dev/sda2 (standard device node according to the SCSI device naming scheme)
- /dev/disk/by-label/SWAP-sda2
- /dev/disk/by-path/ccw-0.0.3c1b-zfcp-0x500507630300c562:0x401040ea00000000-part2
- /dev/disk/by-id/scsi-36005076303ff56200000000000010ea-part2
- /dev/disk/by-uuid/b4a818c8-747c-40a2-bfa2-acaa3ef70ead

For information about multipath devices and multipath partitions, see developerWorks:

Partitioning a SCSI device

You can partition SCSI devices that are attached through an FCP channel in the same way that you can partition SCSI attached devices on other platforms. Use the fdisk command to partition a SCSI disk, not fdasd.

udev creates device nodes for partitions automatically. For the SCSI disk /dev/sda, the partition device nodes are called /dev/sda1, /dev/sda2, /dev/sda3, and so on.

Example
To partition a SCSI disk with a device node /dev/sda issue:

```
# fdisk /dev/sda
```

zfcp HBA API (FC-HBA) support

The zfcp host bus adapter API (HBA API) provides an interface for SAN management clients that run on System z.

As shown in Figure 10 on page 45 the zfcp HBA API support includes a user space library.
The SNIA (Storage Networking Industry Association) library can interface with the zFCP HBA API. The SNIA library is not part of SUSE Linux Enterprise Server. It is available as hbaapi_src_<x.x>.tgz, and can be found at:
http://hbaapi.sourceforge.net

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

The default method in SUSE Linux Enterprise Server is for applications to use the zFCP HBA API library directly.

For information on setting up the HBA API support, see "Installing the zfcp HBA API library" on page 47.

**FCP LUN access control**

**IBM System z10 and later**

FCP LUN access control is not supported.

Access to devices can be restricted by access control software on the FCP channel. For more information on FCP LUN Access Control, visit The IBM Resource Link™ Web site at:
https://www.ibm.com/servers/resourcelink/

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

N_Port ID Virtualization (NPIV) allows a single FCP port to appear as multiple, distinct ports that provide separate port identification. NPIV support can be configured on the SE per CHPID and LPAR for an FCP adapter. The zfcp device driver supports NPIV error messages and adapter attributes. See “Displaying adapter information” on page 49 for the adapter attributes.

For more details, refer to the connectivity page at

N_Port ID Virtualization is available on IBM System z9 and later.

Further information

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

Getting Started with zSeries® Fibre Channel Protocol
Introduces the concepts of Fibre Channel Protocol support, and shows how various SCSI devices can be configured to build an IBM mainframe FCP environment. The information is written for Linux 2.4, but much of it is of a general nature and also applies to Linux 2.6:
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/pdfs/sg246344.pdf

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/systems/z/connectivity/

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

Setting up the zfcp device driver
This section provides information on how you can specify a SCSI boot device.

zfcp module parameters
This section describes how to load and configure the zfcp device driver.
zfcp module parameter syntax

```
modprobe zfcp device=<device_bus_id>,<wwpn>,<fcp_lun>,
dbfsize=4,
dbfsize=<pages>
```

where:

- `<device_bus_id>` specifies the device bus-ID of the FCP channel through which the SCSI device is attached.
- `<wwpn>` specifies the target port through which the SCSI device is accessed.
- `<fcp_lun>` specifies the LUN of the SCSI device.
- `<pages>` specifies the number of pages which should be used for the debug feature.

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

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

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

```
zfcp.dbfsize=6
```

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

---

### Installing the zfcp HBA API library

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

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

You can install the libzfcphbaapi0 RPM using YaST.

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

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

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

- Working with FCP channels
  - Setting an FCP channel online or offline
  - Displaying adapter information
  - Recovering a failed FCP channel
  - Configuring and removing ports
  - Finding out if NPIV is in use
  - Starting and stopping collection of QDIO performance statistics
- 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
  - Recovering a failed SCSI device
  - Removing SCSI devices

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

Setting an FCP channel online or offline

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

Use the `chccwdev` command ("chccwdev - Set a CCW device online” on page 334) 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

  ```
  ```
Displaying adapter information

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

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hardware_version</td>
<td>Hardware version</td>
</tr>
<tr>
<td>card_version</td>
<td>Adapter version</td>
</tr>
<tr>
<td>lic_version</td>
<td>Hardware microcode level</td>
</tr>
<tr>
<td>in_recovery</td>
<td>Shows if adapter is in recovery (0 or 1)</td>
</tr>
<tr>
<td>failed</td>
<td>Shows if adapter is in failed state (0 or 1)</td>
</tr>
<tr>
<td>online</td>
<td>Shows if adapter is online (0 or 1)</td>
</tr>
<tr>
<td>peer_wwnn</td>
<td>WWNN of peer for a point-to-point connection</td>
</tr>
<tr>
<td>peer_wwpn</td>
<td>WWPN of peer for a point-to-point connection</td>
</tr>
<tr>
<td>peer_d_id</td>
<td>Destination ID of the peer for a point-to-point connection</td>
</tr>
<tr>
<td>port_remove</td>
<td>Contains the WWPN of a port to be removed from an FCP channel. See <a href="#">Configuring and removing ports</a></td>
</tr>
<tr>
<td>port_rescan</td>
<td>Triggers a port scan. See <a href="#">Configuring and removing ports</a></td>
</tr>
</tbody>
</table>

For the attributes availability, cmb_enable, and cutype, see [Devices and device attributes](#) on page 9. The status attribute is reserved.

Table 10. Relevant transport class attributes, fc_host attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxframe_size</td>
<td>Maximum frame size of adapter</td>
</tr>
<tr>
<td>node_name</td>
<td>Worldwide node name (WWNN) of adapter</td>
</tr>
<tr>
<td>permanent_port_name</td>
<td>WWPN associated with the physical port of the FCP channel</td>
</tr>
<tr>
<td>port_id</td>
<td>Destination ID of the adapter port.</td>
</tr>
<tr>
<td>port_name</td>
<td>WWPN. If N_Port ID Virtualization is not available, this shows the same value as permanent_port_name.</td>
</tr>
<tr>
<td>port_type</td>
<td>Port type indicating topology of port.</td>
</tr>
<tr>
<td>serial_number</td>
<td>Serial number of adapter.</td>
</tr>
<tr>
<td>speed</td>
<td>Speed of FC link.</td>
</tr>
<tr>
<td>supported_classes</td>
<td>Supported FC service class.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>supported_speeds</td>
<td>Supported speeds.</td>
</tr>
<tr>
<td>tgid_bind_type</td>
<td>Target binding type.</td>
</tr>
</tbody>
</table>

Table 11. Relevant transport class attributes, fc_host statistics

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset_statistics</td>
<td>Writeable attribute to reset statistic counters.</td>
</tr>
<tr>
<td>seconds_since_last_reset</td>
<td>Seconds since last reset of statistic counters.</td>
</tr>
<tr>
<td>tx_frames</td>
<td>Transmitted FC frames.</td>
</tr>
<tr>
<td>tx_words</td>
<td>Transmitted FC words.</td>
</tr>
<tr>
<td>rx_frames</td>
<td>Received FC frames.</td>
</tr>
<tr>
<td>rx_words</td>
<td>Received FC words.</td>
</tr>
<tr>
<td>lip_count</td>
<td>Number of LIP sequences.</td>
</tr>
<tr>
<td>nos_count</td>
<td>Number of NOS sequences.</td>
</tr>
<tr>
<td>error_frames</td>
<td>Number of frames received in error.</td>
</tr>
<tr>
<td>dumped_frames</td>
<td>Number of frames lost due to lack of host resources.</td>
</tr>
<tr>
<td>link_failure_count</td>
<td>Link failure count.</td>
</tr>
<tr>
<td>loss_of_sync_count</td>
<td>Loss of synchronization count.</td>
</tr>
<tr>
<td>loss_of_signal_count</td>
<td>Loss of signal count.</td>
</tr>
<tr>
<td>prim_seq_protocol_err_count</td>
<td>Primitive sequence protocol error count.</td>
</tr>
<tr>
<td>invalid_tx_word_count</td>
<td>Invalid transmission word count.</td>
</tr>
<tr>
<td>invalid_crc_count</td>
<td>Invalid CRC count.</td>
</tr>
<tr>
<td>fcp_input_requests</td>
<td>Number of FCP operations with data input.</td>
</tr>
<tr>
<td>fcp_output_requests</td>
<td>Number of FCP operations with data output.</td>
</tr>
<tr>
<td>fcp_control_requests</td>
<td>Number of FCP operations without data movement.</td>
</tr>
<tr>
<td>fcp_input_megabytes</td>
<td>Megabytes of FCP data input.</td>
</tr>
<tr>
<td>fcp_output_megabytes</td>
<td>Megabytes of FCP data output.</td>
</tr>
</tbody>
</table>

Issue a command of this form to read an attribute:

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

where:
- `<device_bus_id>` is the device bus-ID that corresponds to the FCP channel.
- `<attribute>` is one of the attributes in Table 9 on page 49.

To read attributes of the associated fc_host use:

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

where:
- `<host_name>` is the ID of the host.
- `<attribute>` is one of the attributes in Table 10 on page 49.
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:

```bash
# 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 383) to display all attributes of an adapter:

```bash
# 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 = "0x00000000"
modalias = "ccw:t1731m03dt1732m03"
one = "0"
peer_d_id = "0x000000"
peer_wwnn = "0x0000000000000000"
peer_wwpn = "0x0000000000000000"
status = "0x5400008e"
Class = "fc_host"
    maxframe_size = "2112 bytes"
    node_name = "0x5005076400cd6aad"
    permanent_port_name = "0x50050764016219a0"
    port_name = "0x5005076401c08f98"
    port_id = "0x650f13"
    port_type = "NPort (fabric via point-to-point)"
    serial_number = "IBM02000000006A0"
    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"
sq_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:

```bash
# cat /sys/bus/ccw/drivers/zfcp/0.0.1940/host0/fc_host/host0/port_name
0xc05076ffef805388
# cat /sys/bus/ccw/drivers/zfcp/0.0.1940/host0/fc_host/host0/permanent_port_name
0x50050764016219a0
```
Alternatively you can use `lszfcp` (see "lszfcp - List zfcp devices" on page 383) to display the above attributes:

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

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

**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:

```bash
# 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:

```bash
# 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:

```bash
# 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:

```bash
# 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.

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

Normally there is no need to start a scan for ports, however, if a remote storage port was accidentally deleted from the adapter configuration, or if you are unsure whether all ports are attached, you can use the port_rescan attribute. Issue a command of this form:

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

where:

`<device_bus_id>`

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

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

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 55) or if the port is in use, for example, by error recovery. Note that the next port scan will attach a removed port again if the port is available. If you do not want this, consider zoning.

**Example**

In this example, a port with WWPN 0x500507630303c562 has already been configured for an FCP Channel with device bus-ID 0.0.3d0c. An additional target port with WWPN 0x500507630300c562 is automatically configured by triggering a port scan, and then the port with WWPN 0x500507630303c562 is removed.

```
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/port_rescan
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
0x500507630300c562
# 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 12 on page 54 summarizes the relevant attributes.
Table 12. Attributes with port information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access_denied</td>
<td>Flag that indicates if the port access is restricted by access control software on the FCP channel (see FCP LUN access control on page 45). The value is “1” if access is denied and “0” if access is permitted.</td>
</tr>
<tr>
<td>in_recovery</td>
<td>Shows if port is in recovery (0 or 1)</td>
</tr>
<tr>
<td>failed</td>
<td>Shows if port is in failed state (0 or 1)</td>
</tr>
</tbody>
</table>

Table 13. Transport class attributes with port information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_name</td>
<td>WWNN of the remote port.</td>
</tr>
<tr>
<td>port_name</td>
<td>WWPN of remote port.</td>
</tr>
<tr>
<td>port_id</td>
<td>Destination ID of remote port.</td>
</tr>
<tr>
<td>port_state</td>
<td>State of remote port.</td>
</tr>
<tr>
<td>roles</td>
<td>Role of remote port (usually FCP target).</td>
</tr>
<tr>
<td>scsi_target_id</td>
<td>Linux SCSI ID of remote port.</td>
</tr>
<tr>
<td>supported_classes</td>
<td>Supported classes of service.</td>
</tr>
</tbody>
</table>

Issue a command of this form to read an attribute:

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

where:
- `<device_bus_id>` is the device bus-ID that corresponds to the FCP channel.
- `<wwpn>` is the WWPN of the target port.
- `<attribute>` is one of the attributes in Table 12

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

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

where:
- `<rport_name>` is the name of the remote port.
- `<attribute>` is one of the attributes in Table 13

Examples

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

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/access_denied
0
```

- To display transport class attributes of a target port you can use `lszfcp`:
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 53.

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
```

Chapter 4. SCSI-over-Fibre Channel device driver  55
where:

\(<fcp\_lun>\)

is the LUN of the SCSI device to be configured. The LUN is a 16 digit hexadecimal value padded with zeroes, for example 0x4010403300000000.

\(<device\_bus\_id>\)

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

\(<wwpn>\)

is the WWPN of the target port.

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 55)
- 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 11 on page 57 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.
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 383).

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 14 summarizes the relevant attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access_denied</td>
<td>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 &quot;FCP LUN access control&quot; on page 45).</td>
</tr>
</tbody>
</table>
### Table 14. Attributes with device access information (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access_shared</td>
<td>Flag that indicates if access to the device is shared or exclusive.</td>
</tr>
<tr>
<td></td>
<td>The value is “1” if access is shared and “0” if access is exclusive. (See</td>
</tr>
<tr>
<td></td>
<td>“FCP LUN access control” on page 45).</td>
</tr>
<tr>
<td>access_readonly</td>
<td>Flag that indicates if write access to the device is permitted or if access</td>
</tr>
<tr>
<td></td>
<td>is restricted to read-only.</td>
</tr>
<tr>
<td></td>
<td>The value is “1” if access is restricted read-only and “0” if write access</td>
</tr>
<tr>
<td></td>
<td>is permitted. (See “FCP LUN access control” on page 45).</td>
</tr>
<tr>
<td>in_recovery</td>
<td>Shows if unit is in recovery (0 or 1)</td>
</tr>
<tr>
<td>failed</td>
<td>Shows if unit is in failed state (0 or 1)</td>
</tr>
</tbody>
</table>

### Table 15. SCSI device class attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>device_blocked</td>
<td>Flag that indicates if device is in blocked state (0 or 1).</td>
</tr>
<tr>
<td>iocounterbits</td>
<td>The number of bits used for I/O counters.</td>
</tr>
<tr>
<td>iodone_cnt</td>
<td>The number of completed or rejected scsi commands.</td>
</tr>
<tr>
<td>ioerr_cnt</td>
<td>The number of SCSI commands that completed with an error.</td>
</tr>
<tr>
<td>iorequest_cnt</td>
<td>The number of SCSI commands.</td>
</tr>
<tr>
<td>queue_depth</td>
<td>The depth of the queue for this SCSI device.</td>
</tr>
<tr>
<td>queue_type</td>
<td>The type of queue for the SCSI device. The value can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>• none</td>
</tr>
<tr>
<td></td>
<td>• simple</td>
</tr>
<tr>
<td></td>
<td>• ordered</td>
</tr>
<tr>
<td>model</td>
<td>The model of the SCSI device (received from inquiry data).</td>
</tr>
<tr>
<td>rev</td>
<td>The revision of the SCSI device (received from inquiry data).</td>
</tr>
<tr>
<td>scsi_level</td>
<td>The SCSI revision level (received from inquiry data).</td>
</tr>
<tr>
<td>type</td>
<td>The type of the SCSI device (received from inquiry data).</td>
</tr>
<tr>
<td>vendor</td>
<td>The vendor of the SCSI device (received from inquiry data).</td>
</tr>
<tr>
<td>fcp_lun</td>
<td>The LUN of the SCSI device in 64-bit format.</td>
</tr>
<tr>
<td>hba_id</td>
<td>The bus ID of the SCSI device.</td>
</tr>
<tr>
<td>wwpn</td>
<td>The WWPN of the remote port.</td>
</tr>
<tr>
<td>state</td>
<td>Shows the state of SCSI devices (writeable attribute). The value is one of</td>
</tr>
<tr>
<td></td>
<td>the following:</td>
</tr>
<tr>
<td></td>
<td>• created</td>
</tr>
<tr>
<td></td>
<td>• running</td>
</tr>
<tr>
<td></td>
<td>• cancel</td>
</tr>
<tr>
<td></td>
<td>• quiesce</td>
</tr>
<tr>
<td></td>
<td>• offline</td>
</tr>
<tr>
<td></td>
<td>• blocked</td>
</tr>
<tr>
<td>timeout</td>
<td>Shows the timeout for SCSI commands sent on this device (writeable attribute).</td>
</tr>
<tr>
<td>delete</td>
<td>Triggers deletion of a SCSI device (write-only attribute).</td>
</tr>
<tr>
<td>rescan</td>
<td>Triggers a re-scan of a SCSI device (write-only attribute).</td>
</tr>
</tbody>
</table>

Issue a command of this form to read an attribute:
where:
<device_bus_id> is the device bus-ID that corresponds to the FCP channel.
<wwpn> is the WWPN of the target port.
<fcp_lun> is the FCP LUN of the SCSI device.
<attribute> is one of the attributes in Table 14 on page 57.

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

```
# cat /sys/class/scsi_device/<device_name>/<attribute>
```

where:
<device_name> is the name of the associated SCSI device.
<attribute> is one of the attributes in Table 15 on page 58.

Tip: For SCSI tape devices you can display a summary of this information by using the `lstape` command (see "lstape - List tape devices" on page 378).

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.

```bash
# 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 53).

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

```bash
# 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"
    ierror_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"
```
Starting and stopping collection of QDIO performance statistics

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

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

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

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

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

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

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

To display QDIO performance statistics issue:

```bash
cat /proc/qdio_perf
```

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:

```bash
# 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 55.

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:

```bash
# 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:

```bash
# 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:

```bash
# 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 56](#) for information on how to find this directory. Issue a command of this form:

```bash
# 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:

```bash
# 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 55](#).

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.

```bash
# 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.

```bash
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/online
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/port_rescan
# 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:
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 SUSE Linux Enterprise Server 11 for System z.

Functions provided

The zfcp HBA API (see "zfcp HBA API (FC-HBA) support" on page 44) 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_RegisterLibrary()
- HBA_RegisterLibraryV2()
- HBA_GetNumberOfAdapters()
- HBA_GetAdapterName()
- HBA_OpenAdapter()
- HBA_CloseAdapter()
- HBA_RefreshInformation()
- HBA_RefreshAdapterConfiguration()
- HBA_GetAdapterAttributes()
- HBA_GetAdapterPortAttributes()
- HBA_GetDiscoveredPortAttributes()
- HBA_GetFcpTargetMapping()
- HBA_GetFcpTargetMappingV2()
- HBA_SendScsiInquiry()
- HBA_SendReadCapacity()
- HBA_SendReportLUNs()
- HBA_SendReportLUNsV2()

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

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

Environment variables

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

LIB_ZFCP_HBAAPI_LOG_LEVEL

to specify the log level. If not set or set to zero there is no logging (default). If set to an integer value greater than 1, logging is enabled.
**LIB_ZFCP_HBAAPI_LOG_FILE**

specifies a file for the logging output. If not specified stderr is used.
Chapter 5. Channel-attached tape device driver

The tape device driver supports channel-attached tape devices on SUSE Linux Enterprise Server 11 for System z.

SCSI tape devices attached through a System z FCP adapter are handled by the zfcp device driver (see Chapter 4, “SCSI-over-Fibre Channel device driver,” on page 41).

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 68).
- Message display support (see "tape390_display - display messages on tape devices and load tapes" on page 419).
- Encryption support (see "tape390_crypt - manage tape encryption" on page 415).
- Up to 128 physical tape devices.

What you should know about channel-attached tape devices

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

Tape device modes and logical devices

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

Non-rewinding character device

Provides sequential (traditional) tape access without any caching done in the kernel.

You can use the character device in the same way as any other Linux tape device. You can write to it and read from it using normal Linux facilities such as GNU tar. You can perform control operations (such as rewinding the tape or skipping a file) with the standard tool mt. Most Linux tape software should work with the character device.

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

Rewinding character device

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

Block device

Provides a read-only tape block device.

This device could be used for the installation of software in the same way as tapes are used under other operating systems on the System z.
platforms. (This is similar to the way most Linux software distributions are shipped on CD using the ISO9660 file system.)

It is advisable to use only the ISO9660 file system on System z tapes, because this file system is optimized for CD-ROM devices, which – just like 3480, 3490, or 3590 tape devices – cannot perform fast 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>
<thead>
<tr>
<th>Names</th>
<th>Minor numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-rewinding character devices ntibm&lt;n&gt;</td>
<td>2x&lt;n&gt;</td>
</tr>
<tr>
<td>Rewinding character devices rtibm&lt;n&gt;</td>
<td>2x&lt;n&gt;+1</td>
</tr>
<tr>
<td>Block devices</td>
<td>btibm&lt;n&gt;</td>
</tr>
</tbody>
</table>

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 Istape command (see “Istape - List tape devices” on page 378) to determine the current mapping of index numbers to physical tape devices.

When the tape device driver is loaded, it dynamically allocates a major number to channel-attached character tape devices and a major number to channel-attached block tape devices. The major numbers can but need not be the same. Different major number might be used when the device driver is reloaded, for example when Linux is rebooted.

For online tape devices directories provide information on the major/minor assignments. The directories have the form:
- /sys/class/tape390/ntibm<n>
- /sys/class/tape390/rtibm<n>
- /sys/block/btibm<n>

Each of these directories has a dev attribute. The value of the dev attribute has the form <major>:<minor>, where <major> is the major number for the character or block tape devices and <minor> is the minor number specific to the logical device.
Example

In this example, four physical tape devices are present, with three of them online. The TapeNo column shows the index number and the BusID indicates the associated physical tape device. In the example, no index number has been allocated to the tape device in the first row. This means that the device is offline and, currently, no names and minor numbers are assigned to it.

```
# lstape --ccw-only

<table>
<thead>
<tr>
<th>TapeNo</th>
<th>BusID</th>
<th>CuType/Model</th>
<th>DevType/DevMod</th>
<th>BlkSize</th>
<th>State</th>
<th>Op</th>
<th>MedState</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0.01a1</td>
<td>3490/10</td>
<td>3490/40</td>
<td>auto</td>
<td>UNUSED</td>
<td>---</td>
<td>UNLOADED</td>
</tr>
<tr>
<td>1</td>
<td>0.0.01a0</td>
<td>3480/01</td>
<td>3480/04</td>
<td>auto</td>
<td>UNUSED</td>
<td>---</td>
<td>UNLOADED</td>
</tr>
<tr>
<td>2</td>
<td>0.0.0172</td>
<td>3590/50</td>
<td>3590/11</td>
<td>auto</td>
<td>IN_USE</td>
<td>---</td>
<td>LOADED</td>
</tr>
<tr>
<td>N/A</td>
<td>0.0.01ac</td>
<td>3490/10</td>
<td>3490/40</td>
<td>N/A</td>
<td>OFFLINE</td>
<td>---</td>
<td>N/A</td>
</tr>
</tbody>
</table>
```

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

<table>
<thead>
<tr>
<th>Bus ID</th>
<th>Index (TapeNo)</th>
<th>Device</th>
<th>Device name</th>
<th>Minor number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.01ac</td>
<td>not assigned</td>
<td>non-rewind</td>
<td>ntibm0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rewind</td>
<td>rtibm0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>block</td>
<td>btibm0</td>
<td>0</td>
</tr>
<tr>
<td>0.0.01a1</td>
<td>0</td>
<td>non-rewind</td>
<td>ntibm1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rewind</td>
<td>rtibm1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>block</td>
<td>btibm1</td>
<td>2</td>
</tr>
<tr>
<td>0.0.01a0</td>
<td>1</td>
<td>non-rewind</td>
<td>ntibm2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rewind</td>
<td>rtibm2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>block</td>
<td>btibm2</td>
<td>4</td>
</tr>
</tbody>
</table>

For the online character devices, the major/minor assignments can be read from their respective representations in /sys/class:

```
# cat /sys/class/tape390/ntibm0/dev
254:0
# cat /sys/class/tape390/rtibm0/dev
254:1
# cat /sys/class/tape390/ntibm1/dev
254:2
# cat /sys/class/tape390/rtibm1/dev
254:3
# cat /sys/class/tape390/ntibm2/dev
254:4
# cat /sys/class/tape390/rtibm2/dev
254:5
```

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

Similarly, the major/minor assignments for the online block devices can be read from their respective representations in /sys/block:
The minor numbers are as expected for the respective device names. In the example, the major number used for block devices is also 254.

**Tape device nodes**

User space programs access tape devices by *device nodes*. SUSE Linux Enterprise Server uses udev to create three device nodes for each tape device. The device nodes have the form `/dev/<name>`, where `<name>` is the device name according to "Tape naming scheme" on page 66.

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

<table>
<thead>
<tr>
<th>Tape device nodes</th>
<th>non-rewind device</th>
<th>rewind device</th>
<th>block device</th>
</tr>
</thead>
<tbody>
<tr>
<td>First tape device</td>
<td>/dev/ntibm0</td>
<td>/dev/rtibm0</td>
<td>/dev/btibm0</td>
</tr>
<tr>
<td>Second tape device</td>
<td>/dev/ntibm1</td>
<td>/dev/rtibm1</td>
<td>/dev/btibm1</td>
</tr>
</tbody>
</table>

**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 72 for details).

Setting up the tape device driver

You must load the appropriate tape device driver module before you can work with
tape devices.

Use the modprobe command to ensure that any other required modules are loaded
in the correct order.

<table>
<thead>
<tr>
<th>Tape module syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>modprobe tape_34xx</td>
</tr>
<tr>
<td>tape_3590</td>
</tr>
</tbody>
</table>

See the modprobe man page for details on modprobe.

Working with the tape device driver

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

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

For information on working with the channel measurement facility, see Chapter 36,
"Channel measurement facility," on page 319.

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 419.

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
334) to set a tape online or offline. Alternatively, you can write “1” to the device’s
online attribute to set it online or “0” to set it offline.

When a physical tape device is set online, the device driver assigns an index
number to it. This index number is used in the standard device nodes (see "Tape
device nodes” on page 68 to identify the corresponding logical devices. The index number is in the range 0 to 127. A maximum of 128 physical tape devices can be online concurrently.

If you are using the standard device nodes, you need to find out which index number the tape device driver has assigned to your tape device. This index number, and consequently the associated standard device node, can change after a tape device has been set offline and back online.

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

```
# lstape --ccw-only <device_bus_id>
```

where `<device_bus_id>` is the device bus-ID that corresponds to the physical tape device. The index number is the value in the TapeNo column of the command output.

**Examples**

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

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

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

  To find the index number the tape device driver has assigned, issue:

  ```
  # lstape 0.0.015f --ccw-only
  ```

  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 18. Tape device attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>online</td>
<td>“1” if the device is online or “0” if it is offline (see “Setting a tape device online or offline” on page 69)</td>
</tr>
<tr>
<td>cmb_enable</td>
<td>“1” if channel measurement block is enabled for the physical device or “0” if it is not enabled (see Chapter 36, “Channel measurement facility,” on page 319)</td>
</tr>
<tr>
<td>cutype</td>
<td>Type and model of the control unit</td>
</tr>
<tr>
<td>devtype</td>
<td>Type and model of the physical tape device</td>
</tr>
<tr>
<td>blocksize</td>
<td>Currently used block size in bytes or “0” for auto</td>
</tr>
<tr>
<td>state</td>
<td>State of the physical tape device, either of:</td>
</tr>
<tr>
<td></td>
<td>UNUSED Device is not in use and is currently available to any operating system image in a shared environment</td>
</tr>
<tr>
<td></td>
<td>IN_USE Device is being used as a character device by a process on this Linux image</td>
</tr>
<tr>
<td></td>
<td>BLKUSE Device is being used as a block device by a process on this Linux image</td>
</tr>
<tr>
<td></td>
<td>OFFLINE The device is offline.</td>
</tr>
<tr>
<td></td>
<td>NOT_OP Device is not operational</td>
</tr>
<tr>
<td>operation</td>
<td>The current tape operation, for example:</td>
</tr>
<tr>
<td></td>
<td>--- No operation</td>
</tr>
<tr>
<td></td>
<td>WRI Write operation</td>
</tr>
<tr>
<td></td>
<td>RFO Read operation</td>
</tr>
<tr>
<td></td>
<td>Several other operation codes exist, for example, for rewind and seek.</td>
</tr>
<tr>
<td>medium_state</td>
<td>The current state of the tape cartridge:</td>
</tr>
<tr>
<td></td>
<td>1 Cartridge is loaded into the tape device</td>
</tr>
<tr>
<td></td>
<td>2 No cartridge is loaded</td>
</tr>
<tr>
<td></td>
<td>0 The tape device driver does not have information about the current cartridge state</td>
</tr>
</tbody>
</table>

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 18.

Tip: You can display a summary of this information by using the istape command (see “istape - List tape devices” on page 378).

Example

The following sequence of commands reads the attributes for a physical tape device with a device bus-ID 0.0.015f:
Issuing an `lstape` command for the same device yields:

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

Enabling compression

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

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

```
# mt -f <node> compression
```

or

```
# mt -f <node> compression 1
```

where `<node>` is the device node for a character device, for example, `/dev/ntibm0`.

To switch compression off, issue:

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

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

**Example**

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

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

**Loading and unloading tapes**

You can unload tapes by issuing a command of this form:

```
# mt -f <node> unload
```
where `<node>` is one of the character device nodes.

Whether or not you can load tapes from your Linux instance depends on the stacker mode of your tape hardware. There are three possible modes:

**manual**

Tapes must always be loaded manually by an operator. You can use the `tape390_display` command (see "tape390_display - display messages on tape devices and load tapes" on page 419) 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 419). You cannot use the `mt` command to load a tape.

**Example**

To expel a tape from a tape device that can be accessed through a device node `/dev/ntibm0`, issue:

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

Assuming that the stacker mode of the tape device is “system” and that a tape is present in the stacker, you can load a new tape by issuing:

```
# tape390_display -l "NEW TAPE" /dev/ntibm0
```

“NEW TAPE” is a message that is displayed on the tape devices display unit until the tape device receives the next tape movement command.

**Scenario: Using a tape block device**

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

**Assumptions:** The following assumptions are made:

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

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

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

2. Set the tape online:

```
# chccwdev -e 0.0.015f
```
3. If you are using standard device nodes, find out which index number the tape device driver has assigned to it. You can skip this step if you are using udev-created device nodes that distinguish devices by device bus-ID rather than the index number.

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

The index number is shown in the TapeNo column of the command output, “1” in the example. The standard device nodes are therefore /dev/ntibml, /dev/rtibml, and /dev/btibml.

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/ntibml rewind
# mt -f /dev/ntibml 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/ntibml 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/ntibml bs=2048
```

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

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

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

```
# mount -t iso9660 -o ro,block=2048 /dev/btibml /mnt
```
Chapter 6. XPRAM device driver

With the XPRAM block device driver SUSE Linux Enterprise Server 11 for System z can access expanded storage. Thus XPRAM can be used as a basis for fast swap devices and/or fast file systems. Expanded storage range can be swapped in or out of the main storage in 4 KB blocks. All XPRAM devices do always provide a block size of 4096 bytes.

XPRAM features

The XPRAM device driver provides the following features:
- Automatic detection of expanded storage.
  If expanded storage is not available, XPRAM fails gracefully with a log message reporting the absence of expanded storage.
- The expanded storage can be divided into up to 32 partitions.

What you should know about XPRAM

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

XPRAM partitions and device nodes

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

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

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

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

<table>
<thead>
<tr>
<th>Minor</th>
<th>Name</th>
<th>To access</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>slram0</td>
<td>the first partition or the entire expanded storage if there are no partitions</td>
</tr>
<tr>
<td>1</td>
<td>slram1</td>
<td>the second partition</td>
</tr>
<tr>
<td>2</td>
<td>slram2</td>
<td>the third partition</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>&lt;n&gt;</td>
<td>slram&lt;n&gt;</td>
<td>the (&lt;n&gt;+1)th partition</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>31</td>
<td>slram31</td>
<td>the 32nd partition</td>
</tr>
</tbody>
</table>

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

Issuing an IPL command to reboot SUSE Linux Enterprise Server 11 for 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.

Setting up the XPRAM device driver

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

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

XPRAM module parameter syntax

```
insmod xpram
modprobe devs=<number_of_partitions>,
sizes=<partition_size>
```

where:

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

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

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

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

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

Examples

- The following specification allocates the extended storage into four partitions. Partition 1 has 2 GB (2097152 KB), partition 4 has 4 GB (4194304 KB), and partitions 2 and 3 use equal parts of the remaining storage. If the total amount of extended storage was 16 GB, then partitions 3 and 4 would each have approximately 5 GB.
  
  # modprobe xpram devs=4 sizes=2097152,0,0,4194304

- The following specification allocates the extended storage into three partitions. The partition 2 has 512 KB and the partitions 1 and 3 use equal parts of the remaining extended storage.
  
  # modprobe xpram devs=3 sizes=,512

- The following specification allocates the extended storage into two partitions of equal size.
  
  # modprobe xpram devs=2
Part 3. Networking

This part describes the network device drivers for SUSE Linux Enterprise Server 11 for System z.

**Newest version:** You can find the newest version of this book at

**Restrictions:** For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 release notes at

---

**Chapter 7. qeth device driver for OSA-Express (QDIO) and HiperSockets**

Device driver functions ........................................... 83
What you should know about the qeth device driver ............. 85
Setting up the qeth device driver ................................ 91
Working with the qeth device driver .............................. 92
Working with the qeth device driver in layer 3 mode .......... 105
Scenario: VIPA – minimize outage due to adapter failure .... 112
Scenario: Virtual LAN (VLAN) support ......................... 117
HiperSockets Network Concentrator ............................... 120
Setting up for DHCP with IPv4 ................................. 126
Setting up a Linux guest as a z/VM guest LAN sniffer ....... 127

**Chapter 8. OSA-Express SNMP subagent support** ............ 129
What you need to know about osasnmkp ......................... 129
Setting up osasnmkp ............................................. 131
Working with the osasnmkp subagent ............................ 134

**Chapter 9. LAN channel station device driver** ............... 139
Features .................................................................. 139
What you should know about LCS .................................. 139
Setting up the LCS device driver .................................. 140
Working with the LCS device driver .............................. 140

**Chapter 10. CTCM device driver** ................................. 143
Features .................................................................. 143
What you should know about CTCM ............................ 143
Setting up the CTCM device driver ............................... 145
Working with the CTCM device driver ......................... 145
Scenarios .................................................................. 149

**Chapter 11. NETIUCV device driver** ............................ 153
Features .................................................................. 153
What you should know about IUCV ............................ 153
Setting up the NETIUCV device driver ....................... 154
Working with the NETIUCV device driver .................... 155
Scenario: Setting up an IUCV connection to a TCP/IP service machine .... 158

**Chapter 12. CLAW device driver** ................................. 161
Features .................................................................. 161
What you should know about the CLAW device driver .... 161
Setting up the CLAW device driver .............................. 162
An example network setup that uses some available network setup types is shown in Figure 12.

In the example there are three Linux instances; two of them run as z/VM guests under z/VM in one LPAR and a third Linux instance runs in another LPAR. Within z/VM, Linux instances can be connected directly by IUCV, virtual-CTC, or through a guest LAN. Within and between LPARs, you can connect Linux instances through HiperSockets. Connections outside of a System z complex are possible by OSA-Express cards running either in non-QDIO mode (called LCS here) or in QDIO mode.
Chapter 7. qeth device driver for OSA-Express (QDIO) and HiperSockets

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

**Real connections using OSA-Express**
A System z mainframe offers OSA-Express adapters, which are real LAN-adapter hardware, see Figure 13. These adapters provide connections to the outside world, but can also connect virtual systems (between LPARs or between z/VM-guests) within the mainframe. The qeth driver supports these adapters if they are defined to run in queued direct I/O (QDIO) mode (defined as OSD or OSN in the hardware configuration). OSD-devices are the standard System z LAN-adapters, while OSN-devices serve as NCP-adapters.

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

**HiperSockets**
A System z mainframe offers internal connections called HiperSockets. These simulate QDIO network adapters and provide high-speed TCP/IP communication for operating system instances within and across LPARs.

**Virtual connections under z/VM**
z/VM offers virtualized LAN-adapters that enable connections between z/VM-guests and the outside world. It allows definitions of simulated network interface cards (NICs) attached to certain z/VM-guests. The NICs can be connected to a simulated LAN segment called guest LAN for z/VM.
internal communication between z/VM-guests, or they can be connected to a virtual switch called VSWITCH for external LAN connectivity.

**Guest LAN**

Guest LANs represent a simulated LAN segment that can be connected to simulated network interface cards. There are three types of guest LANs:

- Simulated OSA-Express mode
- Simulated HiperSockets mode
- Simulated Ethernet mode

Each guest LAN is isolated from other guest LANs on the same system (unless some member of one LAN group acts as a router to other groups).

**Virtual switch**

A virtual switch (VSWITCH) is a special-purpose guest LAN that provides external LAN connectivity through an additional OSA-Express device served by z/VM without the need for a routing virtual machine, see Figure 14.

![Figure 14. Virtual switch](image)

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

From a Linux point of view there is no difference between guest LAN- and VSWITCH-devices; thus Linux talks about guest LAN-devices independently of their z/VM-attachment to a guest LAN or VSWITCH.
The qeth network device driver supports System z OSA-Express3, OSA-Express2, and OSA-Express features in QDIO mode, HiperSockets, z/VM guest LAN, and z/VM VSWITCH as follows:

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

<table>
<thead>
<tr>
<th>Function</th>
<th>System z10</th>
<th>System z9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hipersockets</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OSA-Express3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>10 Gigabit Ethernet</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>1000Base-T Ethernet</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>OSA-Express2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>10 Gigabit Ethernet</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1000Base-T Ethernet</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OSA-Express</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fast Ethernet</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>1000Base-T Ethernet</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

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

Device driver functions

The qeth device driver supports functions listed in Table 21 and Table 22 on page 84.

Table 21. Real connections

<table>
<thead>
<tr>
<th>Function</th>
<th>OSA Layer 2</th>
<th>OSA Layer 3</th>
<th>HiperSockets Layer 2 Ethernet</th>
<th>HiperSockets Layer 3 Ethernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4/multicast/broadcast</td>
<td>✓</td>
<td>✓</td>
<td>✓/✓/✓</td>
<td>✓/✓/✓</td>
</tr>
<tr>
<td>IPv6/multicast/broadcast</td>
<td>✓</td>
<td>✓</td>
<td>✓/✓/✓</td>
<td>✓/✓/✓</td>
</tr>
<tr>
<td>Non-IP traffic</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VLAN IPv4/IPv6/non IP</td>
<td>sw/sw/sw</td>
<td>hw/sw/sw</td>
<td>sw/sw/sw</td>
<td>hw/-/-</td>
</tr>
<tr>
<td>Linux ARP</td>
<td>✓</td>
<td>- (hw ARP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux neighbor solicitation</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unique MAC address</td>
<td>✓ (random)</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change MAC address</td>
<td>✓</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Promiscuous mode</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC headers send/receive</td>
<td>✓</td>
<td>faked/faked</td>
<td>✓/✓/✓</td>
<td>faked/faked</td>
</tr>
<tr>
<td>ethtool support</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bonding</td>
<td>✓</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 21. Real connections (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>OSA Layer 2</th>
<th>OSA Layer 3</th>
<th>HiperSockets Layer 2</th>
<th>HiperSockets Layer 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority queueing</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Offload features</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP segmentation offload</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enhanced Device Driver</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Packing (EDDP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rx HW checksum</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>OSA/QETH specific features</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Special device driver</td>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setup for VIPA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Special device driver</td>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setup for proxy ARP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Special device driver</td>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setup for IP takeover</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Special device driver</td>
<td></td>
<td>required/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setup for routing</td>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPv4/IPv6</td>
<td>-/-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive buffer count</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct connectivity to z/OS</td>
<td>✔ by HW</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNMP support</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiport support</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Legend:**
- Function not supported or not required
hw Function performed by hardware
sw Function performed by software
✔ Function supported

### Table 22. Guest LAN connections

<table>
<thead>
<tr>
<th>Function</th>
<th>OSA Layer 2</th>
<th>OSA Layer 3</th>
<th>HiperSockets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic device or protocol features</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPv4/multicast/broadcast</td>
<td>✔/✔/✔/✔</td>
<td>✔/✔/✔/✔</td>
<td>✔/✔/✔/✔</td>
</tr>
<tr>
<td>IPv6/multicast/broadcast</td>
<td>✔/✔/✔/✔</td>
<td>✔/✔/✔/✔</td>
<td>-/-/ -/-</td>
</tr>
<tr>
<td>Non-IP traffic</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VLAN IPv4/IPv6/non IP</td>
<td>sw/sw/sw</td>
<td>hw/sw/-</td>
<td>hw/-/-</td>
</tr>
<tr>
<td>Linux ARP</td>
<td>✔</td>
<td></td>
<td>- (hw ARP)</td>
</tr>
<tr>
<td>Linux neighbor solicitation</td>
<td>✔</td>
<td>✔</td>
<td>-</td>
</tr>
<tr>
<td>Unique MAC address</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change MAC address</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Promiscuous mode</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC headers send/receive</td>
<td>✔/✔</td>
<td>faked/faked</td>
<td>faked/faked</td>
</tr>
<tr>
<td>ethtool support</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 22. Guest LAN connections (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>OSA Layer 2</th>
<th>OSA Layer 3</th>
<th>HiperSockets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bonding</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Priority queueing</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

#### Offload features

- **TSO**: -
- **EDDP**: ✓ -
- **rx HW checksum**: - - -

#### OSA/QETH specific features

<table>
<thead>
<tr>
<th>Special device driver setup for</th>
<th>OSA Layer 2</th>
<th>OSA Layer 3</th>
<th>HiperSockets</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIPA</td>
<td>-</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>proxy ARP</td>
<td>-</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>IP takeover</td>
<td>-</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>routing IPv4/IPv6</td>
<td>-/-</td>
<td>required/required</td>
<td>required/required</td>
</tr>
<tr>
<td>Receive buffer count</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Direct connectivity to z/OS</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SNMP support</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Multiport support</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Legend:**
- Function not supported or not required
- **hw**: Function performed by hardware
- **sw**: Function performed by software
- ✓ Function supported

---

### What you should know about the qeth device driver

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

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

**The layer 2 discipline (qeth_l2)**

The layer 2 discipline supports:
- OSA and OSA guest LAN devices
- OSA for NCP devices
- HiperSockets devices

The layer 2 discipline is the default setup for OSA. On HiperSockets the default continues to be layer 3. OSA guest LANs are layer 2 by default, while HiperSockets guest LANs are layer 3 by default.

**The layer 3 discipline (qeth_l3)**

The layer 3 discipline supports:
- OSA and OSA guest LAN devices running in layer 3 mode (with faked link layer headers)
HiperSockets and HiperSockets guest LAN devices running in layer 3 mode (with faked link layer headers)

This discipline supports those devices that are not capable of running in layer 2 mode. Not all Linux networking features are supported and others need special setup or configuration. See Table 27 on page 93. Some performance-critical applications might benefit from being layer 3.

Note that Layer 2 cannot communicate with Layer 3.

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 15. I/O subchannel interface

The three device bus-IDs that correspond to the subchannel triplet are grouped as one qeth group device. The following rules apply for the device bus-IDs:

read    no specific rules.
write   must be the device bus-ID of the read subchannel plus one.
data    can be any free device bus-ID on the same CHPID.

You can configure different triplets of device bus-IDs on the same CHPID differently. For example, if you have two triplets on the same CHPID they can have different attribute values for priority queueing.

Overview of the steps for setting up a qeth group device

Before you start: Find out how the hardware is configured and which qeth device bus-IDs are on which CHPID, for example by looking at the IOCDS. Identify the device bus-IDs that you want to group into a qeth group device. The three device bus-IDs must be on the same CHPID.

You need to perform several steps before user-space applications on your Linux instance can use a qeth group device:

1. Create the qeth group device.
   After booting Linux, each qeth device bus-ID is represented by a subdirectory in /sys/bus/ccw/devices/. These subdirectories are then named with the bus IDs of the devices. For example, a qeth device with bus-IDs 0.0.fc00, 0.0.fc01, and 0.0.fc02 is represented as /sys/bus/ccw/drivers/qeth/0.0.fc00
2. Configure the device.
3. Set the device online.
4. Activate the device and assign an IP address to it.

These tasks and the configuration options are described in detail in “Working with
the qeth device driver” on page 92.

**qeth interface names and device directories**

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

- **eth<n>**
  - for Ethernet features.
- **hsi<n>**
  - for HiperSockets devices.
- **osn<n>**
  - for ESCON/CDLC bridge (OSA NCP).

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

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

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

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

The qeth device driver shares the name space for Ethernet interfaces with the LCS
device driver. Each driver uses the name with the lowest free identifier **<n>**,regardless of which device driver occupies the other names. For example, if the first
qeth Ethernet feature is set online and there is already one LCS Ethernet feature
online, the LCS feature is named “eth0” and the qeth feature is named “eth1”. See
also “LCS interface names” on page 139.

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 100 and “Finding
out the bus ID of a qeth interface” on page 101 provide information on how to map
device bus-IDs and interface names.

**Support for IP Version 6 (IPv6)**

IPv6 is supported on:
- Ethernet interfaces of the OSA-Express adapter running in QDIO mode.
- HiperSockets layer 2 and layer 3 interfaces
- z/VM guest LAN interfaces running in QDIO mode.

There are noticeable differences between the IP stacks for versions 4 and 6. Some
concepts in IPv6 are different from IPv4, such as neighbor discovery, broadcast,
and IPSec. IPv6 uses a 16-byte address field, while the addresses under IPv4 are 4
bytes in length.
Stateless autoconfiguration generates unique IP addresses for all Linux instances, even if they share an OSA-Express adapter with other operating systems.

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

**MAC headers in layer 2 mode**

In LAN environments, data packets find their destination through Media Access Control (MAC) addresses in their MAC header (see Figure 16).

![Figure 16. Standard IPv4 processing](image)

MAC address handling as shown in Figure 16 applies to non-mainframe environments and a mainframe environment with an OSA-Express adapter where the layer2 option is enabled.

The layer2 option keeps the MAC addresses on incoming packets. Incoming and outgoing packets are complete with a MAC header at all stages between the Linux network stack and the LAN as shown in Figure 16. This layer2-based forwarding requires unique MAC addresses for all concerned Linux instances.

In layer 2 mode, the Linux TCP/IP stack has full control over the MAC headers and the neighbor lookup. The Linux TCP/IP stack does not configure IPv4 or IPv6 addresses into the hardware, but requires a unique MAC address for the card and the functionality to configure additional Ethernet multicast addresses on the card.

For HiperSockets connections, a MAC address is generated.

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

For Linux instances that are directly attached to an OSA-Express adapter in QDIO mode, you should assign the MAC addresses yourself. You can add a line `LLADDR='<MAC address>'` to the configuration file `/etc/sysconfig/network/ifcfg-<interface>` Alternatively, you can change the MAC address by issuing the command: `ifconfig <interface> hw ether <MAC address>`
Note: Be sure not to assign the MAC address of the OSA-Express adapter to your Linux instance.

MAC headers in layer 3 mode
Since a qeth layer 3 mode device driver is an Ethernet offload engine for IPv4 and a partial Ethernet offload engine for IPv6 there are some special things to understand about the layer 3 mode.

To support IPv6 and protocols other than IPv4 the device driver registers a layer 3 card as an Ethernet device to the Linux TCP/IP stack.

In layer 3 mode, the OSA-Express adapter in QDIO mode removes the MAC header with the MAC address from incoming IPv4 packets and uses the registered IP addresses to forward a packet to the recipient TCP/IP stack. Thus the OSA-Express adapter is able to deliver IPv4 packets to the correct Linux images. Apart from broadcast packets, a Linux image can only get packets for IP addresses it has configured in the stack and registered with the OSA-Express adapter.

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

This can be a problem for applications that expect MAC headers. For examples of how such problems can be resolved see Setting up for DHCP with IPv4 on page 126.

Outgoing frames
The qeth device driver registers the layer 3 card as an Ethernet device. Therefore, the Linux TCP/IP stack will provide complete Ethernet frames to the device driver. If the hardware does not require the Ethernet frame (for example, for IPv4) the driver removes the Ethernet header prior to sending the frame to the hardware. If necessary information like the Ethernet target address is not available (because of the offload functionality) the value is filled with the hardcoded address FAKELL.

Table 23. Ethernet addresses of outgoing frames

<table>
<thead>
<tr>
<th>Destination address</th>
<th>Source address</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4</td>
<td>FAKELL</td>
</tr>
<tr>
<td>IPv6</td>
<td>Real destination address</td>
</tr>
<tr>
<td>Other packets</td>
<td>Real destination address</td>
</tr>
</tbody>
</table>

Incoming frames
The device driver provides Ethernet headers for all incoming frames. If necessary information like the Ethernet source address is not available (because of the offload functionality) the value is filled with the hardcoded address FAKELL.

Table 24. Ethernet addresses of incoming frames

<table>
<thead>
<tr>
<th>Destination address</th>
<th>Source address</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4</td>
<td>Real device address</td>
</tr>
<tr>
<td>IPv6</td>
<td>Real device address</td>
</tr>
<tr>
<td>Other packets</td>
<td>Real device address</td>
</tr>
</tbody>
</table>

Real source address
Note that if a source or destination address is a multicast or broadcast address the
device driver can provide the corresponding (real) Ethernet multicast or broadcast
address even when the packet was delivered or sent through the offload engine.
Always providing the link layer headers enables packet socket applications like
tcptrace to work properly on a qeth layer 3 device without any changes in the
application itself (the patch for libpcap is no longer required).

While the faked headers are syntactically correct, the addresses are not authentic,
and hence applications requiring authentic addresses will not work. Some examples
are given in Table 25.

Table 25. Applications that react differently to faked headers

<table>
<thead>
<tr>
<th>Application</th>
<th>Support</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcpdump</td>
<td>Yes</td>
<td>Displays only frames, fake Ethernet information is displayed.</td>
</tr>
<tr>
<td>iptables</td>
<td>Partially</td>
<td>As long as the rule does not deal with Ethernet information of an IPv4 frame.</td>
</tr>
<tr>
<td>dhcp</td>
<td>Yes</td>
<td>Is non-IPv4 traffic.</td>
</tr>
</tbody>
</table>

**IP addresses**
The network stack of each operating system that shares an OSA-Express adapter
in QDIO mode registers all its IP addresses with the adapter. Whenever IP
addresses are deleted from or added to a network stack, the device drivers
download the resulting IP address list changes to the OSA-Express adapter.

For the registered IP addresses, the OSA-Express adapter off-loads various
functions, in particular also:

- Handling MAC addresses and MAC headers
- ARP processing

**ARP**: The OSA-Express adapter in QDIO mode responds to Address Resolution
Protocol (ARP) requests for all registered IPv4 addresses.

ARP is a TCP/IP protocol that translates 32-bit IPv4 addresses into the
corresponding hardware addresses. For example, for an Ethernet device, the
hardware addresses are 48-bit Ethernet Media Access Control (MAC) addresses.
The mapping of IPv4 addresses to the corresponding hardware addresses is
defined in the ARP cache. When it needs to send a packet, a host consults the
ARP cache of its network adapter to find the MAC address of the target host.

If there is an entry for the destination IPv4 address, the corresponding MAC
address is copied into the MAC header and the packet is added to the appropriate
interface’s output queue. If the entry is not found, the ARP functions retain the IPv4
packet, and broadcast an ARP request asking the destination host for its MAC
address. When a reply is received, the packet is sent to its destination.

**Notes**:
1. On an OSA-Express adapter in QDIO mode, do not set the NO_ARP flag on the
   Linux Ethernet device. The device driver disables the ARP resolution for IPv4.
   Since the hardware requires no neighbor lookup for IPv4, but neighbor
   solicitation for IPv6, the NO_ARP flag is not allowed on the Linux Ethernet
device.
2. On HiperSockets, which is a full Ethernet offload engine for IPv4 and IPv6 and supports no other traffic, the device driver sets the NO_ARP flag on the Linux Ethernet interface. Do not remove this flag from the interface.

**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](http://www.linuxvm.org/Info/HOWTOs/guestlan.html)
- *HiperSockets Implementation Guide*, SG24-6816
- *z/VM CP Commands and Utilities Reference*, SC24-6081

For information on proxy ARP, visit: [http://www.sjdjweis.com/linux/proxyarp/](http://www.sjdjweis.com/linux/proxyarp/)

For information on layer2, refer to:
- *OSA-Express Customer's Guide and Reference*, SA22-7935
- *OSA-Express Implementation Guide*, SG25-5848
- *Networking Overview for Linux on zSeries*, REDP-3901
- *z/VM Connectivity*, SC24-6080

---

**Setting up the qeth device driver**

No module parameters exist for the qeth device driver. qeth devices are set up using sysfs.

**Loading the qeth device driver modules**

You must load the qeth device driver before you can work with qeth devices. Use the `modprobe` command to load the qeth device driver to automatically load all required additional modules in the correct order:

```bash
!modprobe qeth
 qeth_l2
 qeth_l3
```

where:

- **qeth** is the core module that contains common functions used for both layer 2 and layer 3 disciplines.
- **qeth_l2** is the module that contains layer 2 discipline-specific code.
- **qeth_l3** is the module that contains layer 3 discipline-specific code.

When a qeth device is configured for a particular discipline the driver tries to automatically load the corresponding discipline module.
Switching the discipline of a qeth device

To switch the discipline of a device the device must be offline. If the new discipline is accepted by the device driver the old network interface will be deleted. When the new discipline is set online the first time the new network interface is created.

Removing the modules

Removing a module is not possible if there are cross dependencies between the discipline modules and the core module. To release the dependencies from the core module to the discipline module all devices of this discipline must be ungrouped. Now the discipline module can be removed. If all discipline modules are removed the core module can be removed.

Working with the qeth device driver

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

Most of these tasks involve writing to and reading from attributes of qeth group devices in sysfs. This is useful on a running system where you want to make dynamic changes. If you want to make the changes persistent across IPLs, use the configuration dialog in YaST. YaST, in turn, creates a udev configuration file called /etc/udev/rules.d/xx-qeth-0.0.xxxx.rules. Additionally, cross-platform network configuration parameters are defined in /etc/sysconfig/network/ifcfg-<if_name>

Table 26 and Table 27 on page 93 serve as both a task overview and a summary of the attributes and the possible values you can write to them. Underlined values are defaults.

Not all attributes are applicable to each device. Some attributes apply only to HiperSockets or only to OSA-Express CHPIDs in QDIO mode, other attributes are applicable to IPv4 interfaces only. Refer to the respective task descriptions to see the applicability of each attribute.

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

- if_name
- card_type
- buffer_count
- recover

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Corresponding attributes</th>
<th>Possible attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Creating a qeth group device&quot; on page 94</td>
<td>group</td>
<td>n/a, see &quot;Devices and device attributes&quot; on page 9</td>
</tr>
<tr>
<td>&quot;Setting the layer2 attribute&quot; on page 95</td>
<td>layer2</td>
<td>0 or 1</td>
</tr>
<tr>
<td></td>
<td>portname</td>
<td>any valid port name</td>
</tr>
<tr>
<td>&quot;Providing Large Send - TCP segmentation offload&quot; on page 96</td>
<td>large_send</td>
<td>no EDDP, EDP, TSO</td>
</tr>
</tbody>
</table>
### Table 26. qeth tasks and attributes common to layer2 and layer3 (continued).

<table>
<thead>
<tr>
<th>Task</th>
<th>Corresponding attributes</th>
<th>Possible attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Using priority queueing” on page 97</td>
<td>priority_queueing</td>
<td>prio_queueing, prio_queueing_tos, no_prio_queueing, no_prio_queueing:0, no_prio_queueing:1, no_prio_queueing:2, no_prio_queueing:3</td>
</tr>
<tr>
<td>“Specifying the number of inbound buffers” on page 98</td>
<td>buffer_count</td>
<td>integer in the range 8 to 128, the default is 16</td>
</tr>
<tr>
<td>“Specifying the relative port number” on page 99</td>
<td>portno</td>
<td>integer, either 0 or 1, the default is 0</td>
</tr>
<tr>
<td>“Finding out the type of your network adapter” on page 99</td>
<td>card_type</td>
<td>n/a, read-only</td>
</tr>
<tr>
<td>“Setting a device online or offline” on page 100</td>
<td>online</td>
<td>0 or 1</td>
</tr>
<tr>
<td>“Finding out the interface name of a qeth group device” on page 100</td>
<td>if_name</td>
<td>n/a, read-only</td>
</tr>
<tr>
<td>“Finding out the bus ID of a qeth interface” on page 101</td>
<td>none</td>
<td>n/a</td>
</tr>
<tr>
<td>“Activating an interface” on page 101</td>
<td>none</td>
<td>n/a</td>
</tr>
<tr>
<td>“Deactivating an interface” on page 103</td>
<td>recover</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table 27. qeth tasks and attributes in layer 3 mode.

<table>
<thead>
<tr>
<th>Task</th>
<th>Corresponding attributes</th>
<th>Possible attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Setting up a Linux router” on page 105</td>
<td>route4, route6</td>
<td>primary_router, secondary_router, primary_connector, secondary_connector, multicast_router, no_router</td>
</tr>
<tr>
<td>“Setting the checksumming method” on page 107</td>
<td>checksumming</td>
<td>hw_checksumming, sw_checksumming, no_checksumming</td>
</tr>
<tr>
<td>“Taking broadcast capability” on page 108</td>
<td>fake_broadcast</td>
<td>0 or 1</td>
</tr>
<tr>
<td>“Starting and stopping collection of QETH performance statistics” on page 109</td>
<td>performance_stats</td>
<td>0 or 1</td>
</tr>
<tr>
<td>“Taking over IP addresses” on page 108</td>
<td>ipa_takeover/enable</td>
<td>0 or 1 or toggle</td>
</tr>
<tr>
<td></td>
<td>ipa_takeover/add4</td>
<td>IPv4 or IPv6 IP address and mask bits</td>
</tr>
<tr>
<td></td>
<td>ipa_takeover/add6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ipa_takeover/del4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ipa_takeover/del6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ipa_takeover/invert4</td>
<td>0 or 1 or toggle</td>
</tr>
<tr>
<td></td>
<td>ipa_takeover/invert6</td>
<td></td>
</tr>
<tr>
<td>“Configuring a device for proxy ARP” on page 111</td>
<td>rxip/add4, rxip/add6, rxip/del4, rxip/del6</td>
<td>IPv4 or IPv6 IP address</td>
</tr>
</tbody>
</table>
Table 27. qeth tasks and attributes in layer 3 mode (continued).

<table>
<thead>
<tr>
<th>Task</th>
<th>Corresponding attributes</th>
<th>Possible attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Configuring a device for virtual IP address (VIPA)&quot; on page 112</td>
<td>vipa/add4</td>
<td>IPv4 or IPv6 IP address</td>
</tr>
<tr>
<td></td>
<td>vipa/add6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vipa/del4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vipa/del6</td>
<td></td>
</tr>
</tbody>
</table>

¹ not valid for HiperSockets

Tip:

- Instead of using the attributes for IPA, proxy ARP and VIPA directly, use the `qethconf` command. In YaST, you can use "IPA Takeover".

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.
- Using SUSE Linux Enterprise Server, you set qeth attributes in YaST. YaST, in turn, creates a udev configuration file called `/etc/udev/rules.d/xx-geth-0.0.xxxx.rules`. Additionally, cross-platform network configuration parameters are defined in `/etc/sysconfig/network/ifcfg-<if_name>`.

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:
Result: The qeth device driver uses the device bus-ID of the read subchannel to create a directory for a group device:
\[ /sys/bus/ccwgroup/drivers/qeth/\langle \text{read\_device\_bus\_id} \rangle \]

This directory contains a number of attributes that determine the settings of the qeth group device. The following sections describe how to use these attributes to configure a qeth group device.

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

Mainframe configuration:

```
# echo \langle \text{read\_device\_bus\_id} \rangle,\langle \text{write\_device\_bus\_id} \rangle,\langle \text{data\_device\_bus\_id} \rangle > /sys/bus/ccwgroup/drivers/qeth/group
```

This command results in the creation of the following directories in sysfs:
- `/sys/bus/ccwgroup/drivers/qeth/\langle \text{read\_device\_bus\_id} \rangle`
- `/sys/bus/ccwgroup/devices/\langle \text{read\_device\_bus\_id} \rangle`
- `/sys/devices/qeth/\langle \text{read\_device\_bus\_id} \rangle`

Both the command and the resulting directories would be the same for a HiperSockets CHPID.

Setting the layer2 attribute
The layer2 attribute for OSA devices defaults to 1 (layer 2 mode).

You can make use of the layer2-mode for almost all device types, however, note the following about layer 2-to-layer 3 conversion:

**real OSA-Express**

Hardware is able to convert layer 2-to-layer 3 traffic and vice versa and thus there are no restrictions.

**HiperSockets**

There is no support for layer 2-to-layer 3-conversion and thus there is no
communication possible between HiperSockets layer 2 interfaces and
HiperSockets layer 3 interfaces. For example, the HiperSockets layer 2 LAN
must be separate from the HiperSockets layer 3 LAN.

z/VM guest LAN
Linux has to configure the same mode as the underlying z/VM virtual LAN
definition. The z/VM definition "Ethernet mode" is available for VSWITCHes
and for guest LANs of type QDIO.

Before you start: If you are using the layer2 option within a QDIO based guest
LAN environment, you cannot define a VLAN with ID “1”, because ID “1” is reserved
for z/VM use.

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

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

where `<integer>` is

- 0 to turn the layer2 attribute off; this results in the layer 3 discipline.
- 1 to turn the layer2 attribute on; this results in the layer 2 discipline.

If you configured the discipline successfully, additional configuration attributes are
displayed (for example route4 for the layer 3 discipline) and can be configured. If an
OSA device is not configured for a discipline but is set online, the device driver
assumes it is a layer 2 device and tries to load the layer 2 discipline.

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

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

### Providing Large Send - TCP segmentation offload

Large Send enables you to offload the TCP segmentation operation from the Linux
network stack to the OSA-Express2 or OSA-Express3 features. Large Send can
lead to enhanced performance and latency for interfaces with predominately large
outgoing packets. Large Send availability depends on the layer2 attribute, see
Table 28 and [Table 29 on page 97](#).

**Table 28. Large Send for layer 2 mode (layer2=1)**

<table>
<thead>
<tr>
<th></th>
<th>Real OSA</th>
<th>Guest LAN OSA</th>
<th>Real HiperSockets</th>
<th>Guest LAN HiperSockets</th>
<th>OSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSO</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No¹</td>
</tr>
<tr>
<td>EDDP</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No¹</td>
</tr>
</tbody>
</table>

¹If neither TSO nor EDDP is supported the large_send sysfs attribute is not
displayed.
To set Large Send, issue a command of the form:

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

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

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

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

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

  Due to hardware limitations the QETH driver might be forced to temporarily switch to EDDP-mode.

- **EDDP**
  The qeth device driver provides the Large Send (software TSO). 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:
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 30] shows how the qeth device driver maps service types to the available queues:

<table>
<thead>
<tr>
<th>Service type</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low latency</td>
<td>0</td>
</tr>
<tr>
<td>High throughput</td>
<td>1</td>
</tr>
<tr>
<td>High reliability</td>
<td>2</td>
</tr>
<tr>
<td>Not important</td>
<td>3</td>
</tr>
</tbody>
</table>

**no_prio_queueing**

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

**no_prio_queueing:0**

causes the qeth device driver to use queue 0 for all packets.

**no_prio_queueing:1**

causes the qeth device driver to use queue 1 for all packets.

**no_prio_queueing:2**

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

**no_prio_queueing:3**

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

**Example**

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

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

**Specifying the number of inbound buffers**

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

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

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

The Linux memory usage for inbound data buffers for the devices is: (number of buffers) × (buffer size).
The buffer size is equivalent to the frame size which is:

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

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

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

**Example**

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

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

### Specifying the relative port number

**Before you start:**

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

  In all other cases only a single port is available.

- The device must be offline while you specify the relative port number.

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

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

Where `<integer>` is either 0 or 1.

**Example**

In this example, port 1 is assigned to the qeth group device.

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

### Finding out the type of your network adapter

You can find out the type of the network adapter through which your device is connected. To find out the type read the device’s card_type attribute. Issue a command of the form:

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

The card_type attribute gives information about both the type of network adaptor and also about the type of network link (if applicable) available at the card’s ports. See Table 31 on page 100 for details.
### Table 31. Possible values of card_type and what they mean

<table>
<thead>
<tr>
<th>Value of card_type</th>
<th>Adapter type</th>
<th>Link type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSD_10GIG</td>
<td>OSA card in OSD mode</td>
<td>10 Gigabit Ethernet</td>
</tr>
<tr>
<td>OSD_1000</td>
<td>OSA card in OSD mode</td>
<td>Gigabit Ethernet</td>
</tr>
<tr>
<td>OSD_100</td>
<td>OSA card in OSD mode</td>
<td>Fast Ethernet</td>
</tr>
<tr>
<td>OSD_GbE_LANE</td>
<td>Gigabit Ethernet, LAN Emulation</td>
<td>Fast Ethernet, LAN Emulation</td>
</tr>
<tr>
<td>OSD_FE_LANE</td>
<td>Gigabit Ethernet, LAN Emulation</td>
<td>Fast Ethernet, LAN Emulation</td>
</tr>
<tr>
<td>OSD_Express</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>OSN</td>
<td>OSA for NCP</td>
<td>ESCON/CDLC bridge or N/A</td>
</tr>
<tr>
<td>HiperSockets</td>
<td>HiperSockets, CHPID type IQD</td>
<td>N/A</td>
</tr>
<tr>
<td>GuestLAN QDIO</td>
<td>Guest LAN based on OSA</td>
<td>N/A</td>
</tr>
<tr>
<td>GuestLAN Hiper</td>
<td>Guest LAN based on HiperSockets</td>
<td>N/A</td>
</tr>
<tr>
<td>Unknown</td>
<td></td>
<td>Other</td>
</tr>
</tbody>
</table>

### 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:

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

**Tip:** you can also use the `lsqeth -p` command (see “lsqeth - List qeth based network devices” on page 374) to obtain a mapping for all qeth interfaces and devices. The `/proc/qeth` file is no longer maintained.

**Example**

```bash
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a100/if_name
eh0
```

**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 -p` command (see “lsqeth - List qeth based network devices” on page 374) 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:

```bash
# 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 100).
- 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 under layer 3**

The Linux network stack design does not allow feedback about IP address changes. If `ifconfig` or an equivalent command fails to set an IP address on an OSA-Express network CHPID, a query with `ifconfig` shows the address as being set on the interface although the address is not actually set on the CHPID.

There are usually failure messages about not being able to set the IP address or duplicate IP addresses in the kernel messages. You can display these messages with `dmesg`. In SUSE Linux Enterprise Server you can also find the messages in `/var/log/messages`.

There may be circumstances that prevent an IP address from being set, most commonly if another system in the network has set that IP address already.

If you are not sure whether an IP address was set properly or experience a networking problem, check the messages or logs to see if an error was
encountered when setting the address. This also applies in the context of HiperSockets and to both IPv4 and IPv6 addresses. It also applies to whether an IP address has been set for IP takeover, for VIPA, or for proxy ARP.

**Duplicate IP addresses**

The OSA-Express adapter in QDIO mode recognizes duplicate IP addresses on the same OSA-Express adapter or in the network using ARP and prevents duplicates.

Several setups require duplicate addresses:

- To perform IP takeover you need to be able to set the IP address to be taken over. This address exists prior to the takeover. See “Taking over IP addresses” on page 108 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 111 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 112 for details.

You can use the `qethconf` command (see “qethconf - Configure qeth devices” on page 400) to maintain a list of IP addresses that your device can take over, a list of IP addresses for which your device can handle ARP, and a list of IP addresses that can be used as virtual IP addresses, regardless of any duplicates on the same OSA-Express adapter or in the LAN.

**Deactivating an interface**

You can deactivate an interface with `ifconfig` or an equivalent command or by setting the network device offline. While setting a device offline involves actions on the attached device, deactivating only stops the interface logically within Linux.

To deactivate an interface with `ifconfig`, issue a command of the form:

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

**Example**

To deactivate eth0 issue:

```
# ifconfig eth0 down
```

**Recovering a device**

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

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

**Example**

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

**Starting and stopping collection of QETH performance statistics**

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

Chapter 7. qeth device driver for OSA-Express (QDIO) and HiperSockets 103
This attribute is initially set to 0, that is QETH performance data is not collected. To start collection for a specific QETH device, write 1 to the attribute, for example:

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

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

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

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

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

```
    # ethtool -S eth0
    NIC statistics:
    rx skbs: 86
    rx buffers: 85
    tx skbs: 86
    tx buffers: 86
    tx skbs no packing: 86
    tx buffers no packing: 86
    tx skbs packing: 0
    tx buffers packing: 0
    tx sg skbs: 0
    tx sg frags: 0
    rx sg skbs: 0
    rx sg frags: 0
    rx sg page allocs: 0
    tx large kbytes: 0
    tx large count: 0
    tx pk state ch n->p: 0
    tx pk state ch p->n: 0
    tx pk watermark high: 5
    queue 0 buffer usage: 0
    queue 1 buffer usage: 0
    queue 2 buffer usage: 0
    queue 3 buffer usage: 0
    rx handler time: 856
    rx handler count: 84
    rx do_QDIO time: 16
    rx do_QDIO count: 11
    tx handler time: 330
    tx handler count: 87
    tx time: 1236
    tx count: 86
    tx do_QDIO time: 997
    tx do_QDIO count: 86
    
    # ethtool -i eth0
    driver: qeth_13
    version: 1.0
    firmware-version: 087a
    bus-info: 0.0.f5f0/0.0.f5f1/0.0.f5f2
```

To control QDIO performance statistics as well, see "Starting and stopping collection of QDIO performance statistics" on page 60.

Tip: use the `ethtool` command to display performance statistics for qeth devices instead of the `/proc/qeth_perf` file which is no longer maintained.
Working with the qeth device driver in layer 3 mode

Setting up a Linux router

Before you start:

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

By default, your Linux instance is not a router. Depending on your IP version, IPv4 or IPv6 you can use the route4 or route6 attribute of your qeth device to define it as a router. You can set the route4 or route6 attribute dynamically, while the qeth device is online.

The same values are possible for route4 and route6 but depend on the type of CHPID:

Table 32. Summary of router setup values

<table>
<thead>
<tr>
<th>Router specification</th>
<th>OSA-Express CHPID in QDIO mode</th>
<th>HiperSockets CHPID</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary_router</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>secondary_router</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>primary_connector</td>
<td>–</td>
<td>✓</td>
</tr>
<tr>
<td>secondary_connector</td>
<td>–</td>
<td>✓</td>
</tr>
<tr>
<td>multicast_router</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>no_router</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

The values are explained in detail below.

An OSA-Express CHPID in QDIO mode honors the following values:

**primary_router**
- to make your Linux instance the principal connection between two networks.

**secondary_router**
- to make your Linux instance a backup connection between two networks.

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

**primary_connector**
- to make your Linux instance the principal connection between a HiperSockets network and an external network (see “HiperSockets Network Concentrator” on page 120).

**secondary_connector**
- to make your Linux instance a backup connection between a HiperSockets network and an external network (see “HiperSockets Network Concentrator” on page 120).

Both types of CHPIDs honor:

**multicast_router**
- causes the qeth driver to receive all multicast packets of the CHPID. For a unicast function for HiperSockets see “HiperSockets Network Concentrator” on page 120.
**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
```

**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. In SUSE Linux Enterprise Server you can set IP forwarding permanently in `/etc/sysctl.conf` or dynamically with the `sysctl` command.

**Mainframe configuration:**

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 0.0.0.0400,0.0.0.0401,0.0.0.0402 > /sys/bus/ccwgroup/drivers/qeth/group
# echo 0.0.0.0200,0.0.0.0201,0.0.0.0202 > /sys/bus/ccwgroup/drivers/qeth/group
```

To make Linux P a primary router for IPv4:
```
# echo primary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0.0400/route4
# echo primary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0.0200/route4
```

**Linux S configuration:**
To create the qeth group devices:

```
# echo 0.0.0.0404,0.0.0.0405,0.0.0.0406 > /sys/bus/ccwgroup/drivers/qeth/group
# echo 0.0.0.0204,0.0.0.0205,0.0.0.0206 > /sys/bus/ccwgroup/drivers/qeth/group
```

To make Linux S a secondary router for IPv4:

```
# echo secondary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0.0404/route4
# echo secondary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0.0204/route4
```

In this example, qeth device 0.01510 is defined as a primary router for IPv6:

```
/sys/bus/ccwgroup/drivers/qeth # cd 0.0.1510
# echo 1 > online
# echo primary_router > route6
# cat route6
primary router
```

See “HiperSockets Network Concentrator” on page 120 for further examples.

### Setting the checksumming method

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

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

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

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

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

- **hw_checksumming**
  - performs the checksumming in hardware if the CHPID is an OSA-Express CHPID in QDIO mode and your OSA adapter hardware supports checksumming.
  - If you set "hw_checksumming" for an adapter that does not support it or for a HiperSockets CHPID, the TCP/IP stack performs the checksumming instead of the adapter.

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

- **no_checksumming**
  - suppresses checksumming.

  **Attention:** Suppressing checksumming might jeopardize data integrity.

**Examples**

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

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

### Faking broadcast capability

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

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

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

```
# ifconfig eth0
eth0 Link encap:Ethernet  HWaddr 00:09:6b:1a:9a:b7
  inet6 addr: fe80::9:6b00:af1a:9ab7/64 Scope:Link
  UP BROADCAST RUNNING MULTICAST  MTU:1492  Metric:1
  RX packets:107792 errors:0 dropped:0 overruns:0 frame:0
  TX packets:12176 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:1000
  RX bytes:29753474 (28.3 MiB) TX bytes:1979603 (1.8 MiB)
```

Some processes, for example, the `gated` routing daemon, require the devices’ broadcast capable flag to be set in the Linux network stack. To set this flag for devices that do not support broadcast set the fake_broadcast attribute of the qeth group device to “1”. To reset the flag set it to “0”.

Issue a command of the form:

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

**Example**

In this example, a device 0.0.a100 is instructed to pretend that it has broadcast capability.

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

### Taking over IP addresses

This section describes how to configure for IP takeover if the layer2 option (see “MAC headers in layer 2 mode” on page 88) is not enabled. If you have enabled the layer2 option, you can configure for IP takeover as you would in a distributed server environment.

Taking over an IP address overrides any previous allocation of this address to another LPAR. If another LPAR on the same CHPID has already registered for that IP address, this association is removed.
An OSA-Express CHPID in QDIO mode can take over IP addresses from any System z operating system. IP takeover for HiperSockets CHPIIDs 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 addresses to be activated or deactivated. See “qethconf - Configure qeth devices” on page 400 for more details on the `qethconf` command.

**Example:** In this example, there is only one range of IP addresses (192.168.10.0 to 192.168.10.255) that can be taken over by device hsi0.
The following command adds a range of IP addresses that can be taken over by
device eth0.

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

qethconf: Added 192.168.10.0/24 to /sys/class/net/hsi0/device/ipa_takeover/add4.
qethconf: Use "qethconf ipa list" to check for the result

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

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

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

```
# qethconf ipa del 192.168.11.0/24 eth0
```

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 109).
- 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 109).

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 under
layer 3" on page 102 when using IP takeover.
Configuring a device for proxy ARP

This section describes how to configure for proxy ARP if the layer2 option (see “MAC headers in layer 2 mode” on page 86) 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 400).

Be aware of the information in “Confirming that an IP address has been set under layer 3” on page 102 when working with proxy ARP.

Example

Figure 17 shows an environment where proxy ARP is used.

![Diagram of network environment]

Figure 17. Example of proxy ARP usage

G1, G2, and G3 are Linux guests (connected, for example, through a guest LAN to a Linux router R), reached from GW (or the outside world) via R. R is the ARP proxy for G1, G2, and G3. That is, R agrees to take care of packets destined for G1, G2, and G3. The advantage of using proxy ARP is that GW does not need to know that G1, G2, and G3 are behind a router.

To receive packets for 1.2.3.4, so that it can forward them to G1 1.2.3.4, R would add 1.2.3.4 to its list of IP addresses for proxy ARP for the interface that connects it to the OSA adapter.

```
# qethconf parp add 1.2.3.4 eth0
qethconf: Added 1.2.3.4 to /sys/class/net/eth0/device/rxip/add4.
qethconf: Use "qethconf parp list" to check for the result
```

After issuing similar commands for the IP addresses 1.2.3.5 and 1.2.3.6 the proxy ARP configuration of R would be:

```
# qethconf parp list
parp add 1.2.3.4 eth0
parp add 1.2.3.5 eth0
parp add 1.2.3.6 eth0
```
Configuring a device for virtual IP address (VIPA)

This section describes how to configure for VIPA if the layer2 option (see “MAC headers in layer 2 mode” on page 88) is not enabled. If you have enabled the layer2 option, you can configure for VIPA as you would in a distributed server environment.

Before you start:

- This section does not apply to HiperSockets.

System z use VIPAs to protect against certain types of hardware connection failure. You can assign VIPAs that are independent from particular adapter. VIPAs can be built under Linux using dummy devices (for example, “dummy0” or “dummy1”).

The qeth device driver maintains a list of VIPAs that the OSA-Express adapter accepts for each qeth group device. Use the qethconf utility to add or remove VIPAs (see “qethconf - Configure qeth devices” on page 400).

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

Be aware of “Confirming that an IP address has been set under layer 3” on page 102 when working with VIPAs.

Scenario: VIPA – minimize outage due to adapter failure

This chapter describes how to use

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

Using VIPA you can assign IP addresses that are not associated with a particular adapter. This minimizes outage caused by adapter failure. Standard VIPA is usually sufficient for applications, such as Web 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. See the information in “Confirming that an IP address has been set under layer 3” on page 102 concerning possible failure when setting IP addresses for OSA-Express features in QDIO mode (qeth driver).
2. The configuration file layout for Source VIPA has changed since the 1.x versions. In the 2.0.0 version a policy is included. For details see the README and the man pages provided with the package.

Standard VIPA

Purpose

VIPA is a facility for assigning an IP address to a system, instead of to individual adapters. It is supported by the Linux kernel. The addresses can be in IPv4 or IPv6 format.

Usage

These are the main steps you must follow to set up VIPA in Linux:

1. Create a dummy device with a virtual IP address.
2. Ensure that your service (for example, the Apache Web server) listens to the virtual IP address assigned above.

3. Set up routes to the virtual IP address, on clients or gateways. To do so, you can use either:
   - Static routing (shown in the example of Figure 18).
   - 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 18 shows the network adapter configuration used in the example.

```
1. Define the real interfaces
[server]# ifconfig eth0 10.1.0.2 netmask 255.255.0.0
[server]# ifconfig eth1 10.2.0.2 netmask 255.255.0.0
```

2. Ensure that the dummy module has been loaded. If necessary, load it by issuing:
```
[server]# modprobe dummy
```

3. Create a dummy interface with a virtual IP address 9.164.100.100 and a netmask 255.255.255.0:
```
[server]# ifconfig dummy0 9.164.100.100 netmask 255.255.255.0
```
4. Enable the network devices for this VIPA so that it accepts packets for this IP address.

```
[server]# qethconf vipa add 9.164.100.100 eth0
qethconf: Added 9.164.100.100 to /sys/class/net/eth0/device/vipa/add4.
qethconf: Use "qethconf vipa list" to check for the result
[server]# qethconf vipa add 9.164.100.100 eth1
qethconf: Added 9.164.100.100 to /sys/class/net/eth1/device/vipa/add4.
qethconf: Use "qethconf vipa list" to check for the result
```

For IPv6, the address is specified in IPv6 format:

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

5. Ensure that the addresses have been set:

```
[server]# qethconf vipa list
vipa add 9.164.100.100 eth0
vipa add 9.164.100.100 eth1
```

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

7. Set up a route to the virtual IP address (static routing), so that VIPA can be reached via the gateway with address 10.1.0.2.

```
[router]# route add -host 9.164.100.100 gw 10.1.0.2
```

Now we assume an adapter outage occurs. We must therefore:

1. Delete the previously-created route.

```
[router]# route delete -host 9.164.100.100
```

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

```
[router]# route add -host 9.164.100.100 gw 10.2.0.2
```

### Source VIPA

**Purpose**

Source VIPA is particularly suitable for high-performance environments. It selects one source address out of a range of source addresses when it replaces the source address of a socket. The reason for using several source addresses lies in the inability of some operating system kernels to do load balancing among several connections with the same source and destination address over several interfaces.

To achieve load balancing, a *policy* has to be selected in the *policy* section of the configuration file of Source VIPA (/etc/src_vipa.conf). This policy section also allows to specify several source addresses used for one destination. Source VIPA then applies the source address selection according to the rules of the policy selected in the configuration file.
This Source VIPA solution does not affect kernel stability. Source VIPA is controlled by a configuration file containing flexible rules for when to use Source VIPA based on destination IP address ranges.

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

**Usage**

**Installation:** An RPM is available for Source VIPA. The RPM is called src_vipa-<version>.s390x.rpm. Install the RPM as usual.

**Configuration:** With Source VIPA version 2.0.0 the configuration file has changed: the policy section was added. The default configuration file is /etc/src_vipa.conf.

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

```
# comment
```

D1.D2.D3.D4/MASK 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.

```
```

causes bind calls with .INADDR_ANY as a local address to be intercepted if the port the socket is bound to is between P1 and P2 (inclusive). In this case, .INADDR_ANY will be replaced by one of the source addresses specified (S, T, [...] which can be 0.0.0.0.

All .INADDR_ANY statements will be read and evaluated in order of appearance. This means that multiple .INADDR_ANY statements can be used to have bind calls intercepted for every port outside a certain range. This is useful, for example, for rlogin, which uses the bind command to bind to a local port but with .INADDR_ANY as a source address to use automatic source address selection. See "Policies" below for available load distribution policies.

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

**Policies:** With Source VIPA Extensions you provide a range of dummy source addresses for replacing the source addresses of a socket. The policy selected determines which method is used for selecting the source addresses from the range of dummy addresses.

- **onevipa**
  - Only the first address of all source addresses specified is used as source address.

- **random**
  - The source address used is selected randomly from all the specified source addresses.
Ilr (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

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 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.

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.
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 20. Conventional routed 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 21. Switched VLAN network
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 Ethernet LAN through eth1:

```text
(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
```

```text
(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
```

```text
(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
```

```text
(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
```

```text
(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
```

---

**HiperSockets Network Concentrator**

This section describes how to configure a HiperSockets Network Concentrator on a QETH device in layer 3 mode.

**Before you start:** This section applies to IPv4 only. The HiperSockets Network Concentrator connector settings are available in layer 3 mode only.

The HiperSockets Network Concentrator connects systems to an external LAN within one IP subnet using HiperSockets. HiperSockets Network Concentrator connected systems appear as if they were directly connected to the LAN. This
helps to reduce the complexity of network topologies resulting from server consolidation. HiperSockets Network Concentrator allows to migrate systems from the LAN into a System z Server environment, or systems connected by a different HiperSockets Network Concentrator into a System z Server environment, without changing the network setup. Thus, HiperSockets Network Concentrator helps to simplify network configuration and administration.

Design

A connector Linux system forwards traffic between the external OSA interface and one or more internal HiperSockets interfaces. This is done via IPv4 forwarding for unicast traffic and via a particular bridging code (xciec_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 111) on the OSA interfaces. The script also establishes routes for all internal systems to enable IP forwarding between the interfaces.

All unicast packets that cannot be delivered in the HiperSockets network are handed over to the connector by HiperSockets. The connector also receives all multicast packets to bridge them.

Setup

The setup principles for configuring the HiperSockets Network Concentrator are as follows:

leaf nodes

The leaf nodes do not require a special setup. To attach them to the HiperSockets network, their setup should be as if they were directly attached to the LAN. They do not have to be Linux systems.

connector systems

In the following, HiperSockets Network Concentrator IP refers to the subnet of the LAN that is extended into the HiperSockets net.

- If you want to support forwarding of all packet types, define the OSA interface for traffic into the LAN as a multicast router (see “Setting up a Linux router” on page 105) and set operating_mode=full in /etc/sysconfig/hsnc.

- 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 manually with the command

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

  Alternatively, you can enable IP forwarding in the /etc/sysctl.conf configuration file to activate IP forwarding for the connector partition automatically after booting. For HiperSockets Network Concentrator on SUSE Linux Enterprise Server an additional config file exists: /etc/sysconfig/hsnc.

- 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, issue:
  ```
  service hsnc start
  ```

  In `/etc/sysconfig/hsnc` 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, issue
  ```
  service hsnc stop
  ```

**Availability setups**

If a connector system fails during operation, it can simply be restarted. If all the startup commands are executed automatically, it will instantaneously be operational again after booting. Two common availability setups are mentioned here:

**One connector partition and one monitoring system**

As soon as the monitoring system cannot reach the connector for a specific timeout (for example, 5 seconds), it restarts the connector. The connector itself monitors the monitoring system. If it detects (with a longer timeout than the monitoring system, for example, 15 seconds) a monitor system failure, it restarts the monitoring system.

**Two connector systems monitoring each other**

In this setup, there is an active and a passive system. As soon as the passive system detects a failure of the active connector, it takes over operation. In order to do this it needs to reset the other system to release all OSA resources for the multicast_router operation. The failed system can then be restarted manually or automatically, depending on the configuration. The passive backup HiperSockets interface can either switch into primary_connector mode during the failover, or it can be setup as secondary_connector. A secondary_connector takes over the connecting functionality, as soon as there is no active primary_connector. This setup has a faster failover time than the first one.

**Hints**

- The MTU of the OSA and HiperSockets link should be of the same size. Otherwise multicast packets not fitting in the link’s MTU are discarded as there is no IP fragmentation for multicast bridging. Warnings are printed to `/var/log/messages` or a corresponding syslog destination.
- The script `ip_watcher.pl` prints error messages to the standard error descriptor of the process.
- `xcec-bridge` logs messages and errors to syslog. On SUSE Linux Enterprise Server 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:

Examples

Figure 23 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.

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: See SUSE Linux Enterprise Server 11 Administration Guide for information about 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 hs10
```

To start the HiperSockets network concentrator issue:

```
# service hsnc start
```

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 24 shows the example of Figure 23 on page 123 with an additional mainframe. On the second mainframe a Linux instance D acts as a HiperSockets network concentrator.

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:

```bash
# 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 therefore only forwards unicast packets.

To enable IP forwarding issue:

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

Tip: See SUSE Linux Enterprise Server 11 Administration Guide for information about how to use configuration files to automatically enable IP forwarding when booting.

To start the HiperSockets network concentrator issue:

```bash
# service hsnc start
```
Setup for H:
No special setup required. The HiperSockets interface has IP address 10.20.30.55, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

Setting up for DHCP with IPv4

For connections through an OSA-Express adapter in QDIO mode, the OSA-Express adapter offloads ARP, MAC header, and MAC address handling (see “MAC headers in layer 3 mode” on page 89). Because a HiperSockets connection does not go out on a physical network, there are no ARP, MAC headers, and MAC addresses for packets in a HiperSockets LAN. The resulting problems for DHCP are the same in both cases and the fixes for connections through the OSA-Express adapter also apply to HiperSockets.

Dynamic Host Configuration Protocol (DHCP) is a TCP/IP protocol that allows clients to obtain IP network configuration information (including an IP address) from a central DHCP server. The DHCP server controls whether the address it provides to a client is allocated permanently or is leased temporarily. DHCP specifications are described by RFC 2131 “Dynamic Host Configuration Protocol” and RFC 2132 “DHCP options and BOOTP Vendor Extensions”, which are available on the Internet at: [http://www.ietf.org/](http://www.ietf.org/)

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

For information on setting up DHCP for a SUSE Linux Enterprise Server 11 for System z instance in a z/VM guest LAN environment, refer to Redpaper Linux on IBM eServer™ zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN, REDP-3596 at: [ibm.com/redbooks/](http://ibm.com/redbooks/)

As an example of DHCP client and server that you can use, this book discusses the dhcpcd client and server dhcp.

Required options for using dhcpcd with layer3

The following options are required:
- 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.

You need no special options for using dhcp.
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 AS/400 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 module has been loaded.

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 IBM books" on page xi).
Chapter 8. OSA-Express SNMP subagent support

The OSA-Express Simple Network Management Protocol (SNMP) subagent (osasnmpd) supports management information bases (MIBs) for the following OSA-Express features in QDIO mode only:

- OSA-Express
  - Fast Ethernet
  - 1000Base-T Ethernet
  - Gigabit Ethernet
- OSA-Express2
  - Gigabit Ethernet
  - 10 Gigabit Ethernet
  - 1000Base-T Ethernet (System z9 and System z10)
- OSA-Express3
  - Gigabit Ethernet
  - 10 Gigabit Ethernet
  - 1000Base-T Ethernet

This subagent capability through the OSA-Express features is also called Direct SNMP to distinguish it from another method of accessing OSA SNMP data through OSA/SF, a package for monitoring and managing OSA features that does not run on Linux.

To use the osasnmpd subagent you need:

- An OSA-Express feature running in QDIO mode with the latest textual MIB file for the appropriate LIC level (recommended)
- The qeth device driver for OSA-Express (QDIO) and HiperSockets
- The osasnmpd subagent from s390-tools
- 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 131), but you do not need to update the subagent. Place the updated MIB file in a directory that is searched by the master agent.

**Example:** This example shows the processes running after the snmpd master agent and the osasnmud subagent have been started. When you start osasnmud, a daemon called osasnmud-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       COMM
root  687 snmpd   0 11:57 pts/1    00:00:00 snmp
root  729 osasnmud-2.6   0 13:22 pts/1    00:00:00 osasnmud-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

You can set up osasnmpd using YaST; this section describes how to set up osasnmpd using the command line. In YaST, go to "/etc/sysconfig Editor", then select Network -> SNMP -> OSA Express SNMP agent -> OSASNMPD_PARAMETERS

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.

   - 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.
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:
     ```
     ==&gt;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 133).

**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](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 the SUSE Linux Enterprise Server documentation to find out where you need to place the snmpd.conf file. Some of the possible locations are:
   • /etc
   • /etc/snmp

2. Open snmpd.conf with your preferred text editor. There might be a sample in
usr/share/doc/packages/net-snmp/EXAMPLE.conf

3. Find the security name section and include a line of this form to map a community name to a security name:
   com2sec <security-name> <source> <community-name>

   where:
   <security-name>
   is given access rights through further specifications within snmpd.conf.
   <source>
   is the IP-address or DNS-name of the accessing system, typically a Network Management Station.
   <community-name>
   is the community string used for basic SNMP password protection.

   Example:
   # sec.name source community
   com2sec osasec default osacom
   com2sec pubsec localhost public

4. Find the group section. Use the security name to define a group with different versions of the master agent for which you want to grant access rights. Include a line of this form for each master agent version:
   group <group-name> <security-model> <security-name>

   where:
   <group-name>
   is a group name of your choice.
   <security-model>
   is the security model of the SNMP version.
   <security-name>
   is the same as in step 3

   Example:
   # groupName securityModel securityName
   group osagroup v1 osasec
   group osagroup v2c osasec
   group osagroup usm osasec
   group osasnmppd v2c pubsec

   Group "osasnmppd" with community "public" is required by osasnmppd 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 osasnmqd 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 132.

<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 osasnmqd "" 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 131.

   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 132.

   Example:
   
   defVersion 2c
   defCommunity osacom

   These default specifications simplify issuing master agent commands.

13. Save and close snmp.conf.

---

**Working with the osasnmprd subagent**

This section describes the following tasks:

- Starting the osasnmprd subagent
- Checking the log file
- Issuing queries
- Stopping osasnmprd

**Starting the osasnmprd subagent**

In SUSE Linux Enterprise Server you start the osasnmprd subagent using the command:

```
# service snmp start
```

or the start script:

```
# rcsnmpd start
```

The osasnmprd subagent, in turn, starts a daemon called osasnmprd-2.6.

Define osasnmprd parameters in YaST. You can specify the following parameters:

- `-I` 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/osasnmprd.log`.

- `-L` or `--stderrlog`
  - print messages and warnings to stdout or 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.

YaST creates a configuration file called /etc/sysconfig/osasnmpd, for example:

```bash
## Path: Network/SNMP/OSA Express SNMP agent
## Description: OSA Express SNMP agent parameters
## Type: string
## Default: ""
## ServiceRestart: snmpd
#
# OSA Express SNMP agent command-line parameters
#
# Enter the parameters you want to be passed on to the OSA Express SNMP
# agent.
#
# Example: OSASNMPD_PARAMETERS="-l /var/log/my_private_logfile"
# OSASNMPD_PARAMETERS="-A"
```

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.

```bash
# 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 374) 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
chipid      : 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_11     : 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 131.

  If you have provided defaults for the SNMP version and the community (see step 12 on page 134), 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:
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

To stop both snmpd and the osasnmpd subagent, issue the command:

```
# service snmpd stop
```

or using the script:

```
# rcsnmpd stop
```
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-Express
  - Fast Ethernet
  - 1000Base-T Ethernet
- OSA-Express2
  - 1000Base-T Ethernet (System z9 and System z10)
- OSA-Express3
  - 1000Base-T Ethernet (System z10 only)

Features

The LCS device driver supports the following devices and functions:

- Automatically detects an Ethernet connection
- Internet Protocol, version 4 (IPv4) only

What you should know about LCS

This section provides information about LCS group devices and interfaces.

LCS group devices

The LCS device driver requires two I/O subchannels for each LCS interface, a read subchannel and a write subchannel. The corresponding bus-IDs must be configured for control unit type 3088.

The device bus-IDs that correspond to the subchannel pair are grouped as one LCS group device. The following rules apply for the device bus-IDs:

- read must be even.
- write must be the device bus-ID of the read subchannel plus one.

LCS interface names

When an LCS group device is set online, the LCS device driver automatically assigns an interface name to it. The naming scheme uses the base name:

- `eth<n>` for Ethernet features

where `<n>` is an integer that uniquely identifies the device. When the first device for a base name is set online it is assigned 0, the second is assigned 1, the third 2, and so on. Each base name is counted separately.
For example, the interface name of the first Ethernet feature that is set online is “eth0”, the second “eth1”, and so on.

The LCS device driver shares the name space for Ethernet interfaces with the qeth device driver. Each driver uses the name with the lowest free identifier <n>, regardless of which device driver occupies the other names. For example, if at the time the first LCS Ethernet feature is set online, there is already one qeth Ethernet feature online, the qeth feature is named “eth0” and the LCS feature is named “eth1”. See also “qeth interface names and device directories” on page 87.

### Setting up the LCS device driver

There are no module parameters for the LCS device driver.

You need to load the lcs module before you can work with LCS devices. Load the lcs module with the modprobe command to ensure that any other required modules are loaded in the correct order:

```
# modprobe lcs
```

### Working with the LCS device driver

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

- Creating an LCS group device
- Specifying a timeout for LCS LAN commands
- Setting a device online or offline
- Activating and deactivating an interface
- Recovering a device

### 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
You can specify a timeout for the interval that the LCS device driver waits for a reply after issuing a LAN command to the LAN adapter. For older hardware the replies may take a longer time. The default is 5 s.

To set a timeout issue a command of this form:

```
# echo <timeout> > /sys/bus/ccwgroup/drivers/lcs/<device_bus_id>/lancmd_timeout
```

where `<timeout>` is the timeout interval in seconds in the range from 1 to 60.

**Example**
In this example, the timeout for a device 0.0.d000 is set to 10 s.

```
# echo 10 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/lancmd_timeout
```

Setting a device online or offline
To set an LCS group device online, set the online device group attribute to “1”. To set a LCS group device offline, set the online device group attribute to “0”. Issue a command of this form:

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

Setting a device online associates it with an interface name. Setting the device offline preserves the interface name.

Read /var/log/messages or issue dmesg to find out which interface name has been assigned. You will need to know the interface name to activate the network interface.

For each online interface, there is a symbolic link of the form /sys/class/net/<interface_name>/device in sysfs. You can confirm that you have found the correct interface name by reading the link.

**Example**
To set an LCS device with bus ID 0.0.d000 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/online
# dmesg
...
lcs: LCS device eth0 without IPv6 support
lcs: LCS device eth0 with Multicast support
...
```

The interface name that has been assigned to the LCS group device in the example is eth0. To confirm that this is the correct name for our group device issue:
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" on page 141).

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

#### Examples

- This example activates an Ethernet interface:

  ```
  # ifconfig eth0 192.168.100.10 netmask 255.255.255.0
  ```

- This example deactivates the Ethernet interface:

  ```
  # ifconfig eth0 down
  ```

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

  ```
  # ifconfig eth0 up
  ```

### Recovering a device

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

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

#### Example

```
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/0.0.d100/recover
```
Chapter 10. CTCM device driver

The CTCM device driver provides Channel-to-Channel (CTC) connections and CTC-based Multi-Path Channel (MPC) connections. The CTCM device driver is required by Communications Server for Linux.

CTC connections are high-speed point-to-point connections between two operating system instances on System z.

Communications Server for Linux uses MPC connections to connect SUSE Linux Enterprise Server to VTAM® on traditional mainframe operating systems.

Deprecated connection type

CTC connections are only supported for migration from earlier versions. Do not use for new development.

Features

The CTCM device driver provides:
- MPC connections to VTAM on traditional mainframe operating systems.
- ESCON or FICON CTC connections (standard CTC and basic CTC) between mainframes in basic mode, LPARs or z/VM guests.
- Virtual CTC/A connections between guests of the same z/VM system.
- CTC connections to other Linux instances or other mainframe operating systems.

What you should know about CTCM

This section provides information about CTCM group devices and the network interfaces that are created by the CTCM device driver.

CTCM group devices

The CTCM device driver requires two I/O subchannels for each interface, a read subchannel and a write subchannel. The device bus-IDs that correspond to the two subchannels must be configured for control unit type 3088.
The device bus-IDs that correspond to the subchannel pair are grouped as one CTCM group device. There are no constraints on the device bus-IDs of read subchannel and write subchannel, in particular, it is possible to group non-consecutive device bus-IDs.

On the communication peer operating system instance, read and write subchannels are reversed. That is, the write subchannel of the local interface is connected to the read subchannel of the remote interface and vice-versa.

Depending on the protocol, the interfaces can be CTC interfaces or MPC interfaces. MPC interfaces are used by Communications Server for Linux and connect to peer interfaces that run under VTAM.

**Interface names assigned by the CTCM device driver**

When a CTCM group device is set online, the CTCM device driver automatically assigns an interface name to it. The interface name depends on the protocol.

If the protocol is set to 4, you get an MPC connection and the interface names are of the form mpc<n>.

If the protocol is set to 0, 1, or 3, you get a CTC connection and the interface name is of the form ctc<n>.

<n> is an integer that identifies the device. When the first device is set online it is assigned 0, the second is assigned 1, the third 2, and so on. The devices are counted separately for CTC and MPC.

**Network connections**

This section applies to CTC interfaces only.

If your CTC connection is to a router or z/VM TCP/IP service machine, you can connect to an external network.

![Figure 28. I/O subchannel interface](image)

**Further information**


For more information about FICON, see Redpaper *FICON CTC Implementation*, REDP-0158.
Setting up the CTCM device driver

There are no module parameters for the CTCM device driver. You need to load the ctcm module before using it. Load it with the `modprobe` command to ensure that any other required modules are loaded:

```
# modprobe ctcm
```

Working with the CTCM device driver

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

- Creating a CTCM group device
- Displaying the channel type
- Setting the protocol
- Setting a device online or offline
- Setting the maximum buffer size (CTC only)
- Activating and deactivating a CTC interface (CTC only)
- Recovering a lost CTC connection (CTC only)

See the Communications Server for Linux documentation for information on how to configure and activate MPC interfaces.

Creating a CTCM group device

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

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

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

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

```
/sys/bus/ccwgroup/drivers/ctcm/<read_device_bus_id>
```

This directory contains a number of attributes that determine the settings of the CTCM group device.

**Example**

Assuming that device bus-ID 0.0.2000 corresponds to a read subchannel:

```
# echo 0.0.2000,0.0.2001 > /sys/bus/ccwgroup/drivers/ctcm/group
```

This command results in the creation of the following directories in sysfs:

- `/sys/bus/ccwgroup/drivers/ctcm/0.0.2000`
- `/sys/bus/ccwgroup/devices/0.0.2000`
- `/sys/devices/cu3088/0.0.2000`
**Displaying the channel type**

Issue a command of this form to display the channel type of a CTCM group device:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/type
```

where `<device_bus_id>` is the device bus-ID that corresponds to the CTCM read channel. Possible values are: CTC/A, ESCON, and FICON.

**Example**

In this example, the channel type is displayed for a CTCM group device with device bus-ID 0.0.f000:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f000/type
ESCON
```

**Setting the protocol**

**Before you start**: The device must be offline while you set the protocol.

The type of interface depends on the protocol. Protocol 4 results in MPC interfaces with interface names mpc<n>. Protocols 0, 1, or 3 result in CTC interfaces with interface names of the form ctc<n>.

To choose a protocol set the protocol attribute to one of the following values:

- **0**: This protocol provides compatibility with peers other than OS/390®, or z/OS, for example, a z/VM TCP service machine. This is the default.
- **1**: This protocol provides enhanced package checking for Linux peers.
- **3**: This protocol provides for compatibility with OS/390 or z/OS peers.
- **4**: This protocol provides for MPC connections to VTAM on traditional mainframe operating systems.

Issue a command of this form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/protocol
```

**Example**

In this example, the protocol is set for a CTCM group device 0.0.2000:

```
# echo 4 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/protocol
```

**Setting a device online or offline**

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

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

Setting a group device online associates it with an interface name. Setting the group device offline and back online with the same protocol preserves the association with the interface name. If you change the protocol before setting the
group device back online, the interface name can change as described in "Interface names assigned by the CTCM device driver" on page 144.

Read /var/log/messages or issue dmesg to find out which interface name has been assigned to the group device. You will need to know the interface name to access the CTCM group device.

For each online interface, there is a symbolic link of the form /sys/class/net/<interface_name>/device in sysfs. You can confirm that you have found the correct interface name by reading the link.

**Example**

To set a CTCM device with bus ID 0.0.2000 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/online
# dmesg | fgrep "ch-0.0.2000"
mpc0: read: ch-0.0.2000, write: ch-0.0.2001, proto: 4
```

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

```
# readlink /sys/class/net/mpc0/device
../../../devices/cu3088/0.0.2000
```

To set group device 0.0.2000 offline issue:

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

### Setting the maximum buffer size

**Before you start:**

- This section applies to CTC interfaces only. MPC interfaces automatically use the highest possible maximum buffer size.
- The device must be online when setting the buffer size.

You can set the maximum buffer size for a CTC interface. The permissible range of values depends on the MTU settings. It must be in the range `<minimum MTU + header size>` to `<maximum MTU + header size>`. The header space is typically 8 byte. The default for the maximum buffer size is 32768 byte (32 KB).

Changing the buffer size is accompanied by an MTU size change to the value `<buffer size - header size>`.

To set the maximum buffer size issue a command of this form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/buffer
```

where `<value>` is the number of bytes you want to set. If you specify a value outside the valid range, the command is ignored.

**Example**

In this example, the maximum buffer size of a CTCM group device 0.0.f000 is set to 16384 byte.
Activating and deactivating a CTC interface

Before you start activating a CTC interface:

• This section applies to CTC interfaces only. For information about how to activate MPC interfaces see the Communications Server for Linux documentation.

• You need to know the interface name (see “Setting a device online or offline” on page 146).

Use ifconfig or an equivalent command to activate the interface:

```
# ifconfig <interface> <ip_address> pointopoint <peer_ip_address> mtu <max_transfer_unit>
```

Where:

`<interface>`

is the interface name that was assigned when the CTCM group device was set online.

`<ip_address>`

is the IP address you want to assign to the interface.

`<peer_ip_address>`

is the IP address of the remote side.

`<max_transfer_unit>`

is the size of the largest IP packet which may be transmitted. Be sure to use the same MTU size on both sides of the connection. The MTU must be in the range of 576 byte to 65,536 byte (64 KB).

To deactivate an interface issue a command of this form:

```
# ifconfig <interface> down
```

Examples

• This example activates a CTC interface ctc0 with an IP address 10.0.51.3 for a peer with address 10.0.50.1 and an MTU of 32760.

```
# ifconfig ctc0 10.0.51.3 pointopoint 10.0.50.1 mtu 32760
```

• This example deactivates ctc0:

```
# ifconfig ctc0 down
```
Recovering a lost CTC connection

This section applies to CTC interfaces only.

If one side of a CTC connection crashes, you cannot simply reconnect after a reboot. You also need to deactivate the interface on the crashed side's peer. Proceed like this:

1. Reboot the crashed side.
2. Deactivate the interface on the peer (see "Activating and deactivating a CTC interface" on page 148).
3. Activate the interface on the crashed side and on the peer (see "Activating and deactivating a CTC interface" on page 148).

If the connection is between a Linux instance and a non-Linux instance, activate the interface on the Linux instance first. Otherwise you can activate the interfaces in any order.

If the CTC connection is uncoupled, you must couple it again and re-configure the interface of both peers using `ifconfig` (see "Activating and deactivating a CTC interface" on page 148).

Scenarios

This section provides some typical scenarios for CTC connections:

- Connecting to a peer in a different LPAR
- Connecting a Linux guest to a peer guest in the same z/VM

Connecting to a peer in a different LPAR

A Linux instance and a peer run in LPAR mode on the same or on different mainframes and are to be connected with a CTC FICON or CTC ESCON network interface.

Assumptions:

- Locally, the read and write channels have been configured for type 3088 and use device bus-IDs 0.0.f008 and 0.0.f009.
- IP address 10.0.50.4 is to be used locally and 10.0.50.5 for the peer.

Figure 29. CTC scenario with peer in a different LPAR

1. Create a CTCM group device. Issue:
2. Confirm that the device uses CTC FICON or CTC ESCON:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/type
ESCON
```

In this example, ESCON is used. You would proceed the same for FICON.

3. Select a protocol. The choice depends on the peer.

<table>
<thead>
<tr>
<th>If the peer is ...</th>
<th>Choose ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>1</td>
</tr>
<tr>
<td>z/OS or OS/390</td>
<td>3</td>
</tr>
<tr>
<td>Any other operating system</td>
<td>0</td>
</tr>
</tbody>
</table>

Assuming that the peer is Linux:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/protocol
```

4. Set the CTCM group device online and find out the assigned interface name:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/online
# dmesg | fgrep "ch-0.0.f008"
ctc0: read: ch-0.0.f008, write: ch-0.0.f009, proto: 1
```

In the example, the interface name is ctc0.

5. Assure that the peer interface is configured.

6. Activate the interface locally and on the peer. If you are connecting two Linux instances, either instance can be activated first. If the peer is not Linux, activate the interface on Linux first. To activate the local interface:

```
# ifconfig ctc0 10.0.50.4 pointopoint 10.0.50.5
```

### Connecting a Linux guest to a peer guest in the same z/VM

A Linux instance is running as a z/VM guest and to be connected to another guest of the same z/VM using a virtual CTC/A connection.

**Assumptions:**
- The guest ID of the peer is “guestp”.
- A separate subnet has been obtained from the TCP/IP network administrator. IP addresses 10.0.100.100 and 10.0.100.101 are to be used by the Linux guest and the peer, respectively.
1. Define two virtual channels to your user ID. The channels can be defined in the VM User Directory using directory control SPECIAL statements, for example:

```plaintext
special f004 ctca
special f005 ctca
```

Alternatively, you can use the CP commands:

```plaintext
# define ctca as f004
# define ctca as f005
```

from the console of the running CMS machine (preceded by #CP if necessary), or from an EXEC file (such as PROFILE EXEC A).

2. Assure that the peer interface is configured.

3. Connect the virtual channels. Assuming that the read channel on the peer corresponds to device number 0xf010 and the write channel to 0xf011 issue:

```plaintext
# couple f004 to guestp f011
# couple f005 to guestp f010
```

Be sure that you couple the read channel to the peers write channel and vice-versa.

4. From your booted Linux instance, create a CTCM group device. Issue:

```plaintext
# echo 0.0.f004,0.0.f005 > /sys/bus/ccwgroup/drivers/ctcm/group
```

5. Confirm that the group device is a CTC/A device:

```plaintext
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/type
CTC/A
```

6. Select a protocol. The choice depends on the peer.

<table>
<thead>
<tr>
<th>If the peer is ...</th>
<th>Choose ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>1</td>
</tr>
<tr>
<td>z/OS or OS/390</td>
<td>3</td>
</tr>
<tr>
<td>Any other operating system</td>
<td>0</td>
</tr>
</tbody>
</table>

Assuming that the peer is Linux:

```plaintext
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/protocol
```
7. Set the CTCM group device online and find out the assigned interface name:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/online
# dmesg | fgrep "ch-0.0.f004"
ctc1: read: ch-0.0.f004, write: ch-0.0.f005, proto: 1
```

In the example, the interface name is ctc1.

8. Activate the interface locally and on the peer. If you are connecting two Linux instances, either can be activated first. If the peer is not Linux, activate the local interface first. To activate the local interface:

```
# ifconfig ctc1 10.0.100.100 pointopoint 10.0.100.101
```

Be sure that the MTU on both sides of the connection is the same. If necessary change the default MTU (see "Activating and deactivating a CTC interface" on page 148).

9. Ensure that the buffer size on both sides of the connection is the same. For the Linux side see "Setting the maximum buffer size" on page 147 if the peer is not Linux, refer to the respective operating system documentation.
Chapter 11. NETIUCV device driver

Deprecated device driver
NETIUCV connections are only supported for migration from earlier versions. Do not use for new development.

The Inter-User Communication Vehicle (IUCV) is a VM communication facility that enables a program running in one z/VM guest to communicate with another z/VM guest, or with a control program, or even with itself.

The NETIUCV device driver is a network device driver, that uses IUCV to connect Linux guests running on different z/VM user IDs, or to connect a Linux guest to another z/VM guest such as a TCP/IP service machine.

Features

The NETIUCV device driver supports the following functions:
• Multiple output paths from a Linux guest
• Multiple input paths to a Linux guest
• Simultaneous transmission and reception of multiple messages on the same or different paths
• Network connections via a TCP/IP service machine gateway
• Internet Protocol, version 4 (IPv4) only

What you should know about IUCV

This section provides information on IUCV devices and interfaces.

IUCV direct and routed connections

The NETIUCV device driver uses TCP/IP over z/VM virtual communications. The communication peer is a guest of the same z/VM or the z/VM control program. No subchannels are involved.

If your IUCV connection is to a router, the peer can be remote and connected through an external network.
IUCV interfaces and devices

The NETIUCV device driver uses the base name iucv<\(n\)> for its interfaces. When the first IUCV interface is created (see "Creating an IUCV device" on page 155) it is assigned the name iucv0, the second is assigned iucv1, the third iucv2, and so on.

For each interface, a corresponding IUCV device is created in sysfs at /sys/bus/iucv/devices/netiucv<\(n\)> where <\(n\)> is the same index number that also identifies the corresponding interface.

For example, interface iucv0 corresponds to device name netiucv0, iucv1 corresponds to netiucv1, iucv2 corresponds to netiucv2, and so on.

Further information

The standard definitions in the z/VM TCP/IP configuration files apply.

For more information of the z/VM TCP/IP configuration see: z/VM TCP/IP Planning and Customization, SC24-6125.

Setting up the NETIUCV device driver

There are no module parameters for the NETIUCV device driver. This section describes how to load the netiucv module. It also explains how to enable a z/VM guest virtual machine for IUCV.

Loading the IUCV modules

The NETIUCV device driver has been compiled as a separate module that you need to load before you can work with IUCV devices. Use modprobe to load the module to ensure that any other required modules are also loaded.

# modprobe netiucv

Enabling your z/VM guest for IUCV

To enable your z/VM guest for IUCV add the following statements to your z/VM USER DIRECT entry:

IUCV ALLOW
IUCV ANY
Working with the NETIUCV device driver

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

- **Creating an IUCV device**
- **Changing the peer**
- **Setting the maximum buffer size**
- **Activating an interface**
- **Deactivating and removing an interface**

## Creating an IUCV device

To define an IUCV device write the user ID of the peer z/VM guest to /sys/bus/iucv/drivers/netiucv/connection.

Issue a command of this form:

```
# echo <peer_id> > /sys/bus/iucv/drivers/netiucv/connection
```

where `<peer_id>` is the guest ID of the z/VM guest you want to connect to. The NETIUCV device driver interprets the ID as uppercase.

**Result:** An interface iucv<n> is created and the following corresponding sysfs directories:

- /sys/bus/iucv/devices/netiucv<n>
- /sys/devices/iucv/netiucv<n>
- /sys/class/net/iucv<n>

<n> is an index number that identifies an individual IUCV device and its corresponding interface. You can use the attributes of the sysfs entry to configure the device.

To verify that an index number corresponds to a given guest ID read the name attribute. Issue a command of this form:

```
# cat /sys/bus/iucv/drivers/netiucv/netiucv<n>/user
```

**Example**

To create an IUCV device to connect to a z/VM guest with a guest user ID “LINUXP” issue:

```
# echo linuxp > /sys/bus/iucv/drivers/netiucv/connection
```

If this is the first IUCV device to be created, the corresponding interface name is iucv0. To confirm that this is the interface that connects to “LINUXP”:

```
# cat /sys/bus/iucv/drivers/netiucv/netiucv0/user
linuxp
```

## Changing the peer

**Before you start:** The interface must not be active when changing the name of the peer z/VM guest.
You can change the z/VM guest that an interface connects to. To change the peer z/VM guest issue a command of this form:

```
# echo <peer_ID> > /sys/bus/iucv/drivers/netiucv/netiucv<n>/user
```

where:

- `<peer_ID>`
  is the z/VM guest ID of the new communication peer. The value must be a valid guest ID. The NETIUCV device driver interprets the ID as uppercase.

- `<n>`
  is an index that identifies the IUCV device and the corresponding interface.

**Example**

In this example, “LINUX22” is set as the new peer z/VM guest.

```
# echo linux22 > /sys/bus/iucv/drivers/netiucv/netiucv0/user
```

### Setting the maximum buffer size

The upper limit for the maximum buffer size is 32768 bytes (32 KB). The lower limit is 580 bytes in general and in addition, if the interface is up and running `<current MTU + header size>`. The header space is typically 4 bytes.

Changing the buffer size is accompanied by an mtu size change to the value `<buffer size - header size>`.

To set the maximum buffer size issue a command of this form:

```
# echo <value> > /sys/bus/iucv/drivers/netiucv/netiucv<n>/buffer
```

where:

- `<value>`
  is the number of bytes you want to set. If you specify a value outside the valid range, the command is ignored.

- `<n>`
  is an index that identifies the IUCV device and the corresponding interface.

**Note:** If IUCV performance deteriorates and IUCV issues “out of memory” messages on the console, consider using a buffer size less than 4K.

**Example**

In this example, the maximum buffer size of an IUCV device netiucv0 is set to 16384 byte.

```
# echo 16384 > /sys/bus/iucv/drivers/netiucv/netiucv0/buffer
```

### Activating an interface

Use `ifconfig` or an equivalent command to activate an interface.
ifconfig syntax for an IUCV connection

```
ifconfig <interface> <ip_address> pointopoint <peer_ip_address>
```

```
mtu 9216
```

```
mtu <max_transfer_unit> netmask <mask_value>
```

where:

- `<interface>` is the interface name.
- `<ip_address>` is the IP address of your Linux guest.
- `<peer_ip_address>` for direct connections this is the IP address of the communication peer; for routed connections this is the IP address of the TCP/IP service machine or Linux router to connect to.
- `<max_transfer_unit>` is the size in byte of the largest IP packets which may be transmitted. The default is 9216. The valid range is 576 through 32764.

Note: An increase in buffer size is accompanied by an increased risk of running into memory problems. Thus a large buffer size increases speed of data transfer only if no “out of memory”-conditions occur.

- `<mask_value>` is a mask to identify the addresses served by this connection. Applies to routed connections only.

For more details, refer to the `ifconfig` man page.

For routed connections, you need to set up a route. Issue commands of this form:

```
# route add -net default <interface>
# inetd
```

Example

This example activates a connection to a TCP/IP service machine with IP address 1.2.3.200 using a maximum transfer unit of 32764 bytes.

```
# ifconfig iucv1 1.2.3.100 pointopoint 1.2.3.200 mtu 32764 netmask 255.255.255.0
# route add -net default iucv1
# inetd
```

Deactivating and removing an interface

You deactivate an interface with `ifconfig` or an equivalent command. Issue a command of this form:

```
# ifconfig <interface> down
```
where `<interface>` is the name of the interface to be deactivated.

You can remove the interface and its corresponding IUCV device by writing the interface name to the NETIUCV device driver's remove attribute. Issue a command of this form:

```
# echo <interface> > /sys/bus/iucv/drivers/netiucv/remove
```

where `<interface>` is the name of the interface to be removed. The interface name is of the form `iucv<n>`.

After the interface has been removed the interface name can be assigned again as interfaces are activated.

**Example**

This Example deactivates and removes an interface `iucv0` and its corresponding IUCV device:

```
# ifconfig iucv0 down
# echo iucv0 > /sys/bus/iucv/drivers/netiucv/remove
```

---

**Scenario: Setting up an IUCV connection to a TCP/IP service machine**

Two Linux guests with guest IDs “LNX1” and “LNX2” are to be connected through a TCP/IP service machine with guest ID “VMTCPIP”. Both Linux guests and the service machine all run in the same z/VM. A separate IP subnet (different from the subnet used on the LAN) has been obtained from the network administrator. IP address 1.2.3.4 is assigned to guest “LNX1”, 1.2.3.5 is assigned to guest “LNX2”, and 1.2.3.10 is assigned to the service machine.

**Setting up the service machine**

Proceed like this to set up the service machine:

1. For each guest that is to have an IUCV connection to the service machine add a home entry, device, link, and start statement to the service machine's PROFILE TCPIP file. The statements have the form:

```
Home
<ip_address1> <link_name1>
<ip_address2> <link_name2>
...

Device <device_name> IUCV 0 0 <guest_ID> A
Link <link_name1> IUCV 0 <device_name>
Device <device_name2> IUCV 0 0 <guest_ID> A
Link <link_name2> IUCV 0 <device_name2>
```

```
where

<i>ip_address1>, <ip_address2>
    is the IP address of a Linux guest.

<i>link_name1>, <link_name2>, ...
    are variables that associate the link statements with the respective home statements.

<i>device_name1>, <device_name2>, ...
    are variables that associate the device statements with the respective link statements and start commands.

<i>guest_ID1>, <guest_ID1>, ...
    are the guest IDs of the connected Linux guests.

In our example, the PROFILE TCPIP entries for our example might look of this form:

```
Home
  1.2.3.4 LNK1
  1.2.3.5 LNK2

Device DEV1 IUCV 0 0 LNX1 A
Link LNK1 IUCV 0 DEV1

Device DEV2 IUCV 0 0 LNX2 A
Link LNK2 IUCV 0 DEV2

Start DEV1
Start DEV2
```

2. Add the necessary z/VM TCP/IP routing statements (BsdRoutingParms or Gateway). Use an MTU size of 9216 and a point-to-point host route (subnet mask 255.255.255.255). If you use dynamic routing, but do not wish to run routed or gated on Linux, update the z/VM ETC GATEWAYS file to include "permanent" host entries for each Linux guest.

3. Bring these updates online by using OBEYFILE or by recycling TCPIP and/or ROUTED as needed.

**Setting up the Linux guest LNX1**

Proceed like this to set up the IUCV connection on the Linux guest:

1. Set up the NETIUCV device driver as described in "Setting up the NETIUCV device driver" on page 154.

2. Create an IUCV interface for connecting to the service machine:

```
# echo VMTCPIP /sys/bus/iucv/drivers/netiucv/connection
```

This creates an interface, for example, iucv0, with a corresponding IUCV device and a device entry in sysfs /sys/bus/iucv/devices/netiucv0.

3. The peer, LNX2 is set up accordingly. When both interfaces are ready to be connected to, activate the connection.
The peer, LNX2, is set up accordingly.
Chapter 12. CLAW device driver

**Deprecated device driver**

CLAW connections are only supported for migration from earlier versions. Do not use for new development.

Common Link Access to Workstation (CLAW) is a point-to-point protocol. A CLAW device is a channel connected device that supports the CLAW protocol. CLAW devices can connect your SUSE Linux Enterprise Server instance to a communication peer, for example, on a RISC System/6000 (RS/6000®) or on a Cisco Channel Interface Processor (CIP).

### Features

The CLAW device driver supports the following devices and functions:

- The CLAW driver supports up to 256 devices.

### What you should know about the CLAW device driver

This section provides information about CLAW group devices and interfaces.

#### CLAW group devices

The CLAW device driver requires two I/O subchannels for each CLAW interface, a read subchannel and a write subchannel. The corresponding bus-IDs must be configured for control unit type 3088.

![Diagram](image)

*Figure 33. I/O subchannel interface*

The device bus-IDs that correspond to the subchannel pair are grouped as one CLAW group device. The device bus-IDs can be any consecutive device bus-IDs where the read subchannel is the lower of the two IDs.

The read subchannel is linked to the write subchannel on the connected RS/6000 or CIP and vice versa.

#### CLAW interface names

When a CLAW group device is set online, the CLAW device driver automatically assigns an interface name to it. The interface names are of the form claw<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.
MTU size

You can set the MTU when you activate your CLAW group device (see “Activating a CLAW group device” on page 165).

The following apply to setting the MTU:
- The default MTU is 4096 byte.
- If the MTU of the attached CLAW interface on the RS/6000 or CIP is less than 4096 byte, it can be advantageous to match the MTU of the CLAW device to this lower value.
- You cannot set an MTU that is greater than the buffer size. The buffer size is 32 kilobyte for connection type PACKED (see “Setting the connection type” on page 163) and 4 kilobyte otherwise.
- The maximum MTU you can set is 4096 byte.

Setting up the CLAW device driver

There are no module parameters for the CLAW device driver.

The CLAW component is compiled as a separate module that you need to load before you can work with CLAW group devices. Load the claw module with the modprobe command to ensure that any other required modules are loaded:

```
# modprobe claw
```

Working with the CLAW device driver

This section describes typical tasks that you need to perform when working with CLAW devices.
- Creating a CLAW group device
- Setting the host and adapter name
- Setting the connection type
- Setting the number of read and write buffers
- Setting a CLAW group device online or offline
- Activating a CLAW group device

Creating a CLAW group device

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

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

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

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

```
/sys/bus/ccwgroup/drivers/claw/<read_device_bus_id>
```

This directory contains a number of attributes that determine the settings of the CLAW group device.
Assuming that device bus-ID 0.0.2d00 corresponds to a read subchannel:

```
# echo 0.0.2d00,0.0.2d01 > /sys/bus/ccwgroup/drivers/claw/group
```

This command results in the creation of the following directories in sysfs:

- /sys/bus/ccwgroup/drivers/claw/0.0.2d00
- /sys/bus/ccwgroup/devices/0.0.2d00
- /sys/devices/cu3088/0.0.2d00

### Setting the host and adapter name

Host and adapter names identify the communication peers to one another. The local host name must match the remote adapter name and vice versa.

Set the host and adapter name before you set the CLAW group device online. Changing a name for an online device does not take effect until the device is set offline and back online.

To set the host name issue a command of this form:

```
# echo <host> > /sys/bus/ccwgroup/drivers/claw/<device_bus_id>/host_name
```

To set the adapter name issue a command of this form:

```
# echo <adapter> > /sys/bus/ccwgroup/drivers/claw/<device_bus_id>/adapter_name
```

where `<host>` is the host name and `<adapter>` the adapter name. The names can be from 1 to 8 characters and are case sensitive.

#### Example

In this example, the host name for a claw group device with device bus-ID 0.0.d200 is set to “LNX1” and the adapter name to “RS1”.

```
# echo LNX1 > /sys/bus/ccwgroup/drivers/claw/0.0.d200/host_name
# echo RS1 > /sys/bus/ccwgroup/drivers/claw/0.0.d200/adapter_name
```

To make this connection work, the adapter name on the communication peer must be set to “LNX1” and the host name to “RS1”.

### Setting the connection type

The connection type determines the packing method used for outgoing packets. The connection type must match the connection type on the connected RS/6000 or CIP.

Set the connection type before you set the CLAW group device online. Changing the connection type for an online device does not take effect until the device is set offline and back online.

To set the connection type issue a command of this form:

```
# echo <type> > /sys/bus/ccwgroup/drivers/claw/<device_bus_id>/api_type
```
where `<type>` can be either of:

- **IP** to use the IP protocol for CLAW.
- **PACKED** to use enhanced packing with TCP/IP for better performance.
- **TCP/IP** to use the TCP/IP protocol for CLAW.

**Example**

In this example, the connection type “PACKED” is set for a CLAW group device with device bus-ID 0.0.d200.

```
# echo PACKED > /sys/bus/ccwgroup/drivers/claw/0.0.d200/api_type
```

### Setting the number of read and write buffers

You can allocate the number of read buffers and the number of write buffers for your CLAW group device separately. Set the number of buffers before you set the CLAW group device online. You can change the number of buffers at any time, but new values for an online device do not take effect until the device is set offline and back online.

To set the number of read buffers issue a command of this form:

```
# echo <number> > /sys/bus/ccwgroup/drivers/claw/<device_bus_id>/read_buffer
```

To set the number of write buffers issue a command of this form:

```
# echo <number> > /sys/bus/ccwgroup/drivers/claw/<device_bus_id>/write_buffer
```

where `<number>` is the number of buffers you want to allocate. The valid range of numbers you can specify is the same for read and write buffers. The range depends on your connection type (see “Setting the connection type” on page 163):

- For connection type PACKED you can allocate 2 to 64 buffers of 32 KB.
- For the other connection types you can allocate 2 to 512 buffers of 4 KB.

**Example**

In this example, 4 read buffers and 5 write buffers are allocated to a claw group device with device bus-ID 0.0.d200.

```
# echo 4 > /sys/bus/ccwgroup/drivers/claw/0.0.d200/read_buffer
# echo 5 > /sys/bus/ccwgroup/drivers/claw/0.0.d200/write_buffer
```

### Setting a CLAW group device online or offline

To set a CLAW group device online set the online device group attribute to “1”. To set a CLAW group device offline set the online device group attribute to “0”. Issue a command of this form:

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

Setting a device online for the first time 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 CLAW 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 CLAW device with bus ID 0.0.d200 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/claw/0.0.d200/online
# dmesg
claw0:readsize=4096 writesize=4096 readbuffer=4 writebuffer=5 read=0xd200 write=0xd201
claw0:host_name:LNX1 , adapter_name :RS1 api_type: PACKED
```

The interface name that has been assigned to the CLAW group device in the example is claw0. To confirm that this is the correct name for our group device issue:

```
# readlink /sys/class/net/claw0/device
../../../devices/cu3088/0.0.d200
```

To set the same device offline issue:

```
# echo 0 > /sys/bus/ccwgroup/drivers/claw/0.0.d200/online
```

**Activating a CLAW group device**

You can activate a CLAW group device with `ifconfig` or an equivalent command. See “MTU size” on page 162 for information on possible MTU settings.

**Example**

```c
ifconfig claw0 10.22.34.5 netmask 255.255.255.248 dstaddr 10.22.34.6
```
Part 4. z/VM virtual server integration

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

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

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 release notes at
www.novell.com/linux/releasenotes/s390x/SUSE-SLES/11/

Chapter 13. z/VM concepts ................................. 169
Performance monitoring for z/VM guests .................. 169
Cooperative memory management background .......... 171

Chapter 14. Writing kernel APPLDATA records .......... 173
Setting up the APPLDATA record support ................ 173
Working with the APPLDATA record support ............ 173
APPLDATA monitor record layout ....................... 175
Programming interfaces ................................. 178

Chapter 15. Writing application APPLDATA records .... 179
Features ............................................. 179
What you should know about the monitor stream application device driver 179
Setting up the monitor stream application device driver . 179
Working with the monitor stream application device driver . 180

Chapter 16. Reading z/VM monitor records ............... 183
Features ............................................. 183
What you should know about the z/VM *MONITOR record reader device driver 183
Setting up the z/VM *MONITOR record reader device driver 184
Working with the z/VM *MONITOR record reader device driver 185

Chapter 17. z/VM recording device driver .................. 189
Features ............................................. 189
What you should know about the z/VM recording device driver 189
Setting up the z/VM recording device driver .......... 190
Working with z/VM recording devices .................... 190
Scenario: Connecting to the *ACCOUNT service ......... 193

Chapter 18. z/VM unit record device driver ............... 197
What you should know about the z/VM unit record device driver 197
Working with the vmur device driver .................... 197

Chapter 19. z/VM DCSS device driver ....................... 199
Features ............................................. 199
What you should know about DCSS ...................... 199
Setting up the DCSS device driver ...................... 200
Avoiding overlaps with your Linux guest storage ....... 201
Working with the DCSS device driver .................. 202
Changing the contents of a DCSS ....................... 206
Chapter 13. z/VM concepts

This chapter contains information that is not strictly needed to run the functionality in question, however, it might help you understand some of the background.

Performance monitoring for z/VM guests

You can monitor the performance of Linux instances or other VM guests with performance monitoring tools on z/VM or on Linux. These tools can be your own, IBM tools such as the Performance Toolkit for VM, or third party tools. The guests being monitored require agents that write monitor data.

Monitoring on z/VM

z/VM monitoring tools need to read performance data. In the case of monitoring Linux guests, this data is APPLDATA monitor records. Linux instances need to write these records for the tool to read, as shown in Figure 34.

![Figure 34. Linux instances write APPLDATA records for performance monitoring tools](image)

Both user space applications and the Linux kernel can write performance data to APPLDATA records. Applications use the monwriter device driver to write APPLDATA records. The Linux kernel can be configured to collect system level data such as memory, CPU usage, and network related data, and write it to data records.

For file system size data there is a command, mon_fsstatd, a user space tool that uses the monwriter device driver to write file system size information as defined records.

For process data there is a command, mon_procd, a user space tool that uses the monwriter device driver to write system information as defined records.

In summary, SUSE Linux Enterprise Server 11 for System z supports writing and collecting performance data as follows:

- The Linux kernel can write z/VM monitor data for Linux instances, see Chapter 14, “Writing kernel APPLDATA records,” on page 173.
Applications running on Linux guests can write z/VM monitor data, see Chapter 15, “Writing application APPLDATA records,” on page 179.

You can collect monitor file system size information, see mon.fsstatd – Monitor z/VM guest file system size on page 385.

You can collect system information on up to 100 concurrent running processes, see mon.procd – Monitor Linux guest” on page 390.

Monitoring on Linux

For performance monitoring on Linux, you can use a tool such as Tivoli® OMEGAMON®, or write your own software, and set up a Linux instance to read the monitor data as shown in Figure 35. A Linux instance can read the monitor data using the monreader device driver.

Figure 35. Performance monitoring using monitor DCSS data

In summary, SUSE Linux Enterprise Server 11 for System z supports reading performance data as follows:

- Read access to z/VM monitor data for Linux guests, see Chapter 16, “Reading z/VM monitor records,” on page 183.

Further information

- Refer to z/VM Getting Started with Linux on System z, SC24-6096, the chapter on monitoring performance for information on using the CP Monitor and the Performance Toolkit for VM.

- Refer to z/VM: Saved Segments Planning and Administration, SC24-6116 for general information on DCSSs (z/VM keeps monitor records in a DCSS).

- Refer to z/VM: Performance, SC24-6109 for information on how to create a monitor DCSS.

- Refer to z/VM: CP Commands and Utilities 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/mon520/index.html and refer to Chapter 14, “Writing kernel APPLDATA records,” on page 173.

For more information about performance monitoring on z/VM, visit ibm.com/vm/perf
Cooperative memory management background

This section gives some background information about cooperative memory management (CMM, or "cmm1"). For information about setting it up, see Chapter 24, "Cooperative memory management," on page 221.

In a virtualized environment it is common practice to give the virtual machines more memory than is actually available to the hypervisor. Linux has the tendency to use all of its available memory. As a result, the hypervisor (z/VM) might start swapping.

To avoid excessive z/VM swapping the available Linux guest memory can be reduced. To reduce Linux guest memory size CMM allocates pages to page pools that make the pages unusable to Linux. Two such page pools exist for a Linux guest, as shown in Figure 36.

The two page pools are:

**A static page pool**

The page pool is controlled by a resource manager that changes the pool size at intervals according to guest activity as well as overall memory usage on z/VM (see Figure 37).

**A timed page pool**

Pages are released from this pool at a speed set in the release rate (see Figure 38 on page 172). According to guest activity and overall memory usage on z/VM, a resource manager adds pages at intervals. If no pages are added and the release rate is not zero, the pool will empty.
The external resource manager that controls the pools can be the z/VM resource monitor (VMRM) or a third party systems management tool.

VMRM controls the pools over a message interface. Setting up the external resource manager is beyond the scope of this book. For more information, see the chapter on VMRM in z/VM Performance, SC24-6109.

Third party tools can use a Linux daemon that receives commands for the memory allocation through TCP/IP. The daemon, in turn, uses the a /proc-based interface. You can use the /proc interface to read the pool sizes. This is useful for diagnostics.

Figure 38. Timed page pool. Pages are freed at a set release rate.

The external resource manager that controls the pools can be the z/VM resource monitor (VMRM) or a third party systems management tool.

VMRM controls the pools over a message interface. Setting up the external resource manager is beyond the scope of this book. For more information, see the chapter on VMRM in z/VM Performance, SC24-6109.

Third party tools can use a Linux daemon that receives commands for the memory allocation through TCP/IP. The daemon, in turn, uses the a /proc-based interface. You can use the /proc interface to read the pool sizes. This is useful for diagnostics.
Chapter 14. Writing kernel APPLDATA records

z/VM is a convenient point for collecting VM guest performance data and statistics for an entire server farm. Linux instances can export such data to z/VM by means of APPLDATA monitor records. z/VM regularly collects these records. The records are then available to z/VM performance monitoring tools.

A virtual CPU timer on the Linux guest to be monitored controls when data is collected. The timer only accounts for busy time to avoid unnecessarily waking up an idle guest. The APPLDATA record support comprises several modules. A base module provides an intra-kernel interface and the timer function. The intra-kernel interface is used by data gathering modules that collect actual data and determine the layout of a corresponding APPLDATA monitor record (see “APPLDATA monitor record layout” on page 175).

For an overview of performance monitoring support, see “Performance monitoring for z/VM guests” on page 169.

Setting up the APPLDATA record support

There are no module parameters for the monitor stream support. This section describes how to load those components of the support that have been compiled as separate modules and how to set up your VM guest for the APPLDATA record support.

Loading data gathering modules

The data gathering components 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.

### APPLDATA record support module parameter syntax

```plaintext
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 your Linux guest for data gathering ensure that the Linux guest directory includes the option APPLMON.

Working with the APPLDATA record support

You control the monitor stream support through the procfs. You can set the timer interval and switch on or off data collection. APPLDATA monitor records are produced if both a particular data gathering module and the monitoring support in general are switched on.
Switching the support on or off
You switch on or off the monitoring support by writing “1” (on) or “0” (off) to /proc/sys/appldata/timer.

To read the current setting issue:
```
# cat /proc/sys/appldata/timer
```

To switch on the monitoring support issue:
```
# echo 1 > /proc/sys/appldata/timer
```

To switch off the monitoring support issue:
```
# echo 0 > /proc/sys/appldata/timer
```

Activating or deactivating individual data gathering modules
You can activate or deactivate the data gathering modules individually. Each data gathering module has a procfs entry that contains a value “1” if the module is active and “0” if the module is inactive. The entries are:
- /proc/sys/appldata/mem for the memory data gathering module
- /proc/sys/appldata/os for the CPU data gathering module
- /proc/sys/appldata/net_sum for the net data gathering module

To check if a module is active look at the content of the corresponding procfs entry.

To activate a data gathering module write “1” to the corresponding procfs entry. To deactivate a data gathering module write “0” to the corresponding procfs entry. Issue a command of this form:
```
# echo <flag> > /proc/sys/appldata/<data_type>
```

where `<data_type>` is one of mem, os, or net_sum.

Note: An active data gathering module produces APPLDATA monitor records only if the monitoring support is switched on (see “Switching the support on or off”).

Example
To find out if memory data gathering is active issue:
```
# cat /proc/sys/appldata/mem
0
```

In the example, memory data gathering is off. To activate memory data gathering issue:
```
# echo 1 > /proc/sys/appldata/mem
```

To deactivate the memory data gathering module issue:
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:

```bash
# 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:

```bash
# 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:

```bash
# 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 33 on page 176)
- Processor data (see Table 34 on page 177)
- Networking (see Table 35 on page 178)

z/VM can identify the records by their unique product ID. The product ID is an EBCDIC string of this form: “LINUXKRNL<record ID>260100”. The `<record ID>` is treated as a byte value, not a string.

The records contain data of the following types:

- `u32` unsigned 4 byte integer
- `u64` unsigned 8 byte integer
<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0</td>
<td>u64 timestamp</td>
<td>TOD timestamp generated on the Linux side after record update</td>
</tr>
<tr>
<td>8</td>
<td>0x8</td>
<td>u32 sync_count_1</td>
<td>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.</td>
</tr>
<tr>
<td>12</td>
<td>0xC</td>
<td>u32 sync_count_2</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0x10</td>
<td>u64 pgpgin</td>
<td>Data read from disk (in KB)</td>
</tr>
<tr>
<td>24</td>
<td>0x18</td>
<td>u64 pgpgout</td>
<td>Data written to disk (in KB)</td>
</tr>
<tr>
<td>32</td>
<td>0x20</td>
<td>u64 pswpin</td>
<td>Pages swapped in</td>
</tr>
<tr>
<td>40</td>
<td>0x28</td>
<td>u64 pswpout</td>
<td>Pages swapped out</td>
</tr>
<tr>
<td>48</td>
<td>0x30</td>
<td>u64 sharedram</td>
<td>Shared RAM in KB, set to 0</td>
</tr>
<tr>
<td>56</td>
<td>0x38</td>
<td>u64 totalram</td>
<td>Total usable main memory size in KB</td>
</tr>
<tr>
<td>64</td>
<td>0x40</td>
<td>u64 freeram</td>
<td>Available memory size in KB</td>
</tr>
<tr>
<td>72</td>
<td>0x48</td>
<td>u64 totalhigh</td>
<td>Total high memory size in KB</td>
</tr>
<tr>
<td>80</td>
<td>0x50</td>
<td>u64 freehigh</td>
<td>Available high memory size in KB</td>
</tr>
<tr>
<td>88</td>
<td>0x58</td>
<td>u64 bufferram</td>
<td>Memory reserved for buffers, free cache in KB</td>
</tr>
<tr>
<td>96</td>
<td>0x60</td>
<td>u64 cached</td>
<td>Size of used cache, without buffers in KB</td>
</tr>
<tr>
<td>104</td>
<td>0x68</td>
<td>u64 totalswap</td>
<td>Total swap space size in KB</td>
</tr>
<tr>
<td>112</td>
<td>0x70</td>
<td>u64 freeswap</td>
<td>Free swap space in KB</td>
</tr>
<tr>
<td>120</td>
<td>0x78</td>
<td>u64 pgalloc</td>
<td>Page allocations</td>
</tr>
<tr>
<td>128</td>
<td>0x80</td>
<td>u64 pgmajfault</td>
<td>Page faults (major+minor)</td>
</tr>
<tr>
<td>136</td>
<td>0x88</td>
<td>u64 pgmajfault</td>
<td>Page faults (major only)</td>
</tr>
</tbody>
</table>
### Table 34. APPLDATA_OS_DATA record (Record ID 0x02)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type (size)</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>u64</td>
<td>timestamp</td>
<td>TOD timestamp generated on the Linux side after record update.</td>
</tr>
<tr>
<td>8</td>
<td>u32</td>
<td>sync_count_1</td>
<td>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.</td>
</tr>
<tr>
<td>12</td>
<td>u32</td>
<td>sync_count_2</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>u32</td>
<td>nr_cpus</td>
<td>Number of virtual CPUs.</td>
</tr>
<tr>
<td>20</td>
<td>u32</td>
<td>per_cpu_size</td>
<td>Size of the per_cpu_data for each CPU (= 36).</td>
</tr>
<tr>
<td>24</td>
<td>u32</td>
<td>cpu_offset</td>
<td>Offset of the first per_cpu_data (= 52).</td>
</tr>
<tr>
<td>28</td>
<td>u32</td>
<td>nr_running</td>
<td>Number of runnable threads.</td>
</tr>
<tr>
<td>32</td>
<td>u32</td>
<td>nr_threads</td>
<td>Number of threads.</td>
</tr>
<tr>
<td>36</td>
<td>u32</td>
<td>avenrun[3]</td>
<td>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.</td>
</tr>
<tr>
<td>48</td>
<td>u32</td>
<td>nr_iowait</td>
<td>Number of blocked threads (waiting for I/O).</td>
</tr>
<tr>
<td>52</td>
<td>u32</td>
<td>per_cpu_data</td>
<td>Time spent in user, kernel, idle, nice, etc for every CPU. See note 2 at the end of this table.</td>
</tr>
<tr>
<td>52</td>
<td>u32</td>
<td>per_cpu_user</td>
<td>Timer ticks spent in user mode.</td>
</tr>
<tr>
<td>56</td>
<td>u32</td>
<td>per_cpu_nice</td>
<td>Timer ticks spent with modified priority.</td>
</tr>
<tr>
<td>60</td>
<td>u32</td>
<td>per_cpu_system</td>
<td>Timer ticks spent in kernel mode.</td>
</tr>
<tr>
<td>64</td>
<td>u32</td>
<td>per_cpu_idle</td>
<td>Timer ticks spent in idle mode.</td>
</tr>
<tr>
<td>68</td>
<td>u32</td>
<td>per_cpu_irq</td>
<td>Timer ticks spent in interrupts.</td>
</tr>
<tr>
<td>72</td>
<td>u32</td>
<td>per_cpu_softirq</td>
<td>Timer ticks spent in softirqs.</td>
</tr>
<tr>
<td>76</td>
<td>u32</td>
<td>per_cpu_iowait</td>
<td>Timer ticks spent while waiting for I/O.</td>
</tr>
<tr>
<td>80</td>
<td>u32</td>
<td>per_cpu_steal</td>
<td>Timer ticks “stolen” by hypervisor.</td>
</tr>
<tr>
<td>84</td>
<td>u32</td>
<td>cpu_id</td>
<td>The number of this CPU.</td>
</tr>
</tbody>
</table>

**Notes:**

1. The following C-Macros are used inside Linux to transform these into values with two decimal places:
   ```c
   #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 35. APPLDATA_NET_SUM_DATA record (Record ID 0x03)

<table>
<thead>
<tr>
<th>Offset (Decimal)</th>
<th>Type (Hex)</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0</td>
<td>u64 timestamp</td>
<td>TOD timestamp generated on the Linux side after record update</td>
</tr>
<tr>
<td>8</td>
<td>0x8</td>
<td>u32 sync_count_1</td>
<td>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.</td>
</tr>
<tr>
<td>12</td>
<td>0xC</td>
<td>u32 sync_count_2</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0x10</td>
<td>u32 nr_interfaces</td>
<td>Number of interfaces being monitored</td>
</tr>
<tr>
<td>20</td>
<td>0x14</td>
<td>u32 padding</td>
<td>Unused. The next value is 64-bit aligned, so these 4 byte would be padded out by compiler</td>
</tr>
<tr>
<td>24</td>
<td>0x18</td>
<td>u64 rx_packets</td>
<td>Total packets received</td>
</tr>
<tr>
<td>32</td>
<td>0x20</td>
<td>u64 tx_packets</td>
<td>Total packets transmitted</td>
</tr>
<tr>
<td>40</td>
<td>0x28</td>
<td>u64 rx_bytes</td>
<td>Total bytes received</td>
</tr>
<tr>
<td>48</td>
<td>0x30</td>
<td>u64 tx_bytes</td>
<td>Total bytes transmitted</td>
</tr>
<tr>
<td>56</td>
<td>0x38</td>
<td>u64 rx_errors</td>
<td>Number of bad packets received</td>
</tr>
<tr>
<td>64</td>
<td>0x40</td>
<td>u64 tx_errors</td>
<td>Number of packet transmit problems</td>
</tr>
<tr>
<td>72</td>
<td>0x48</td>
<td>u64 rx_dropped</td>
<td>Number of incoming packets dropped because of insufficient space in Linux buffers</td>
</tr>
<tr>
<td>80</td>
<td>0x50</td>
<td>u64 tx_dropped</td>
<td>Number of outgoing packets dropped because of insufficient space in Linux buffers</td>
</tr>
<tr>
<td>88</td>
<td>0x58</td>
<td>u64 collisions</td>
<td>Number of collisions while transmitting</td>
</tr>
</tbody>
</table>

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 175 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 15. Writing application APPLDATA records

Applications can easily write monitor data in the form of APPLDATA records to the z/VM monitor stream by using the monitor stream application device driver. This character device enables writing of z/VM monitor APPLDATA records.

For an overview of performance monitoring support, see "Performance monitoring for z/VM guests" on page 169.

Features

The monitor stream application device driver provides the following functions:

- An interface to the z/VM monitor stream.
- A means of writing z/VM monitor APPLDATA records.

What you should know about the monitor stream application device driver

The monitor stream application device driver interacts with the z/VM monitor APPLDATA facilities for performance monitoring. A better understanding of these z/VM facilities might help when using this device driver.

Further information

- Refer to 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 Commands and Utilities Reference, SC24-6081 for information on the CP commands.
- Refer to z/VM Performance, SC24-6109 for information on monitor APPLDATA.

Setting up the monitor stream application device driver

This section describes the parameters that you can use to configure the monitor stream write support.

Module parameters

The monitor stream application device driver is compiled as a separate module that you need to load before you can work with it. This section describes how to load and configure the monwriter module.

Monitor stream application device driver module parameter syntax

```
insmod monwriter max_bufs=\<NUMBUFS>  
```

where NUMBUFS is the maximum number of monitor sample and configuration data buffers that can exist in the Linux guest at one time. The default is 255.
Example
To load the monwriter module and set the maximum number of buffers to NUMBUFS, use the following command:

```
# modprobe monwriter max_bufs=NUMBUFS
```

Setting up the user
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 Commands and Utilities Reference*, SC24-6081 for information on the CP MONITOR command.

Working with the monitor stream application device driver
This device driver writes to the z/VM monitor stream through the z/VM CP instruction DIAG X'DC'. See *z/VM CP Programming Services* 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 pppppppppnnnnnnn, where:

- `pppppppp` 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 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:

```c
/* 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:

```c
/* 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 16. Reading z/VM monitor records

Monitoring software on Linux can access z/VM guest data through the z/VM *MONITOR record reader device driver.

z/VM uses the z/VM monitor system service (*MONITOR) to collect monitor records from agents on its guests. z/VM writes the records to a discontiguous saved segment (DCSS). The z/VM *MONITOR record reader device driver uses IUCV to connect to *MONITOR and accesses the DCSS as a character device.

For an overview of performance monitoring support, see "Performance monitoring for z/VM guests" on page 169.

Features

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

- Read access to the z/VM *MONITOR DCSS.
- Reading *MONITOR records for z/VM.
- Access to *MONITOR records as described on: ibm.com/vm/pubs/ctlblk.html
- Access to the records provided by the Linux monitor stream (see Chapter 14, “Writing kernel APPLDATA records,” on page 173).

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

The data that is collected by *MONITOR depends on how you have set up the service. The z/VM *MONITOR record reader device driver only reads data from the monitor DCSS; it does not control the system service.

z/VM only supports a single monitor DCSS. All monitoring software that requires monitor records from z/VM uses the same DCSS to read *MONITOR data. Usually, a DCSS called “MONDCSS” is already defined and used by existing monitoring software. If this is the case, you must also use MONDCSS. See “Assuring that the DCSS is addressable for your Linux guest” on page 184 for information on how to check if MONDCSS exists.

Further information

- Refer to z/VM: 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 z/VM: CP Commands and Utilities 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 14, “Writing kernel APPLDATA records,” on page 173.
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 183 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
```

Assuring that the DCSS is addressable for your Linux guest

The DCSS address range must not overlap with the storage of your z/VM guest virtual machine. To find out the start and end address of the DCSS by issuing the following CP command from a CMS session with privilege class E:

```
#cp q nss map
```

the output gives you the start and end addresses of all defined DCSSs in units of 4 kilobyte pages:

```
00: FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP ...
00: 0011 MONDCSS CPDCSS N/A 09000 097FF SC R 00003 N/A N/A ...
```

If the DCSS overlaps with the guest storage follow the procedure in "Avoiding overlaps with your Linux guest storage" on page 201.

Specifying the monitor DCSS name

By default, the z/VM *MONITOR record reader device driver assumes that the monitor DCSS on z/VM is called MONDCSS. If you want to use a different DCSS name you need to specify it. Specify the DCSS name as a module parameter when you load the module.

Module parameter
This section describes how to load the monitor read support. It also tells you how to specify a DCSS name, if applicable.
Load the monitor read support module with `modprobe` to assure that any other required modules are also loaded. You need IUCV support if you want to use the monitor read support.

```plaintext
modprobe monreader
```

where `<dcss>` is the name of the DCSS that z/VM uses for the monitor records.

**Example:** To load the monitor read support module and specify MYDCSS as the DCSS issue:

```plaintext
modprobe monreader mondcss=mydcss
```

### z/VM *MONITOR record device node

SUSE Linux Enterprise Server creates a device node for you using udev. The device node is called `/dev/monreader` and is a miscellaneous character device that you can use to access the monitor DCSS.

### Working with the z/VM *MONITOR record reader device driver

This section describes how to work with the monitor read support.

- Opening and closing the character device
- Reading monitor records

#### Opening and closing the character device

Only one user can open the character device at any one time. Once you have opened the device you need to close it to make it accessible to other users.

The open function can fail (return a negative value) with one of the following values for errno:

**EBUSY**

The device has already been opened by another user.

**EIO**

No IUCV connection to the z/VM MONITOR system service could be established. An error message with an IPUSER SEVER code is printed into syslog. 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](http://ibm.com/vm/pubs/ctlblk.html)

The layout of the data stream provided by the monreader device is as follows:

```
... <0 byte read>
<first MCE>   /
<first set of records>   |... ...
          |   |- data set
<last MCE>   |
<last set of records>   /
<0 byte read>
...
```

There may be more than one combination of MCE and a corresponding record set within one data set. The end of each data set is indicated by a successful read with a return value of 0 (0 byte read). Received data is not to be considered valid unless a complete record set is read successfully, including the closing 0-Byte read. You are advised to always read the complete set into a user space buffer before processing the data.

When designing a buffer, allow for record sizes up to the size of the entire monitor DCSS, or use dynamic memory allocation. The size of the monitor DCSS will be printed into syslog after loading the module. You can also use the (Class E privileged) CP command Q NSS MAP to list all available segments and information about them (see "Assuring that the DCSS is addressable for your Linux guest" on page 184).

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 17. 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 191).

For general information about CP recording system services refer to z/VM CP Programming Services, SC24-6084.

What you should know about the z/VM recording device driver

The z/VM recording device driver is a character device driver that is grouped under the IUCV category of device drivers (see "Device categories" on page 7). There is one device for each recording service. The devices are created for you when the z/VM recording device driver module is loaded.

z/VM recording device nodes

Each recording service has a name that corresponds to the name of the service as shown in Table 36:

<table>
<thead>
<tr>
<th>z/VM recording service</th>
<th>Standard device name</th>
</tr>
</thead>
<tbody>
<tr>
<td>*LOGREC</td>
<td>logrec</td>
</tr>
<tr>
<td>*ACCOUNT</td>
<td>account</td>
</tr>
<tr>
<td>*SYMPTOM</td>
<td>symptom</td>
</tr>
</tbody>
</table>

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 39 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](http://ibm.com/vm/pubs/ctlblk.html).

---

Setting up the z/VM recording device driver

This section provides information on the guest authorization you need to be able to collect records and on how to load the device driver module.

Authorizing the Linux guest

The Linux guest must be authorized to use the z/VM RECORDING command. Depending on the z/VM environment, this could be either of the following authorization classes: A, B, C, E, or F.

The guest must also be authorized to connect to those IUCV services it needs to use.

Loading the z/VM recording device driver

There are no module parameters for the z/VM recording device driver.

You need to load the z/VM recording device driver module before you can work with z/VM recording devices. Load the vmlogradr module with the modprobe command to ensure that any other required modules are loaded in the correct order:

```
# modprobe vmlogradr
```

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 192](#)).

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 on page 192](#).
**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:
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 191) to enable automatic record collection while a device is open.

You can use a device’s autopurge attribute (see “Purging existing records” on page 192) 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:
   ```bash
   # cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
   ```
   The results depend on the system, but should be similar to the following:

<table>
<thead>
<tr>
<th>RECORDING</th>
<th>COUNT</th>
<th>LMT</th>
<th>USERID</th>
<th>COMMUNICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EREP ON</td>
<td>00000000</td>
<td>002</td>
<td>EREP</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>ACCOUNT ON</td>
<td>00001774</td>
<td>020</td>
<td>DISKACNT</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>SYMPTOM ON</td>
<td>00000000</td>
<td>002</td>
<td>OPERSYMP</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>ACCOUNT OFF</td>
<td>00000000</td>
<td>020</td>
<td>LINUX31</td>
<td>INACTIVE</td>
</tr>
</tbody>
</table>

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:
Switch autopurge and autorecord off:

```
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/autopurge
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/autorecording
```

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 OPERSYM ACTIVE
ACCOUNT OFF 00000009 020 LINUX31 INACTIVE
```

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 OPERSYM ACTIVE
ACCOUNT OFF 00000009 020 LINUX31 INACTIVE
```

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 00000000 002 EREP ACTIVE
ACCOUNT OFF 00000009 020 LINUX31 INACTIVE
```

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 00000000 002 EREP ACTIVE
ACCOUNT OFF 00000009 020 LINUX31 INACTIVE
```

Recording did not start, in the message logs you may find a message:
vmlogrdr: recording response: HCPCRCB887I 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 18. z/VM unit record device driver

The z/VM unit record device driver provides Linux access to virtual unit record devices. Unit record devices comprise punch card readers, card punches, and line printers. Linux access is limited to virtual unit record devices with default device types (2540 for reader and punch, 1403 for printer).

To write Linux files to the virtual punch or printer (that is, to the corresponding spool file queues) or to receive VM reader files (for example CONSOLE files) to Linux files, use the `vmur` command that is part of the s390-tools package (see “vmur - Work with z/VM spool file queues” on page 426).

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

The z/VM unit record device driver is compiled as a separate module, vmur.

z/VM unit record device nodes

When the vmur module is loaded, it registers a character device. The following device nodes are created for a unit record device when it is set online:

- Reader: /dev/vmrdr-0.0.<device_number>
- Punch: /dev/vmpun-0.0.<device_number>
- Printer: /dev/vmprt-0.0.<device_number>

Working with the vmur device driver

After loading the vmur module, the required virtual unit record devices need to be set online, for example:

```
chccwdev -e c /* set online attribute in /sys/bus/ccw/devices/0.0.000c */
chccwdev -e d /* set online attribute in /sys/bus/ccw/devices/0.0.000d */
chccwdev -e e /* set online attribute in /sys/bus/ccw/devices/0.0.000e */
```

When unloading vmur (with rmmod) the respective unit record device nodes must not be open, otherwise the error message "Module vmur is in use" is displayed.

Serialization is implemented per device; only one process can open a given device node at a given time.
Chapter 19. z/VM DCSS device driver

The z/VM discontiguous saved segments (DCSS) device driver provides disk-like fixed block access to z/VM discontiguous saved segments.

Features

The DCSS device driver facilitates:

- Initializing and updating ext2 compatible file system images in z/VM saved segments for use with the xip option of the ext2 file system.
- Implementing a shared read-write RAM disk for Linux guests, for example, for a file system that can be shared among multiple Linux images that run as guest systems under the same z/VM.

Starting with z/VM 5.4, you can:

- Locate a DCSS above 2047 MB
- Set up DCSS devices with a size above 2047 MB by mapping multiple DCSSs to a single DCSS block device

What you should know about DCSS

This section provides information about the DCSS device names and nodes.

Important

DCSSs occupy spool space. Be sure that you have enough spool space available (multiple times the DCSS size).

DCSS naming scheme

The standard device names are of the form dcssblk<n>, where <n> is the corresponding minor number. The first DCSS device that is added is assigned the name dcssblk0, the second dcssblk1, and so on. When a DCSS device is removed, its device name and corresponding minor number are free and can be reassigned. A DCSS device that is added always receives the lowest free minor number.

z/VM DCSS device nodes

User space programs access DCSS devices by device nodes. SUSE Linux Enterprise Server provides udev to create standard DCSS device nodes of the form /dev/<device_name>, for example:

```
/dev/dcssblk0
/dev/dcssblk1
...```

Further information

- For information on DCSS see z/VM Saved Segments Planning and Administration, SC24-6116
- For related z/VM information see CP Commands and Utilities Reference, SC24-6008.
• For an example of how the xip option for the ext2 file system and DCSS can be used see How to use Execute-in-Place Technology with Linux on z/VM on developerWorks at: [ibm.com/developerworks/linux/linux390/documentation_dev.html](ibm.com/developerworks/linux/linux390/documentation_dev.html)

### Setting up the DCSS device driver

Before you can load and use DCSSs, you must load the DCSS block device driver. Use the `segments` module parameter to load one or more DCSSs when the DCSS device driver is loaded.

#### DCSS module parameter syntax

```
insmod dcssblk segments=<dcss>[,(local)],<dcss>...
```

- `<dcss>` specifies the name of a DCSS as defined on the z/VM hypervisor. The specification for `<dcss>` is converted from ASCII to uppercase EBCDIC.
- `:` the colon (:) separates DCSSs within a set of DCSSs to be mapped to a single DCSS device. You can map a set of DCSSs to a single DCSS device if the DCSSs in the set form a contiguous memory space.
- You can specify the DCSSs in any order. The name of the first DCSS you specify is used to represent the device under `/sys/devices/dcssblk`.
- `(local)` sets the access mode to exclusive-writable after the DCSS or set of DCSSs have been loaded.
- `,` the comma (,) separates DCSS devices.

#### Examples

The following command loads the DCSS device driver and three DCSSs: DCSS1, DCSS2, and DCSS3. DCSS2 is accessed in exclusive-writable mode.

```
# modprobe dcssblk segments="dcss1,dcss2(local),dcss3"
```

The following command loads the DCSS device driver and four DCSSs: DCSS4, DCSS5, DCSS6, and DCSS7. The device driver creates two DCSS devices. One device maps to DCSS4 and the other maps to the combined storage space of DCSS5, DCSS6, and DCSS7 as a single device.

```
# modprobe dcssblk segments="dcss4,dcss5:dcss6:dcss7"
```
Avoiding overlaps with your Linux guest storage

Ensure that your DCSSs do not overlap with the memory of your z/VM guest virtual machine (guest storage). To find the start and end addresses of the DCSSs, enter the following CP command from a CMS session with privilege class E:

```
#cp q nss map
```

the output gives you the start and end addresses of all defined DCSSs in units of 4 kilobyte pages:

```
00: FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
... 00: 0011 MONDCSS CPDCSS N/A 09000 097FF SC R 00003 N/A N/A ...
```

If all DCSSs that you intend to access are located above the guest storage, you do not need to take any action.

If any DCSS that you intend to access with your guest machine overlaps with the guest storage, redefine the guest storage as two or more discontiguous storage extents such that the storage gap with the lowest address range covers all your DCSSs’ address ranges.

When defining storage gaps be aware of the following restrictions:

- You cannot place a DCSS into a storage gap other than the storage gap with the lowest address range.
- A z/VM guest that has been defined with one or more storage gaps cannot access a DCSS above the guest storage.

From a CMS session, use the DEF STORE command to define your guest storage as discontiguous storage extents. Ensure that the storage gap between the extents covers all your DCSSs’ address ranges. Issue a command of this form:

```
DEF STOR CONFIG 0.<storage_gap_begin> <storage_gap_end>.<storage above gap>
```

where:

- `<storage_gap_begin>` is the lower limit of the storage gap. This limit must be at or below the lowest address of the DCSS with the lowest address range.

  Because the lower address ranges are required for memory management functions make the lower limit at least 128 MB. The lower limit for the DCSS increases with the total memory size and 128 MB is not an exact value but it is an approximation that is sufficient for most cases.

- `<storage_gap_end>` is the upper limit of the storage gap. The upper limit must be above the upper limit of the DCSS with the highest address range.

- `<storage above gap>` is the amount of storage above the storage gap. The total guest storage is `<storage_gap_begin> + <storage above gap>`.

All values can be suffixed with M to provide the values in megabyte. Refer to CP Commands and Utilities 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.

---

**Working with the DCSS device driver**

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

- Adding a DCSS device
- Listing the DCSSs that map to a particular device
- Finding the minor number for a DCSS device
- Setting the access mode
- Saving updates to a DCSS or set of DCSSs
- Removing a DCSS device

**Adding a DCSS device**

**Before you start:**

- You need to have set up one or more DCSSs on z/VM and know the names assigned to the DCSSs on z/VM.
- If you use the watchdog device driver, turn off the watchdog before adding a DCSS device. Adding a DCSS device can result in a watchdog timeout if the watchdog is active.
- You cannot concurrently access overlapping DCSSs.
- You cannot access a DCSS that overlaps with your z/VM guest virtual storage (see "Avoiding overlaps with your Linux guest storage" on page 201).
- If a z/VM guest has been defined with multiple storage gaps, you can only add DCSSs that are located in the storage gap with the lowest address range.
- If a z/VM guest has been defined with one or more storage gaps, you cannot add a DCSS that is located above the guest storage.

To add a DCSS device enter a command of this form:

```
# echo <dcss-list> > /sys/devices/dcssblk/add
```

**<dcss-list>**

the name, as defined on z/VM, of a single DCSS or a colon (:) separated list of names of DCSSs to be mapped to a single DCSS device. You can map a set of DCSSs to a single DCSS device if the DCSSs in the set form a contiguous memory space. You can specify the DCSSs in any order. The name of the first DCSS you specify is used to represent the device under /sys/devices/dcssblk.

**Examples**

To add a DCSS called "MYDCSS" enter:

```
# echo MYDCSS > /sys/devices/dcssblk/add
```
To add three DCSSs “MYDCSS1”, “MYDCSS2”, and “MYDCSS3” as a single device enter:

```
# echo MYDCSS2:MYDCSS1:MYDCSS3 > /sys/devices/dcssblk/add
```

In sysfs, the resulting device is represented as /sys/devices/dcssblk/MYDCSS2.

### Listing the DCSSs that map to a particular device

To list the DCSSs that map to a DCSS device, issue a command like this:

```
# cat /sys/devices/dcssblk/<dcss-name>/seglist
```

where `<dcss-name>` is the DCSS name that represents the DCSS device.

#### Examples

In this example, DCSS device MYDCSS maps to a single DCSS, “MYDCSS”.

```
# cat /sys/devices/dcssblk/MYDCSS/seglist
MYDCSS
```

In this example, DCSS device MYDCSS2 maps to three DCSSs, “MYDCSS1”, “MYDCSS2”, and “MYDCSS3”.

```
# cat /sys/devices/dcssblk/MYDCSS2/seglist
MYDCSS2
MYDCSS1
MYDCSS3
```

### Finding the minor number for a DCSS device

When you add a DCSS device, a minor number is assigned to it. Unless you use dynamically created device nodes as provided by udev, you might need to know the minor device number that has been assigned to the DCSS (see “DCSS naming scheme” on page 199).

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

```
/sys/devices/dcssblk/<dcss-name>
```

where `<dcss-name>` is the DCSS name that represents the DCSS device.

This directory contains a symbolic link, block, that helps you to find out the device name and minor number. The link is of the form `../../../block/dcssblk<n>`, where `dcssblk<n>` is the device name and `<n>` is the minor number.

#### Example

To find out the minor number assigned to a DCSS device that is represented by the directory `sys/devices/dcssblk/MYDCSS` issue:

```
# readlink /sys/devices/dcssblk/MYDCSS/block
../../../block/dcssblk0
```

In the example, the assigned minor number is “0”.
Setting the access mode

You might want to access the DCSS device with write access to change the content of the DCSS or set of DCSSs that map to the device. There are two possible write access modes to the DCSS device:

**shared**

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

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

**exclusive-writable**

In the exclusive-writable mode you write to private copies of DCSSs. A private copy is writable, even if the original DCSS is read-only. Changes you make to a private copy are invisible to other guests until you save the changes (see "Saving updates to a DCSS or set of DCSSs").

After saving the changes to a DCSS, all guests that open the DCSS access the changed copy. z/VM retains a copy of the original DCSS for those guests that continue accessing it, until the last guest has stopped using it.

For either access mode the changes are volatile until they are saved (see "Saving updates to a DCSS or set of DCSSs").

Set the access mode before you open the DCSS device. To set the access mode to exclusive-writable set the DCSS device’s shared attribute to “0”. To reset the access mode to shared set the DCSS device’s shared attribute to “1”.

Issue a command of this form:

```
# echo <flag> > /sys/devices/dcssblk/<dcss-name>/shared
```

where `<dcss-name>` is the DCSS name that represents the DCSS device.

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

**Example**

To find out the current access mode of a DCSS device represented by the DCSS name “MYDCSS”:

```
# cat /sys/devices/dcssblk/MYDCSS/shared
```

“1” means that the current access mode is shared. To set the access mode to exclusive-writable issue:

```
# echo 0 > /sys/devices/dcssblk/MYDCSS/shared
```

Saving updates to a DCSS or set of DCSSs

**Before you start:**

- If you use the watchdog device driver, turn off the watchdog before saving updates to DCSSs. Saving updates to DCSSs can result in a watchdog timeout if the watchdog is active.
- Do not place save requests before you have accessed the DCSS device.
To place a request for saving changes permanently on the spool disk write “1” to the DCSS device’s save attribute. If a set of DCSSs has been mapped to the DCSS device, the save request applies to all DCSSs in the set.

Issue a command of this form:

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

where `<dcss-name>` is the DCSS name that represents the DCSS device.

Saving is delayed until you close the device.

You can check if a save request is waiting to be performed by reading the contents of the save attribute.

You can cancel a save request by writing “0” to the save attribute.

**Example**

To check if a save request exists for a DCSS device that is represented by the DCSS name “MYDCSS”:

```
# cat /sys/devices/dcssblk/MYDCSS/save
0
```

The “0” means that no save request exists. To place a save request issue:

```
# echo 1 > /sys/devices/dcssblk/MYDCSS/save
```

To purge an existing save request issue:

```
# echo 0 > /sys/devices/dcssblk/MYDCSS/save
```

**Removing a DCSS device**

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

You can remove the DCSS or set of DCSSs that are represented by a DCSS device from your Linux system by issuing a command of this form:

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

where `<dcss-name>` is the DCSS name that represents the DCSS device.

If you have created your own device nodes, you can keep the nodes for reuse. Be aware that the major number of the device might change when you unload and reload the DCSS device driver. When the major number of your device has changed, existing nodes become unusable.

**Example**

To remove a DCSS device that is represented by the DCSS name “MYDCSS” issue:

```
# echo MYDCSS > /sys/devices/dcssblk/remove
```
Changing the contents of a DCSS

The following scenario describes how you can use the DCSS block device driver to change the contents of a DCSS.

Assumptions:
- The Linux instance runs as a z/VM guest with class E user privileges.
- A DCSS has been set up and made accessible to the Linux guest.
- The DCSS does not overlap with the guest's main storage.
- There is only a single DCSS named “MYDCSS”.
- The DCSS block device driver has been set up and is ready to be used.

Note: The description in this scenario can readily be extended to changing the content of a set of DCSSs that form a contiguous memory space. The only change to the procedure would be mapping the DCSSs in the set to a single DCSS device in step 1. The assumptions about the set of DCSSs would be that the contiguous memory space formed by the set does not overlap with the guest storage and that only the DCSSs in the set are added to the Linux instance.

Perform the following steps to change the contents of a DCSS:
1. Add the DCSS to the block device driver.

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

2. Ensure that there is a device node for the DCSS block device. If it is not created for you, for example by udev, create it yourself.

   - Find out the major number used for DCSS block devices. Read /proc/devices:

   ```
   # cat /proc/devices
   ...
   Block devices
   ...
   254 dcssblk
   ...
   ```

   The major number in the example is 254.

   - Find out the minor number used for MYDCSS. If MYDCSS is the first DCSS that has been added the minor number is 0. To be sure you can read a symbolic link that is created when the DCSS is added.

   ```
   # readlink /sys/devices/dcssblk/MYDCSS/block
   ../../../block/dcssblk0
   ```

   The trailing 0 in the standard device name dcssblk0 indicates that the minor number is, indeed, 0.

   - Create the node with the `mknod` command:

   ```
   # mknod /dev/dcssblk0 b 254 0
   ```

3. Set the access mode to exclusive-write.

   ```
   # echo 0 > /sys/devices/dcssblk/MYDCSS/shared
   ```
4. Mount the file system in the DCSS on a spare mount point.
   **Example:**
   
   ```bash
   # mount /dev/dcssblk0 /mnt
   ```

5. Update the data in the DCSS.
6. Create a save request to save the changes.
   ```bash
   # echo 1 > /sys/devices/dcssblk/MYDCSS/save
   ```

7. Unmount the file system.
   ```bash
   # umount /mnt
   ```

The changes to the DCSS are now saved. When the last z/VM guest stops accessing the old version of the DCSS, the old version is discarded. Each guest that opens the DCSS accesses the updated copy.

8. Remove the device.
   ```bash
   # echo MYDCSS > /sys/devices/dcssblk/MyDCSS/remove
   ```

9. If you have created your own device node, you can optionally clean it up.
   ```bash
   # rm -f /dev/dcssblk0
   ```
Chapter 20. Shared kernel support

You can save a Linux kernel in a VM named saved system (NSS). Through an
NSS, z/VM makes operating system code in shared real memory pages available to
z/VM guest virtual machines. Multiple Linux guest operating systems on the z/VM
can then boot from the NSS and run from the single copy of the Linux kernel in
memory.

For a z/VM guest virtual machine a shared kernel in an NSS amounts to a fast boot
device. In a virtual Linux server farm with multiple z/VM guest virtual machines
sharing the NSS, the NSS can help to reduce paging and enhance performance.

What you should know about NSS

Before you create an NSS you need to have a Linux system that supports kernel
sharing installed on a conventional boot device, for example, a DASD or SCSI disk.
You create the NSS when you use a special boot parameter to boot the Linux
system from this original boot device.

Support for z/VM guests with multiple CPUs

The guest virtual machine can use multiple CPUs. For required PTFs see:
ibm.com/developerworks/linux/linux390/distribution_hints.html

Further information

For more information on NSS and the CP commands used in this section see:
• www.vm.ibm.com/linux/linuxnss.html
• z/VM CP Commands and Utilities Reference , SC24-6081 at the IBM Publications
  Center (see “Finding IBM books” on page xi).
• z/VM Virtual Machine Operation , SC24-6128-00 at the IBM Publications Center
  (see “Finding IBM books” on page xi).

Kernel parameter for creating an NSS

You create an NSS with a shared kernel by booting a Linux system with shared
kernel support with the savesys= parameter.

```
 kernel parameter syntax

 savesys=<nss_name>
```

where <nss_name> is the name you want to assign to the NSS. The name can be
one to eight characters long and must consist of alphabetic or numeric characters.
Be sure not to assign a name that matches any of the device numbers used at your
installation.

Note: If <nss_name> contains non-alphanumeric characters, the NSS might be
created successfully. However, this name might not work in CP commands.
Always use alphanumerics characters for the name.
Working with a Linux NSS

This section describes how you can create and maintain a Linux NSS. For information about booting Linux from an NSS see "Using a named saved system" on page 303. Note that Kexec is disabled for Linux guests booted from a kernel NSS.

For each task described in this section you need a z/VM guest virtual machine that:
- Runs with class E privileges
- Runs on a single virtual processor

Setting up a Linux NSS

Perform these steps to create a Linux NSS:
1. Boot Linux.
2. Insert savesys=<nssname> into the kernel parameter file used by your boot configuration, where <nssname> is the name you want to assign to the NSS. The name can be 1-8 characters long and must consist of alphabetic or numeric characters. Examples of valid names include: 73248734, NSSCSITE, or NSS1234. Be sure not to assign a name that matches any of the device numbers used at your installation.
3. Issue a zipl command to write the modified configuration to the boot device.
5. Issue an IPL command to boot Linux from the device that holds the Linux kernel. During the IPL process, the NSS is created and Linux is actually booted from the NSS.

You can now use the NSS to boot Linux in other z/VM guest virtual machines. See "Using a named saved system" on page 303 for details.

Creating a Linux NSS from the CP command line

Before you begin: On z/VM prior to 5.4.0, you require a guest with a single CPU to create a kernel NSS.

You can create a Linux NSS without booting Linux and without editing the zipl parameter file. To boot Linux and save it as an NSS issue an IPL command of this form:

```
IPL <devno> PARM savesys=<nssname>
```

where <devno> refers to a device that designates the Linux system that is to be saved as an NSS; and <nssname> is the name you want to assign to the NSS.

The NSS name can be 1-8 characters long and must consist of alphabetic or numeric characters. Examples of valid names include: 73248734, NSSCSITE, or NSS1234. Be sure not to assign a name that matches any of the device numbers used at your installation.

During the IPL process, the NSS is created and Linux is booted from the NSS.

Example: To create a Linux NSS from CP when a standard Linux system is installed on device 1234, use the following command:
Once the Linux NSS has been defined and saved, it can be booted using its name:

IPL \nxnss

For information about the PARM attribute, see "Using a VM parameter string for Linux re-IPL" on page 313.

Updating a Linux NSS

Perform these steps to update a Linux NSS:

1. Boot the updated version of your Linux system.
2. Include savesys=\(<nssname>\) into the kernel parameters used by your boot configuration, where \(<nssname>\) is the name of the NSS you want to update. See "Preparing a boot device" on page 279 for information about the boot configuration.
3. Issue a \texttt{zipl} command to write the modified configuration to the boot device.
5. Issue an IPL command to boot Linux from the device that holds the updated Linux kernel. During the IPL process, the NSS is updated and Linux is booted from the NSS.

Deleting a Linux NSS

Perform these steps to delete an obsolete Linux NSS:

1. Close down all Linux instances that use the NSS.
2. Issue a CP PURGE NSS NAME command to delete the NSS. For example, issue a command of this form

\texttt{PURGE NSS NAME \(<nssname>\)}

where \(<nssname>\) is the name of the NSS you want to delete.
Chapter 21. 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 (snip for STONITH)” on page 413).

A watchdog application that communicates directly with the z/VM control program (CP) does not require a third operating system to monitor a heartbeat. The watchdog device driver enables you to set up a restart mechanism of this form.

Features

The watchdog device driver provides:

- Access to the z/VM watchdog timer.
- An API for watchdog applications (see “External programming interfaces” on page 215).

What you should know about the watchdog device driver

The watchdog function comprises the watchdog timer that runs on z/VM and a watchdog application that runs on the Linux guest being controlled. While the Linux guest operates satisfactory, the watchdog application reports a positive status to the z/VM watchdog timer at regular intervals. The watchdog application uses a miscellaneous character device to pass these status reports to the z/VM timer (Figure 40).

The watchdog application typically derives its status by monitoring, critical network connections, file systems, and processes on the Linux guest. If a given time elapses without a positive report being received by the watchdog timer, the watchdog timer assumes that the Linux guest is in an error state. The watchdog timer then triggers a predefined action from CP against the Linux guest. Examples of possible actions are: shutting down Linux, rebooting Linux, or initiating a system dump. For information on how to set the default timer and how to perform other actions, see “External programming interfaces” on page 215.

Note: Loading or saving a DCSS can take a long time during which the virtual machine does not respond, depending on the size of the DCSS. This may cause a watchdog to timeout and restart the guest. You are advised not to use the watchdog in combination with loading or saving DCSSs.
You can find an example watchdog application at:
http://www.ibiblio.org/pub/Linux/system/daemons/watchdog/INDEX.html

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.

Module parameters

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

```
watchdog module parameter syntax

modprobe vmwatchdog [cmd=\"IPL CLEAR\"]

conceal=0

conceal=<conceal_flag>

nowayout=<nowayout_flag>
```

where:

- `<command>`
  is the command to be issued by CP if the Linux guest fails. The default “IPL” reboots the guest with the previous boot parameters.

  Instead of rebooting the same system, you could also boot from an alternate IPL device (for example, a dump device). You can also specify multiple commands to be issued, see “Examples” on page 215 for details. For more information on CP commands refer to z/VM CP Commands and Utilities Reference, SC24-6081.

  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 Commands and Utilities Reference, SC24-6081.

- `<nowayout_flag>`
  determines what happens when the watchdog device node is closed by the watchdog application.
If the flag is set to “1” (default), the z/VM watchdog timer keeps running and triggers the command specified for <command> if no positive status report is received within the given time interval. If the character "V" is written to the device and the flag is set to "0", the z/VM watchdog timer is stopped and the Linux guest continues without the watchdog support.

**Examples**
The following command loads the watchdog module and determines that, on failure, the Linux guest is to be IPLed from a device with devno 0xb1a0. The protected application environment is not enabled. The watchdog application can close the watchdog device node after writing “V” to it. As a result the watchdog timer becomes ineffective and does not IPL the guest.

```
modprobe vmwatchdog cmd="ipl b1a0" nowayout=0
```

The following example shows how to specify multiple commands to be issued.

```
echo -en "cmd1\ncmd2\ncmd3" | 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 module parameters while loading the module.

**Watchdog device node**
The watchdog application on Linux needs a misc character device to communicate with the z/VM watchdog timer. This device node is created by udev and is called /dev/watchdog.

**External programming interfaces**

This section provides information for those who want to program watchdog applications that work with the watchdog device driver.

For information on the API refer to the following files in the Linux source tree:
- /Documentation/watchdog/watchdog-api.txt
- include/linux/watchdog.h

The default watchdog timeout is 60 seconds, the minimum timeout that can be set through the IOCTL SETTIMEOUT is 15 seconds.

The following IOCTLs are supported:
- WDIOC_GETSUPPORT
- WDIOC_SETOPTIONS (WDIOS_DISABLECARD, WDIOS_ENABLECARD)
- WDIOC_GETTIMEOUT
- WDIOC_SETTIMEOUT
- WDIOC_KEEPALIVE
Chapter 22. 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:

- Through the /dev/vmcp device node
- Through a user space tool (see "vmcp - Send CP commands to the z/VM hypervisor" on page 424)

You must load the vmcp module before you can use vmcp. If your Linux guest runs under z/VM, you can configure the startup scripts to load the vmcp kernel module automatically during boot, for example, add "vmcp" to MODULES_LOADED_ON_BOOT in /etc/sysconfig/kernel.

The vmcp device driver only works under z/VM and cannot be loaded if the Linux system runs in an LPAR.

Differences between vmcp and a 3270 console

Most CP commands behave identically with vmcp and on a 3270 console. However, some commands show a different behavior:

- Diagnose X'08' (see CP Programming Services, SC24-6084) 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.

Setting up the z/VM CP interface

There are no module parameters for the vmcp device driver.

You must load the module vmcp.ko before you can work with vmcp. Because vmcp does not depend on other modules you can use insmod or modprobe to load vmcp:

```
# modprobe vmcp
```
Working with the device node

The /dev/vmcp device node is a character device node. You can use the device node directly from an application using open, write (to issue the command), read (to get the response), ioctl (to get and set status) and close. The following ioctls are supported:

Table 37. The vmcp ioctls

<table>
<thead>
<tr>
<th>Name</th>
<th>Code definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMCP_GETCODE</td>
<td>_IOR (0x10, 1, int)</td>
<td>Queries the return code of VM.</td>
</tr>
<tr>
<td>VMCP_SETBUF</td>
<td>_IOW(0x10, 2, int)</td>
<td>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).</td>
</tr>
<tr>
<td>VMCP_GETSIZE</td>
<td>_IOR(0x10, 3, int)</td>
<td>Queries the size of the response.</td>
</tr>
</tbody>
</table>
Chapter 23. AF_IUCV protocol support

The Inter-User Communication Vehicle (IUCV) is a z/VM communication facility that enables a program running in one VM guest to communicate with another z/VM guest, or with a control program, or even with itself.

The socket implementation in TCP/IP for z/VM supports two addressing families: AF_INET, and AF_IUCV. The AF_INET domain defines addressing in the Internet domain. The AF_IUCV domain defines addressing in the IUCV domain. In the IUCV domain, address spaces or virtual machines can use the socket interface to communicate with other virtual machines or address spaces within the same operating system.

The AF_IUCV protocol supports communication with z/VM back-end services. It connects socket applications operating in Linux kernels running on different z/VM user IDs, or connect a Linux application to another sockets application running in a z/VM guest.

The AF_IUCV protocol supports stream sockets only.

Features

The AF_IUCV protocol supports:

- Multiple output socket connections from a Linux guest operating system.
- Multiple input socket connections to a Linux guest operating system.
- Socket communication with applications utilizing CMS AF_IUCV support.

What you should know about the AF_IUCV support

This section provides information on IUCV devices and interfaces.

Further information

The standard definitions in the z/VM TCP/IP configuration files apply.

For more information of the z/VM TCP/IP configuration see: z/VM TCP/IP Planning and Customization, SC24-6125.

Setting up the AF_IUCV protocol support

IUCV base support must be available on the Linux instances that use the IUCV socket support. See [z/VM CP Programming Services, SC24-6084] for general IUCV setup information for z/VM guests.

There are no module parameters for the AF_IUCV protocol support. This section describes how to load the components that have been compiled as separate modules.

Enabling your z/VM guest for IUCV

To enable your z/VM guest for IUCV add the following statements to your z/VM USER DIRECT entry:

IUCV ALLOW
IUCV ANY
OPTION MAXCONN <maxno>
where `<maxno>` specifies the maximum number of IUCV connections allowed for this virtual machine. If the OPTION statement is omitted, the default number of connections is 64. The maximum is 65 535.

**Loading the IUCV modules**

You need to load the `af_iucv` module before you can make use of it. Use `modprobe` to load the `AF_IUCV` protocol support module `af_iucv`, which loads in addition to the required `iucv` module.

```bash
# modprobe af_iucv
```

**Working with the AF_IUCV protocol support**

The `AF_IUCV` support is used by modifying a socket application to specify `AF_IUCV` as the socket addressing family and `AF_IUCV` address information in the `sockaddr`. The `AF_IUCV` constant on Linux on System z is 32. The primary difference between `AF_IUCV` sockets and UNIX sockets is how partners are identified (for example, how they are named). The `sockaddr` structure for `AF_IUCV` is:

```c
struct sockaddr_iucv {
  sa_family_t siucv_family;
  unsigned short siucv_port; /* Reserved */
  unsigned int siucv_addr; /* Reserved */
  char siucv_nodeid[8]; /* Reserved */
  char siucv_userid[8]; /* Guest User Id */
  char siucv_name[8]; /* Application Name */
};
```

where:

- **siucv_family**
  - is set to `AF_IUCV` (= 32).

- **siucv_port**, **siucv_addr**, and **siucv_nodeid**
  - are reserved for future use. The `siucv_port` and `siucv_addr` fields must be zero. The `siucv_nodeid` field must be set to exactly eight blank characters.

- **siucv_userid**
  - is set to the z/VM user ID of the Linux guest running the application which owns the address. This field must be eight characters long, padded with blanks on the right.

- **siucv_name**
  - is set to the application name by which the socket is known. A server advertises a particular application name, and this is the name used by the client to connect to the server. The recommended form of the name contains eight characters, padded with blanks to the right.
Chapter 24. Cooperative memory management

The cooperative memory management (CMM, or "cmm1") is a mechanism to reduce the available memory of a Linux guest. CMM allocates pages to a dynamic page pool not available to Linux. A diagnose code indicates to z/VM that the pages in the page pool are out of use. z/VM can then immediately reuse these pages for other guests.

To set up CMM, you need to:
1. Incorporate cmm by loading the cmm module.
2. Set up a resource management tool that controls the page pool. This can be the z/VM resource monitor (VMRM) or a third party systems management tool.

This chapter describes how to set up CMM. For background information on CMM, see "Cooperative memory management background" on page 171.

You can also use the cpuplugd command to define rules for cmm behavior, see "Managing memory" on page 345.

Setting up the external resource manager is beyond the scope of this book. For more information, see the chapter on VMRM in z/VM Performance, SC24-6109.

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.

Loading the cooperative memory management module

The cooperative memory management support is compiled as a module, cmm. Use the \texttt{insmod} or \texttt{modprobe} command to load the module. Refer to the respective man pages for command details.

\begin{center}
\begin{tabular}{|c|}
\hline
cooperative memory management module parameter syntax \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{|c|}
\hline
\texttt{insmod} \texttt{cmm} \texttt{sender=VMRMSVM} \\
\texttt{modprobe} \texttt{cmm} \texttt{sender=\textless guest name\textgreater} \\
\hline
\end{tabular}
\end{center}

where \textless guest name\textgreater{} 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.

Example

To load the cooperative memory management module and allow the guest TESTID to send messages:

\begin{center}
\begin{tabular}{|c|}
\hline
\texttt{# modprobe cmm sender=TESTID} \\
\hline
\end{tabular}
\end{center}
Working with cooperative memory management

After set up, CMM works through the resource manager. No further actions are necessary. The following information is given for diagnostic purposes.

To reduce Linux guest memory size CMM allocates pages to page pools that make the pages unusable to Linux. There are two such page pools for a Linux guest, a static pool and a timed pool. You can use the /proc interface to read the sizes of the page pools.

Reading the size of the static page pool

To read the current size of the static page pool:

```
# cat /proc/sys/vm/cmm_pages
```

Reading the size of the timed page pool

To read the current size of the timed page pool:

```
# cat /proc/sys/vm/cmm_timed_pages
```
Part 5. System resources

This section describes device drivers and features that help to manage the resources of your real or virtual hardware.

**Newest version:** You can find the newest version of this book at


**Restrictions:** For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 release notes at


---

**Chapter 25. Managing CPUs** .......................... 225
CPU capability change ................................ 225
Activating standby CPUs and deactivating operating CPUs ........ 225
Examining the CPU topology ................................ 226
CPU polarization ........................................... 227

**Chapter 26. Managing hotplug memory** ................ 229
What you should know about memory hotplug .................. 229
Setting up hotplug memory ................................ 229
Performing memory management tasks ........................ 230

**Chapter 27. Large page support** ........................ 233
Setting up large page support ............................. 233
Working with large page support ........................... 233

**Chapter 28. S/390 hypervisor file system** ................. 235
Directory structure ........................................ 235
Setting up the S/390 hypervisor file system ............... 238
Working with the S/390 hypervisor file system .............. 238
Chapter 25. Managing CPUs

Some attributes that govern CPUs are available in sysfs under:
=sys/devices/system/cpu/cpu<N>

where <N> is the number of the CPU. You can read CPU capability, activate standby CPUs, and examine the CPU topology using the CPU attributes in sysfs.

CPU capability change

When the CPUs of a mainframe heat or cool, the Linux kernel generates a uevent for all affected online CPUs. You can read the CPU capability in:
=sys/devices/system/cpu/cpu<N>/capability

The capability value is an unsigned integer as defined in the system information block (SYSIB) 1.2.2 (see Architecture Principles of Operation). A lower value indicates a proportionally higher CPU capacity. Beyond that, there is no formal description of the algorithm used to generate this value. The value is used as an indication of the capability of the CPU relative to the capability of other CPU models.

Activating standby CPUs and deactivating operating CPUs

A CPU on an LPAR can be in a configured, standby, or reserved state. Under Linux, on IPL only CPUs that are in a configured state are brought online and used. The kernel operates only with configured CPUs. You can change the state of standby CPUs to configured state and vice versa.

Reserved CPUs cannot be used without manual intervention and therefore are not recognized.

Before you begin:
• Sysfs needs to be mounted to /sys.
• To put a CPU into standby state the underlying hypervisor needs to support this operation.

To configure or deconfigure a CPU its physical address needs to be known. Since the sysfs interface is used to configure a CPU by its sysfs entry this requires a static mapping of physical to logical CPU numbers. The physical address of a CPU can be found in the address attribute of a logical CPU:

# cat /sys/devices/system/cpu/cpu<N>/address

For example:

# cat /sys/devices/system/cpu/cpu0/address
0

To activate a standby CPU:
1. Only present CPUs have a sysfs entry. If you add a CPU to the system the kernel automatically detects it. You can force the detection of a CPU using the rescan attribute. To rescan, write any string to the rescan attribute, for example:
When new CPUs are found new sysfs entries are created and they are in the configured or standby state depending on how the hypervisor added them.

2. Change the state of the CPU to configured by writing "1" to its configure attribute:

```
echo 1 > /sys/devices/system/cpu/cpu<X>/configure
```

where <X> is any CPU in standby state.

3. Bring the CPU online by writing "1" to its online attribute:

```
echo 1 > /sys/devices/system/cpu/cpu<X>/online
```

To deactivate an operating CPU:

1. Bring the CPU offline by writing "0" to its online attribute:

```
echo 0 > /sys/devices/system/cpu/cpu<X>/online
```

2. Change the state of the CPU to standby by writing "0" to its configure attribute:

```
echo 0 > /sys/devices/system/cpu/cpu<X>/configure
```

---

## Examining the CPU topology

If supported by your hardware, an interface is available that you can use to get information about the CPU topology of an LPAR. Use this, for example, to optimize the Linux scheduler, which bases its decisions on which process gets scheduled to which CPU. Depending on the workload, this might increase cache hits and therefore overall performance.

**Note:** By default CPU topology support is disabled in the Linux kernel. If it is advantageous to your workload, enable it by specifying the kernel parameter `topology=on` in your parmfile or zipl.conf.

**Before you begin:**

- The sysfs needs to be mounted to `/sys`.

The common code attribute `core_siblings` will be visible for all online CPUs:

```
/sys/devices/system/cpu/cpu<N>/topology/core_siblings
```

It contains a CPU mask that tells you which CPUs (including the current one) are close to each other. If a machine reconfiguration causes the CPU topology to change, then change uevents will be created for each online CPU.

**Note that when the kernel also supports standby CPU activation/deactivation** (see "Activating standby CPUs and deactivating operating CPUs" on page 225) then the `core_siblings` CPU mask also contains the CPUs that are in a configured, but offline state. Updating the mask after a reconfiguration might take up to a minute.
**CPU polarization**

You can optimize the operation of a vertical SMP environment by adjusting the SMP factor based on the workload demands. During peak workloads the operating system may operate on a large n-way, with all CPUs busy, whereas at other times it may fall back to a single processor. This limits the performance effects of context switches, TLB flushes, cache poisoning, as well as dispatcher workload balancing and the like, by delivering better processor affinity for particular workloads.

**Before you begin:**
- The sysfs needs to be mounted to /sys.

Horizontal CPU polarization means that the underlying hypervisor will dispatch each of the guests' virtual CPUs for the same amount of time.

If vertical CPU polarization is active then the hypervisor will dispatch certain CPUs for a longer time than others for maximum performance. For example, if a guest has three virtual CPUs, each of them with a share of 33%, then in case of vertical CPU polarization all of the processing time would be combined to a single CPU which would run all the time, while the other two CPUs would get nearly no CPU time.

There are three types of vertical CPUs: high, medium and low. Low CPUs hardly get any real CPU time, while high CPUs get a full real CPU. Medium CPUs get something in between.

**Note:** Running a system with different types of vertical CPUs may result in significant performance regressions. If possible, use only one type of vertical CPUs. Set all other CPUs offline and deconfigure them.

Use the dispatching attribute to switch between horizontal and vertical CPU polarization. To switch between the two modes write a 0 for horizontal polarization (the default) or a 1 for vertical polarization to the dispatching attribute.

```
/sys/devices/system/cpu/dispatching
```

The polarization of each CPU can be seen from the polarization attribute of each CPU:

```
/sys/devices/system/cpu/cpu<N>/polarization
```

Its contents is one of:
- horizontal - each of the guests' virtual CPUs is dispatched for the same amount of time.
- vertical:high - full CPU time is allocated.
- vertical:medium - medium CPU time is allocated.
- vertical:low - very little CPU time is allocated.
- unknown

When switching polarization the polarization attribute might contain the value unknown until the configuration change is done and the kernel has figured out the new polarization of each CPU.
Chapter 26. Managing hotplug memory

You can dynamically increase or decrease the memory for your running Linux system. To make memory available as hotplug memory you must define it to your LPAR or z/VM.

What you should know about memory hotplug

This section explains how hotplug memory is represented in sysfs.

How memory is represented in sysfs

In sysfs, both the core memory of a Linux instance and the available hotplug memory are represented in form of memory sections of equal size. Each section is represented as a directory of the form /sys/devices/system/memory/memory<n>, where <n> is an integer. You can find out the section size by reading the /sys/devices/system/memory/block_size_bytes attribute.

In the naming scheme, the memory sections with the lowest address ranges are assigned the lowest integer numbers. Accordingly, the core memory begins with memory0. The hotplug memory sections follow the core memory sections.

You can infer where the hotplug memory begins by calculating the number of core memory sections from the size of the base memory and the section size. For example, for a core memory of 512 MB and a section size of 128 MB, the core memory is represented by 4 sections, memory0 through memory3. In this example, the first hotplug memory section is memory4. Another Linux instance with a core memory of 1024 MB and access to the same hotplug memory, represents this first hotplug memory section as memory8.

The hotplug memory is available to all operating system instances within the z/VM system or LPARs to which it has been defined. The state sysfs attribute of a memory section indicates whether the section is in use by your own Linux system. The state attribute does not indicate whether a section is in use by another operating system instance. Attempts to add memory sections that are already in use fail.

Further information

For more information on memory hotplug, see /Documentation/memory-hotplug.txt in the Linux source tree.

Setting up hotplug memory

Before you can use hotplug memory on your Linux instance, you must define this memory as hotplug memory on your physical or virtual hardware.

Defining hotplug memory to an LPAR

You use the hardware management console (HMC) to define hotplug memory as reserved storage on an LPAR.

For information about defining reserved storage for your LPAR see the Processor Resource/Systems Manager™ Planning Guide for your mainframe.
Defining hotplug memory to z/VM

In z/VM, you define hotplug memory as **standby storage**. z/VM supports standby storage as of version 5.4. There is also **reserved storage** in z/VM, but other than reserved memory defined for an LPAR, reserved storage defined in z/VM is not available as hotplug memory.

For information about defining standby memory for z/VM guest operating systems see the “DEFINE STORAGE” section in *z/VM CP Commands and Utilities Reference*.

Making all hotplug memory removable

When using hotplug memory, Linux consumes memory for hotplug memory management. By default, the corresponding data structures are located in the hotplug memory section with the lowest address range. Because of these data structures, the lowest hotplug memory section cannot be released from a running Linux instance.

You can use the `kernelcore=` kernel parameter to make Linux maintain the hotplug memory management structures within the core memory. To make the lowest hotplug memory segment removable from a running Linux instance, specify a value that is approximately 10% below the upper limit of the core memory.

You can specify the value in byte or use a K, M, or G suffix to specify the value in kilobyte, megabyte, or gigabyte.

**Example:** If your core memory is 512 MB, specify `kernelcore=450M`.

For more information about kernel parameters, see `/Documentation/kernel-parameters.txt` in the Linux source tree.

Performing memory management tasks

This section describes typical memory management tasks.

- Finding out the memory section size
- Displaying the available memory sections
- Adding memory
- Removing memory

Finding out the memory section size

You can find out the size of your memory sections by reading `/sys/devices/system/memory/block_size_bytes`. This sysfs attribute contains the section size in byte in hexadecimal notation.

**Example:**

```bash
# cat /sys/devices/system/memory/block_size_bytes
8000000
```

This hexadecimal value corresponds to 128 MB.
Displaying the available memory sections

You can find out if a memory section is online or offline by reading its state attribute. The following example shows how you can get an overview of all available memory sections:

```
# grep -r --include="state" "line" /sys/devices/system/memory/
/sys/devices/system/memory/memory0/state:online
/sys/devices/system/memory/memory1/state:online
/sys/devices/system/memory/memory2/state:online
/sys/devices/system/memory/memory3/state:online
/sys/devices/system/memory/memory4/state:offline
/sys/devices/system/memory/memory5/state:offline
/sys/devices/system/memory/memory6/state:offline
/sys/devices/system/memory/memory7/state:offline
```

Your core memory is always online and cannot be set offline. The offline sections represent hotplug memory that has not been added to your Linux instance. An offline section can be free to be added to your Linux instance but it might also be in use by another Linux instance.

Adding memory

You add a hotplug memory section by writing `online` to its sysfs state attribute.

**Example:** Enter the following command to add a memory section memory5:

```
# echo online > /sys/devices/system/memory/memory5/state
```

Adding the memory section fails, if the memory section is already in use. The state attribute changes to online when the memory section has been added successfully.

Removing memory

You remove a hotplug memory section by writing `offline` to its sysfs state attribute.

**Example:** Enter the following command to remove a memory section memory5:

```
# echo offline > /sys/devices/system/memory/memory5/state
```

The hotplug memory functions first relocate memory pages to free the memory section and then remove it. The state attribute changes to offline when the memory section has been removed successfully.

The memory section is not removed if it is within the core memory or if it cannot be freed completely. You must take special setup steps to enable the hotplug memory functions to completely free the hotplug memory section with the lowest address range (see “Making all hotplug memory removable” on page 230).
Chapter 27. Large page support

Large page support entails support for the Linux hugetlbfs file system. This virtual file system is backed by larger memory pages than the usual 4 K pages; for System z the hardware page size is 1 MB. In SUSE Linux Enterprise Server 11 the page size is also 1 MB, in contrast to SUSE Linux Enterprise Server 10, which uses a page size of 2 MB.

Applications using large page memory will save a considerable amount of page table memory. Another benefit from the support might be an acceleration in the address translation and overall memory access speed.

Setting up large page support

This section describes the parameters that you can use to configure large page support.

Kernel parameters

This section describes how to configure large page support. You configure large page support by adding parameters to the kernel parameter line.

Large page support kernel parameter syntax

```
SM590000
hugepages = <number>
SM590000
```

where:

- **number**
  - is the number of large pages to be allocated at boot time.

**Note:** If you specify more pages than available, Linux will reserve as many as possible. This will most probably leave too few general pages for the boot process and might stop your system with an out-of-memory error.

Working with large page support

This section describes typical tasks that you need to perform when working with large page support.

**Before you begin:** You must be root to write to the /proc file system.

- The "hugepages=" kernel parameter should be specified with the number of large pages to be allocated at boot time. To read the current number of large pages, issue:
  ```
  cat /proc/sys/vm/nr_hugepages
  ```
- To change the number of large pages dynamically during run-time, write to the /proc file system:
  ```
  echo 12 > /proc/sys/vm/nr_hugepages
  ```
If there is not enough contiguous memory available to fulfill the request, the maximum number of large pages will be reserved.

- To obtain information about amount of large pages currently available and the large page size, issue:

```
    cat /proc/meminfo
    ...
    HugePages_Total: 20
    HugePages_Free: 14
    Hugepagesize: 2048 KB
    ...
```

- To see if hardware large page support is enabled (indicated by the word "edat" in the "features" line), issue:

```
    cat /proc/cpuinfo
    ...
    features : esan3 zarch stfle msa ldisp eimm dfp edat
    ...
```

The large page memory can be used through mmap() or SYSv shared memory system calls, more detailed information can be found in the Linux kernel source tree under Documentation/vm/hugetlbpage.txt, including implementation examples.
Chapter 28. S/390 hypervisor file system

The S/390® hypervisor file system provides a mechanism to access LPAR and z/VM hypervisor data.

The S/390 hypervisor file system requires that the z/VM version supports DIAG 2fc.

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 41 illustrates the file system tree that is created for LPAR.

![Diagram of the hypervisor file system for LPAR]

LPAR directories and attributes

The directories and attributes have the following meaning for LPARs:

**update**
Write only file to trigger an update of all attributes.

**cpus/** Directory for all physical CPUs.

**cpus/<cpu ID>**
Directory for one physical CPU. `<cpu ID>` is the logical (decimal) CPU number.
**type**  Type name of physical CPU, such as CP or IFL.

**mgmtime**  
Physical-LPAR-management time in microseconds (LPAR overhead).

**hyp/**  Directory for hypervisor information.

**hyp/type**  
Type of hypervisor (LPAR hypervisor).

**systems/**  
Directory for all LPARs.

**systems/<lpar name>/**  
Directory for one LPAR.

**systems/<lpar name>/cpus/<cpu ID>/**  
Directory for the virtual CPUs for one LPAR. The `<cpu ID>` is the logical (decimal) cpu number.

**type**  
Type of the logical CPU, such as CP or IFL.

**mgmtime**  
LPAR-management time. Accumulated number of microseconds during which a physical CPU was assigned to the logical cpu and the cpu time was consumed by the hypervisor and was not provided to the LPAR (LPAR overhead).

**cputime**  
Accumulated number of microseconds during which a physical CPU was assigned to the logical cpu and the cpu time was consumed by the LPAR.

**onlinetime**  
Accumulated number of microseconds during which the logical CPU has been online.

**Note:**  For older machines the `onlinetime` attribute might be missing. In general, user space applications should be prepared that attributes are missing or new attributes are added to the file system. To check the content of the files you can use tools such as `cat` or `less`.

**z/VM directories and attributes**

The directories and attributes have the following meaning for z/VM guests:

**update**  
Write only file to trigger an update of all attributes.

**cpus/**  
Directory for all physical CPUs.

**cpus/count**  
Total current CPUs.

**hyp/**  
Directory for hypervisor information.

**hyp/type**  
Type of hypervisor (z/VM hypervisor).

**systems/**  
Directory for all z/VM guests.

**systems/<guest name>/**  
Directory for one guest.
**systems/*<guest name>/onlinetime_us**
Time in microseconds that the guest has been logged on.

**systems/*<guest name>/cpus/**
Directory for the virtual CPUs for one guest.

- **capped**
  Flag that shows whether CPU capping is on for guest (0 = off, 1 = soft, 2 = hard).

- **count**
  Total current virtual CPUs in the guest.

- **cputime_us**
  Number of microseconds where the guest virtual CPU was running on a physical CPU.

- **dedicated**
  Flag that shows if the guest has at least one dedicated CPU (0 = no, 1 = yes).

- **weight_cur**
  Current share of guest (1-10000); 0 for ABSOLUTE SHARE guests.

- **weight_max**
  Maximum share of guest (1-10000); 0 for ABSOLUTE SHARE guests.

- **weight_min**
  Minimum share of guest (1-10000); 0 for ABSOLUTE SHARE guests.

**systems/*<guest name>/samples/**
Directory for sample information for one guest.

- **cpu_delay**
  Number of CPU delay samples attributed to the guest.

- **cpu_using**
  Number of CPU using samples attributed to the guest.

- **idle**
  Number of idle samples attributed to the guest.

- **mem_delay**
  Number of memory delay samples attributed to the guest.

- **other**
  Number of other samples attributed to the guest.

- **total**
  Number of total samples attributed to the guest.

**systems/*<guest name>/mem/**
Directory for memory information for one guest.

- **max_KiB**
  Maximum memory in KiB (1024 bytes).

- **min_KiB**
  Minimum memory in KiB (1024 bytes).

- **share_KiB**
  Guest estimated core working set size in KiB (1024 bytes).

- **used_KiB**
  Resident memory in KiB (1024 bytes).

To check the content of the files you can use tools such as `cat` or `less`. 

---

Chapter 28. S/390 hypervisor file system 237
Setting up the S/390 hypervisor file system

In order to use the file system, it has to be mounted. You can do this either manually with the mount command or with an entry in /etc/fstab.

To mount the file system manually issue the following command:

```
# mount none -t s390_hypfs <mount point>
```

where `<mount point>` is where you want the file system mounted. Preferably, use /sys/hypervisor/s390.

If you want to put hypfs into your /etc/fstab you can add the following line:

```
none <mount point> s390_hypfs defaults 0 0
```

Note that if your z/VM system does not support DIAG 2fc, the s390_hypfs will not be activated and it is not possible to mount the file system. You will see an error message like the following:

```
mount: unknown filesystem type 's390_hypfs'
```

To get data for all z/VM guests, privilege class B is required for the guest, where hypfs is mounted. For non-class B guests, only data for the local guest is provided.

Working with the S/390 hypervisor file system

This section describes typical tasks that you need to perform when working with the S/390 hypervisor file system.

- Defining access rights
- Updating hypfs information

Defining access rights

If no mount options are specified, the files and directories of the file system get the uid and gid of the user who mounted the file system (normally root). It is possible to explicitly define uid and gid using the mount options `uid=<number>` and `gid=<number>`.

Example: You can define uid=1000 and gid=2000 with the following mount command:

```
# mount none -t s390_hypfs -o "uid=1000,gid=2000" <mount point>
```

Alternatively, you can add the following line to the /etc/fstab file:

```
none <mount point> s390_hypfs uid=1000,gid=2000 0 0
```

The first mount defines uid and gid. Subsequent mounts automatically have the same uid and gid setting as the first one.

The permissions for directories and files are as follows:

- Update file: 0220 (--w--w----)
- Regular files: 0440 (-r--r-----)
Updating hypfs information

You trigger the update process by writing something into the update file at the top level hypfs directory. For example, you can do this by writing the following:

```
echo 1 > update
```

During the update the whole directory structure is deleted and rebuilt. If a file was open before the update, subsequent reads will return the old data until the file is opened again. Within one second only one update can be done. If within one second more than one update is triggered, only the first one is done and the subsequent write system calls return -1 and errno is set to EBUSY.

If an application wants to ensure consistent data, the following should be done:
1. Read modification time through `stat(2)` from the update attribute.
2. If data is too old, write to the update attribute and go to 1.
3. Read data from file system.
4. Read modification time of the update attribute again and compare it with first timestamp. If the timestamps do not match then go to 2.
Part 6. Security

This part describes device drivers and features that support security aspects of SUSE Linux Enterprise Server 11 for System z.

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

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 release notes at
www.novell.com/linux/releasenotes/s390x/SUSE-SLES/11/

Chapter 29. Generic cryptographic device driver  ........................................ 243
Features ........................................................................................................... 243
Elements of z90crypt ..................................................................................... 243
Setting up the z90crypt device driver .......................................................... 246
Working with the z90crypt device driver ...................................................... 248
External programming interfaces ................................................................. 251

Chapter 30. Pseudo-random number device driver  .................................. 253
What you should know about the pseudo-random number device driver .... 253
Setting up the pseudo-random number device driver ................................. 253
Reading pseudo-random numbers ............................................................... 253

Chapter 31. Data execution protection for user processes ..................... 255
Features ........................................................................................................... 255
What you should know about the data execution protection feature ........ 255
Setting up the data execution protection feature ....................................... 255
Working with the data execution protection feature ................................. 255
Chapter 29. 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 features. The generic cryptographic device driver (z90crypt) is required when one or more of these devices is available in the hardware.

Features

The cryptographic device driver supports a range of hardware and software functions:

Supported devices

The coprocessors supported are:
- Crypto Express2 Coprocessor (CEX2C)
- Crypto Express2 Accelerator (CEX2A)

Note: When Linux is running under VM and an accelerator card (CEX2A) is present any cryptographic coprocessor cards will be hidden. For VM guests all cards are supported as of z/VM 5.2 or later.

Supported facilities

The cryptographic device driver supports these cryptographic operations:
- Clear key encryption and decryption using the Rivest-Shamir-Adleman (RSA) exponentiation operation using either a modulus-exponent (Mod-Expo) or Chinese-Remainder Theorem (CRT) key.
- Secure key encryption and decryption - see Secure Key Solution with the Common Cryptographic Architecture Application Programmer’s Guide, SC33-8294
- Generation of long random numbers, see “Generating and accessing long random numbers” on page 249.

Elements of z90crypt

This section provides information on the software you need to use z90crypt and the use it makes of cryptographic hardware.

Software components

To run programs that use the z90crypt device driver for clear key encryption, you need:
- the device driver module z90crypt
- the libica library, unless applications call the device driver directly
  You can use the libica library for generation of RSA key pairs, symmetric and asymmetric encryption, and message hashing.
- the openCryptoki library if applications use the PKCS #11 API.

To run programs that use the z90crypt device driver for secure key encryption, you need:
- the device driver module
- the CCA library
Applications can use clear key cryptography with the libica library through the openCryptoki PKCS #11 API or the openssl library with the ibmca plug-in. For secure key cryptography the CCA library must be used. Figure 42 illustrates the software relationships:

In Figure 42 “Application A” is an application that uses the secure key interface, “Application B” uses the openssl library (here you need to install the RPM openssl -ibmca), and “Application C” uses the PKCS #11 API.

Refer to Security on z/VM, SG24-7471 for setup information for the CCA, openssl, and openCryptoki libraries.

See “Setting up the z90crypt device driver” on page 246 for information on how to set up the cryptographic device driver.

**CP Assist for Cryptographic Function (CPACF)**

The libica library includes CPACF instructions that allow applications to use hardware-accelerated cryptography. The following functions are included in libica 1.3.9:

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Supported on System z9</th>
<th>Supported on System z10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>icaDesEncrypt, icaDesDecrypt</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TDES</td>
<td>icaTDesEncrypt, icaTDesDecrypt</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHA-1</td>
<td>icaSha1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHA-224</td>
<td>icaSha224</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>SHA-256</td>
<td>icaSha256</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SHA-384</td>
<td>icaSha384</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>SHA-512</td>
<td>icaSha512</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>
Table 38. CPACF functions included in libica 1.3.9 (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Supported on System z9</th>
<th>Supported on System z10</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES with 128 bit keys</td>
<td>icaAesEncrypt, icaAesDecrypt</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AES with 192 bit keys</td>
<td>icaAesEncrypt, icaAesDecrypt</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>AES with 256 bit keys</td>
<td>icaAesEncrypt, icaAesDecrypt</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Pseudo Random Number Generation</td>
<td>icaRandomNumberGenerate</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

There is a software fallback provided within libica for CPACF functions (see Table 38 on page 244) that are not supported on your hardware.

The function prototypes are provided in the header file, ica_api.h. Applications using these functions must link libica and libcrypto. The libcrypto library is available from the OpenSSL package.

To ascertain what functions are available on your system, use the icainfo command, for example:

```
# icainfo
The following CP Assist for Cryptographic Function (CPACF) operations are supported by libica on this system:
SHA-1: yes
SHA-256: yes
SHA-512: yes
DES: yes
TDES-128: yes
TDES-192: yes
AES-128: yes
AES-192: yes
AES-256: yes
PRNG: yes
```

Prerequisites

If you are running Linux under VM then z/VM 5.2 or later is required.

For the secure key cryptographic functions you must use the CCA library. This can be downloaded from the IBM cryptographic coprocessor web page at http://www.ibm.com/security/cryptocards/

Performance considerations

Load balancing
To maximize performance the driver uses a load balancing algorithm to distribute requests across all available AP bus devices. The algorithm uses a list holding all AP bus devices sorted by increasing utilization. A new request will be submitted to the AP bus device with the lowest utilization. The increased load will move this device further towards the end of the device list after a re-sort. 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 z90crypt device driver assigns work to cryptographic devices according to device type in the following order of preference:
1. CEX2A
2. CEX2C

Setting up for the 31-bit compatibility mode
31-bit applications can access the 64-bit z90crypt driver by using the 31-bit compatibility mode.

Further information
- For information on how to setup your cryptographic environment on Linux under z/VM refer to [Security on z/VM, SG24-7471](#).
- 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 Commands and Utilities Reference, SG24-6008](#).
- For secure key cryptography refer to [Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide, SC33-8294](#).

Setting up the z90crypt device driver
This section describes the z90crypt kernel parameters and the z90crypt module, and how to install additional components required by the device driver.

This section also describes the z90crypt device node. 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.

Monolithic module parameters
This section describes how to load and configure the z90crypt device driver independently of YaST. For alternative methods of starting and stopping z90crypt in SUSE Linux Enterprise Server, see "Working with the z90crypt device driver" on page 248. To make any configuration changes persistent across IPLs, use YaST.

z90crypt module syntax

```
modprobe z90crypt
insmod
```

where

<domain>

is an integer in the range from 0 to 15 that identifies the cryptographic domain for the Linux instance.

The default ("domain=-1") causes the device driver to attempt to autodetect and use the domain index with the maximum number of devices.

You need to specify the domain parameter only if you are running Linux in an LPAR for which multiple cryptographic domains have been defined.
<poll_thread>
is an integer argument and enables a polling thread to increase cryptographic performance. Valid values are 1 (enabled) or 0 (disabled, this is the default).

The z90crypt driver can run with or without polling thread. When running with polling thread one CPU with no outstanding workload is constantly polling the cryptographic cards for finished cryptographic requests. The polling thread will sleep when no cryptographic requests are being processed. This mode uses the cryptographic cards as much as possible at the cost of blocking one CPU during cryptographic operations.

Without polling thread the cryptographic cards are polled at a much lower rate, resulting in higher latency and reduced throughput for cryptographic requests but without a notable CPU load.

Refer to the respective man page for details on modprobe or insmod.

Examples
• This example loads the z90crypt device driver module if Linux runs in an LPAR with only one cryptographic domain:
  ```
  # modprobe z90crypt
  ```

• This example loads the z90crypt device driver module and makes z90crypt operate within the cryptographic domain “1”:
  ```
  # modprobe z90crypt domain=1
  ```

The libica library
The libica RPMs are included with SUSE Linux Enterprise Server, and you can install them using YaST.

The openCryptoki library
The openCryptoki RPMs are included with SUSE Linux Enterprise Server, and you can install them using YaST.

Note: To be able to configure openCryptoki (with pkcsconf) user root must be a member of group pkcs11.

The CCA library
You can obtain the CCA library from the IBM Cryptographic Hardware Web site at:
http://www.ibm.com/security/cryptocards

The library is available from the software download page for the PCI-X Cryptographic Coprocessor. Install the RPM and see the readme file at /opt/IBM/4764/doc/README.linz. The readme explains where files are located, what users are defined, and how to proceed.

See also appendix H in Secure Key Solution with the Common Cryptographic Architecture Application Programmer’s Guide, SC33-8294 for additional installation instructions.
**z90crypt device node**

User space programs address cryptographic devices through a single device node. In SUSE Linux Enterprise Server udev creates the device node /dev/z90crypt for you. The device node z90crypt is assigned to the miscellaneous devices.

---

### Working with the z90crypt device driver

Typically, cryptographic devices are not directly accessed by users but through user programs. Some tasks can be performed through the sysfs interface. This section describes the following tasks:

- "Starting z90crypt"
- "Setting devices online or offline"
- "Setting the polling thread" on page 249
- "Using the high resolution polling timer" on page 249
- "Generating and accessing long random numbers" on page 249
- "Dynamically adding and removing cryptographic adapters" on page 250
- "Displaying z90crypt information" on page 250
- "Stopping z90crypt" on page 251

### Starting z90crypt

In SUSE Linux Enterprise Server you start the z90crypt device driver using the command:

```bash
# service z90crypt start
```

or using the start script:

```bash
# rcz90crypt start
```

These commands loads the z90crypt device driver module if Linux runs in an LPAR with only one cryptographic domain.

### Setting devices online or offline

Use the sysfs attribute online to set devices online or offline by writing 1 or 0 to it, respectively.

#### Examples

- To set a cryptographic device with bus device 0x3e online issue:
  ```bash
  echo 1 > /sysfs/bus/ap/devices/card3e/online
  ```

- To set a cryptographic device with bus device 0x3e offline issue:
  ```bash
  echo 0 > /sysfs/bus/ap/devices/card3e/online
  ```

- To check the online status of the cryptographic device with bus ID 0x3e issue:
  ```bash
  cat /sysfs/bus/ap/devices/card3e/online
  ```

The value is '1' if the device is online and '0' otherwise.
Setting the polling thread

To increase cryptographic performance use the poll_thread attribute. If Linux is running as a guest on z/VM, the poll_thread attribute is disabled by default.

The z90crypt 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.

Examples

- To activate a polling thread for a device 0x3e issue:
  ```
  echo 1 > /sysfs/bus/ap/devices/card3e/poll_thread
  ```

- To deactivate a polling thread for a cryptographic device with bus device 0x3e issue:
  ```
  echo 0 > /sysfs/bus/ap/devices/card3e/poll_thread
  ```

Using the high resolution polling timer

If you are running SUSE Linux Enterprise Server in an LPAR or z/VM, a high resolution timer is used instead of the standard timer. The high resolution timer enables polling at nanosecond intervals rather than the 100 Hz intervals used by the standard timer.

You can set the polling time by using the sysfs attribute poll_timeout. The read-write attribute can be found at the AP bus level.

Example

To read the poll_timeout attribute for the ap bus issue:
  ```
  # cat /sysfs/bus/ap/poll_timeout
  ```

To set the poll_timeout attribute for the ap bus to poll, for example, every microsecond, issue:
  ```
  # echo 1000 > /sysfs/bus/ap/poll_timeout
  ```

Generating and accessing long random numbers

The support of long random numbers enables user-space applications to access large amounts of random number data through a character device.

Before you begin:

- At least one CEX2C feature must be installed in the system and be configured as coprocessor. The CCA library on the CEX2C card must be at least version 3.30.
- Under z/VM at least one CEX2C card must be configured as DEDICATED to the z/VM guest.
- Automatic creation of the random number character device requires udev.
The cryptographic device driver z90crypt must be loaded.

If z90crypt detects at least one CEX2C card capable of generating long random numbers, a new miscellaneous character device is registered and can be found under /proc/misc as hw_random. The default rules provided with udev creates a character device node called /dev/hwrng and a symbolic link /dev/hw_random pointing to /dev/hwrng.

Reading from the character device or the symbolic link returns the hardware generated long random numbers. However, do not read excess amounts of random number data from this character device as the data rate is limited due to the cryptographic hardware architecture.

Removing the last available CEX2C card while z90crypt is loaded automatically removes the random number character device. Reading from the random number character device while all CEX2C cards are set offline results in an input/output error (EIO). After at least one CEX2C card is set online again reading from the random number character device continues to return random number data.

**Dynamically adding and removing cryptographic adapters**

On an LPAR, you can add or remove cryptographic adapters without the need to reactivate the LPAR after a configuration change.

Linux attempts to detect new cryptographic adapters and set them online every time a configuration timer expires. Read or modify the expiration time through the sysfs attribute /sysfs/bus/ap/config_time.

Adding or removing of cryptographic adapters to or from an LPAR is transparent to applications using clear key functions. If a cryptographic adapter is removed while cryptographic requests are being processed, z90crypt automatically re-submits lost requests to the remaining adapters. Special handling is required for secure key.

Secure key requests are usually submitted to a dedicated cryptographic coprocessor. If this coprocessor is removed, lost or new requests cannot be submitted to a different coprocessor. Therefore dynamically adding and removing adapters with a secure key application requires support within the application.

**Displaying z90crypt information**

Each cryptographic adapter is represented in sysfs directory of the form

```
/sysfs/bus/ap/devices/card<XX>
```

where <XX> is the device index for each device. The valid device index range is hex 00 to hex 3f. For example device 0x1a can be found under /sysfs/bus/ap/devices/card1a. The sysfs directory contains a number of attributes with information about the cryptographic adapter.

**Table 39. Cryptographic adapter attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>depth</td>
<td>Read-only attribute representing the input queue length for this device.</td>
</tr>
<tr>
<td>hwtype</td>
<td>Read-only attribute representing the hardware type for this device. The following values are defined:</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>
Table 39. Cryptographic adapter attributes (continued)

<table>
<thead>
<tr>
<th>modalias</th>
<th>Read-only attribute representing an internally used device bus-ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_count</td>
<td>Read-only attribute representing the number of requests already processed by this device.</td>
</tr>
<tr>
<td>type</td>
<td>Read-only attribute representing the type of this device. The following types are defined:</td>
</tr>
<tr>
<td></td>
<td>• CEX2C</td>
</tr>
<tr>
<td></td>
<td>• CEX2A</td>
</tr>
</tbody>
</table>

**Stopping z90crypt**

To stop z90crypt device driver, issue the command::

```
# service z90crypt stop
```

or use the script:

```
# rcz90crypt stop
```

**External programming interfaces**

This section provides information for those who want to program against the different libraries.

For information on the API refer to the following files in the Linux source tree:

- The cryptographic device driver header file /usr/include/asm-s390/zcrypt.h
- The libica library /usr/include/ica_api.h
- The openCryptoki library /usr/include/opencryptoki/pkcs11.h
Chapter 30. Pseudo-random number device driver

The pseudo-random number device driver is a character device driver that provides user-space applications with pseudo-random numbers generated by the pseudo-random number generator of the System z CP Assist for Cryptographic Function (CPACF).

What you should know about the pseudo-random number device driver

The pseudo-random number device provides pseudo-random numbers similar to the Linux pseudo-random number device /dev/urandom but provides a better performance.

Setting up the pseudo-random number device driver

There are no module parameters for the pseudo-random number device driver.

You must load the pseudo-random number module before you can work with it. Use insmod or modprobe to load the module:

```
# modprobe prng
```

Device node

User-space programs access the pseudo-random-number device through a device node, /dev/prandom. SUSE Linux Enterprise Server provides udev to create it for you.

The /dev/prandom device node is a character device node (major number 10) with a dynamic minor number. During load, a sysfs folder called class/misc/prandom/ is created, which contains the dev file for getting the major and minor number of the pseudo-random number device.

Making the device node accessible to non-root users

By default, only user root can read from the pseudo-random number device.

If access to the device is restricted to root on your system, add the following udev rule to automatically extend access to the device to other users.

```
KERNEL="prandom", MODE="0444", OPTIONS="last_rule"
```

Reading pseudo-random numbers

The pseudo-random number device is read-only. You can obtain random numbers by using any of these function:

- `read (/dev/prandom, buffer, bytes)`
- `cat`
- `dd`

**Example:** In this example `bs` specifies the block size in bytes for transfer, and `count` the number of records with block size. The bytes are written to the output file.
dd if=/dev/prandom of=<output file name> bs=<xxxx> count=<nnnn>
Chapter 31. Data execution protection for user processes

The data execution protection feature, similarly to the NX feature on other architectures, provides data execution protection for user processes. The data execution protection prevents, for example, stack overflow exploits and generally makes a system insensitive to buffer-overflow attacks in user space. Using this feature you can switch the addressing modes of kernel and user space. The switch of the addressing modes is a prerequisite to enable the execute protection.

Features

The data execution protection feature provides the following functions:

- Switch the kernel/user space addressing modes
- Data execution protection for user processes

What you should know about the data execution protection feature

This feature is implemented in software, with some hardware support on IBM System z9-109 EC and BC hardware. The hardware support is an instruction that allows copying data between arbitrary address spaces. Without this hardware support, a manual page-table walk is used for kernel-user-copy functions. This has a negative performance impact if you enable the feature through kernel parameter. Selecting the config options will not have this negative effect.

Setting up the data execution protection feature

To enable the data execution protection, 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"):

...  
Linux is running as a z/VM guest operating system in 64-bit mode  
Execute protection active, mvcos available  
Detected 4 CPUs  
...

To enable only the addressing mode switch, 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"):

...  
Linux is running as a z/VM guest operating system in 64-bit mode  
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):

```bash
$ execstack -s /usr/bin/find
$ execstack -q /usr/bin/find
```

---

**Example**

Clear and query the executable stack flag (stack is not executable):

```bash
$ execstack -c /usr/bin/find
$ execstack -q /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"):

```bash
$ execstack -s /usr/bin/find
$ readelf -a /usr/bin/find | grep GNU_STACK -A 1
  GNU_STACK 0x0000000000000000 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"):

```bash
$ execstack -c /usr/bin/find
$ readelf -a /usr/bin/find | grep GNU_STACK -A 1
  GNU_STACK 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 RW 8
```
Part 7. Booting and shutdown

This section describes device drivers and features that are used in the context of booting and shutting down Linux.

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

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 release notes at
www.novell.com/linux/releasenotes/s390x/SUSE-SLES/11/

Chapter 32. Console device drivers ........................................ 259
Console features .............................................................. 259
What you should know about the console device drivers .............. 260
Setting up the console device drivers .................................... 264
Working with Linux terminals ............................................. 268

Chapter 33. Initial program loader for System z - zipl ............... 275
Usage .............................................................................. 275
Parameters ......................................................................... 288
Configuration file structure .................................................. 291

Chapter 34. Booting Linux ...................................................... 297
IPL and booting ..................................................................... 297
Control point and boot medium .......................................... 298
Menu configurations .......................................................... 298
Boot data ........................................................................... 299
Booting a z/VM Linux guest ............................................... 300
Booting Linux in LPAR mode .............................................. 305
Displaying current IPL parameters ...................................... 310
Re-booting from an alternative source .................................. 311
Using a VM parameter string for Linux re-IPL ....................... 313

Chapter 35. Shutdown actions ................................................ 315
Examples ............................................................................ 316
Chapter 32. Console device drivers

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

The only interface to a Linux instance in an LPAR before the boot process is completed is the Hardware Management Console (HMC), see Figure 43. After the boot process has completed, you typically use a network connection to access Linux through a user login, for example, in a telnet or ssh session. The possible connections depend on the configuration of your particular Linux instance.

If you run Linux as a z/VM guest operating system, you typically log in to z/VM first, using a 3270 terminal or terminal emulator. From the 3270 terminal you IPL the Linux boot device. Again, after boot you typically use a network connection to access Linux through a user login rather than a 3270 terminal.

Console features

The console device drivers support the following:
HMC applets
You can use two applets.

Operating System Messages
This is a line-mode terminal. See Figure 44 for an example.

Integrated ASCII Console
This is a full-screen mode terminal.

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

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

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

Figure 44. Linux kernel messages on the HMC Operating System Messages applet

What you should know about the console device drivers
This section defines some of the terms used in the context of the console device drivers and provides information about console device names and nodes, about terminal modes, and about how console devices are accessed.

About the terminology
Terminal and console have special meanings in Linux.

A Linux terminal
is an input/output device through which users interact with Linux and Linux applications. Typically, a login program or a shell runs in a Linux terminal and provides access to the Linux system.
The Linux console
is an output device that displays Linux kernel messages.

A mainframe terminal
is any device that gives a user access to operating systems and applications running on the mainframe. This could be a physical device such as a 3270 terminal hardware linked to the mainframe through a controller, or it can be a terminal emulator on a workstation connected through a network. For example, you access z/OS through a mainframe terminal.

The HMC
is a device that gives a system programmer control over the hardware resources, for example the LPARs. The HMC is a Web application on a Web server that is connected to the support element (SE). The HMC can be accessed from the SE but more commonly is accessed from a workstation within a secure network.

Console device
in the context of the console device drivers, a device, as seen by Linux, to which Linux kernel messages can be directed.

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

Before you have a terminal - the zipl boot menu
Depending on your setup, a zipl boot menu might be displayed when you IPL. The zipl boot menu is part of the boot loader that loads the Linux kernel. Do not confuse the zipl boot menu with the Linux terminal, which has not been set up at this point. The zipl boot menu is very limited in its functionality, for example, there is no way to specify uppercase letters as all input is converted to lowercase. For more details about booting Linux, see Chapter 34, “Booting Linux,” on page 297. For more details about the zipl boot menu, see Chapter 33, “Initial program loader for System z - zipl,” on page 275.

Device and console names
Each terminal device driver can provide a single console device. Table 40 lists the terminal device drivers and the corresponding device names and console names.

<table>
<thead>
<tr>
<th>Device driver</th>
<th>Device name</th>
<th>Console name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCLP line-mode terminal device driver</td>
<td>sclp_line0</td>
<td>ttyS0</td>
</tr>
<tr>
<td>SCLP VT220 terminal device driver</td>
<td>ttySclp0</td>
<td>ttyS1</td>
</tr>
<tr>
<td>3215 line-mode terminal device driver</td>
<td>ttyS0</td>
<td>ttyS0</td>
</tr>
<tr>
<td>3270 terminal device driver</td>
<td>tty0.0.009</td>
<td>tty3270</td>
</tr>
</tbody>
</table>

As shown in Table 40, the console with name ttyS0 can be provided either by the SCLP console device driver or by the 3215 line-mode terminal device driver. The system environment and settings determine which device driver provides ttyS0. For details see the information about the conmode parameter in “Console kernel parameter syntax” on page 264.

You require a device node to make a console device available to applications, for example to a login program (see “Device nodes” on page 262).
Device nodes

Applications access console devices by device nodes. For example, with the default conmode settings, udev creates the following device nodes for console devices:

Table 41. Device nodes created by udev

<table>
<thead>
<tr>
<th>Device driver</th>
<th>On LPAR</th>
<th>On z/VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCLP line-mode terminal device driver</td>
<td>/dev/sclp_line0</td>
<td>n/a</td>
</tr>
<tr>
<td>SCLP VT220 terminal device driver</td>
<td>/dev/ttysclp0</td>
<td>/dev/ttysclp0</td>
</tr>
<tr>
<td>3215 line-mode terminal device driver</td>
<td>n/a</td>
<td>/dev/ttyS0</td>
</tr>
<tr>
<td>3270 terminal device driver</td>
<td>/dev/tty0.0.0009</td>
<td>/dev/tty0.0.0009</td>
</tr>
</tbody>
</table>

Terminal modes

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

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 like the vi editor.

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

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

Table 42 summarizes when to expect which terminal mode.

Table 42. Terminal modes

<table>
<thead>
<tr>
<th>Accessed through</th>
<th>Environment</th>
<th>Device driver</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System Messages</td>
<td>LPAR</td>
<td>SCLP line-mode terminal device driver</td>
<td>Line mode</td>
</tr>
<tr>
<td>applet on the HMC</td>
<td>z/VM</td>
<td>SCLP VT220 terminal device driver</td>
<td>Full-screen mode</td>
</tr>
<tr>
<td>operating System Message</td>
<td>z/VM or LPAR</td>
<td>SCLP VT220 terminal device driver</td>
<td>Full-screen mode</td>
</tr>
<tr>
<td>Messages applet emulation on z/VM</td>
<td>z/VM with CONMODE=3215</td>
<td>3215 line-mode terminal device driver</td>
<td>Line mode</td>
</tr>
<tr>
<td>Integrated ASCII Console</td>
<td>z/VM or LPAR</td>
<td>3215 line-mode terminal device driver</td>
<td>Line mode</td>
</tr>
<tr>
<td>applet on the HMC</td>
<td>z/VM with CONMODE=3270</td>
<td>3270 terminal device driver</td>
<td>Block mode</td>
</tr>
</tbody>
</table>

The 3270 terminal device driver provides three different views. See “Switching the views of the 3270 terminal device driver” on page 268 for details.
How console devices are accessed

How you can access console devices depends on your environment. The diagrams in the following sections omit device drivers that are not relevant for the particular access scenario.

Using the HMC for Linux in an LPAR

Figure 45 shows the possible terminal devices for Linux instances that run directly in an LPAR.

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

Using the HMC for Linux under z/VM

If the ASCII system console has been attached to the z/VM guest virtual machine where the Linux instance runs, you can access the ttyS1 terminal device from the HMC Integrated ASCII Console applet (see Figure 46).

Using 3270 terminal hardware or a 3270 terminal emulation for Linux under z/VM

For a Linux instance that runs as a z/VM guest operating system, you can use 3270 terminal hardware or a 3270 terminal emulation to access a console device. Figure 47 on page 264 illustrates how z/VM can handle the 3270 communication.
Figure 47. Accessing terminal devices from a 3270 device

Note: Figure 47 shows two console devices with the name ttyS0. Only one of these devices can be present at any one time.

**CONMODE=3215**
performs a translation between the 3270 protocol and the 3215 protocol and connects the 3270 terminal hardware or emulation to the 3215 line-mode terminal device driver in the Linux kernel.

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

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

**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.

**Console kernel parameter syntax**
You can use the conmode and console kernel parameters to configure the console device drivers.

```
  conmode=<mode>
  console=<device>
```

where:
conmode

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

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

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

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

Table 43. Default device driver for the line-mode terminal device

<table>
<thead>
<tr>
<th>LPAR</th>
<th>SCLP line-mode terminal device driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/VM</td>
<td>3215 line-mode terminal device driver or 3270 terminal device driver, depending on the z/VM guest's console settings (the CONMODE field in the output of <code>CP QUERY TERMINAL</code>).</td>
</tr>
<tr>
<td></td>
<td>If the device driver you specify with the conmode= kernel parameter contradicts the CONMODE z/VM setting, z/VM is reconfigured to match the specification for the kernel parameter.</td>
</tr>
</tbody>
</table>

You can use the conmode parameter to override the default.

sclp or hwc

specifies the SCLP line-mode terminal device driver.

You need this specification if you want to use the VINPUT command under z/VM (["Using an emulated HMC Operating System Messages applet under z/VM" on page 271].)

3270

specifies the 3270 device driver.

3215

specifies the 3215 device driver.

console

dspects which devices are to be activated to receive Linux kernel messages.

You can activate two terminal devices to receive Linux kernel messages by specifying a console parameter for each terminal device.

If you activate two terminal devices, the second specification designates the preferred console. The preferred console is used as initial terminal device, beginning at the stage of the boot process when the 'init'-program is called. Messages issued by programs that are run at this stage are therefore only displayed on the preferred console. Only one of the activated terminal devices can be the preferred console.

By default, ttyS0 is activated and used as the preferred console. Use the console parameter to activate ttyS1 or tty3270 instead of ttyS0, or to activate two terminal devices and designate one of them as the preferred console.

<name1>

is the name of the console to be activated.

<name2>

is the name of a second console if both are to be active. If both consoles are
specify, this is the preferred console.
Use the console names according to Table 40 on page 261 for \( <name1> \) and \( <name2> \).

**Examples**

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

- To activate the full-screen terminal device instead of the line-mode terminal device, add the following line to the kernel command line:
  
  ```
  console=ttys1
  ```

- To activate ttys0 and ttys1, provide a specification for both drivers, for example:
  
  ```
  console=ttys0 console=ttys1
  ```

  The last statement determines the preferred console, ttys1 in the example.

- To use the 3270 terminal device driver and activate tty3270, add the following line to the kernel command line:
  
  ```
  conmode=3270 console=3270
  ```

  With conmode=tty3270 you must explicitly activate an available device because there is no automatically activated ttys0.

**Setting up a line-mode terminal**

The line-mode terminals are primarily intended for booting Linux. The preferred user access to a running SUSE Linux Enterprise Server instance is through a user login that runs, for example, in a telnet or ssh session. See “Terminal modes” on page 262 for information about the available line-mode terminals.

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

```bash
# export TERM=dumb
```

**Setting up a full-screen mode terminal**

The full-screen terminal can be used for full-screen text editors, such as vi, and terminal-based full-screen system administration tools. See “Terminal modes” on page 262 for information about the available full-screen mode terminals.

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

```bash
# export TERM=linux
```

**Setting up a terminal provided by the 3270 terminal device driver**

The terminal provided by the 3270 terminal device driver is neither a line-mode terminal nor a typical full-screen mode terminal. The terminal provides limited support for full-screen applications. For example, the ned editor is supported, but not vi.
Tip: If the terminal does not provide the expected output, ensure that linux is assigned to the TERM environment variable. For example, issue the following command on the bash shell:

```
# export TERM=linux
```

### Enabling a terminal for user logins using inittab

You can use an inittab entry to allow user logins from a terminal. To enable user logins with the mingetty program, add a line of this form to the /etc/inittab file:

```
<id>:2345:respawn:/sbin/mingetty --noclear <term> <dev>
```

where:
- `<id>` is a unique identifier for the entry in the inittab file.
- `<dev>` specifies the device node of the terminal, omitting the /dev (see Table 41 on page 262). For example, instead of specifying /dev/sclp_line0, specify sclp_line0.
- `<term>` optionally specifies the terminal name. The terminal name indicates the capabilities of the terminal device. Examples for terminal names are linux, dumb, xterm, or vt220; dumb is the default.

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 or ID. See Table 41 on page 262 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 41 on page 262 you can comment it out and replace it.

If you want to permit user logins on tty0.0.0009, you must add it to /etc/securetty.

For more details see the man page for the inittab file.

### Examples

To enable the line-mode device ttyS0 for user logins with mingetty specify, for example:

```
a:2345:respawn:/sbin/mingetty --noclear dumb ttyS0
```

To enable the tty0.0.0009 device for user logins with mingetty specify, for example:

```
b:2345:respawn:/sbin/mingetty --noclear linux tty0.0.0009
```

To enable the full-screen mode device ttysclp0 for user logins with mingetty specify, for example:

```
c:2345:respawn:/sbin/mingetty ttysclp0 --noclear linux
```

### Setting up the code page for an x3270 emulation on Linux

If you are accessing z/VM from Linux by using the x3270 terminal emulation, add the following settings to the .Xdefaults file to get the correct code translation:

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

This section describes typical tasks that you need to perform when working with Linux terminals.
- "Using the terminal applets on the HMC"
- "Switching the views of the 3270 terminal device driver"
- "Setting a CCW terminal device online or offline" on page 269
- "Entering control and special characters on line-mode terminals" on page 270
- "Using the magic sysrequest functions on line-mode terminals" on page 270
- "Using an emulated HMC Operating System Messages applet under z/VM" on page 271
- "Simulating the Enter and Spacebar keys" on page 274
- "Using a 3270 terminal in 3215 mode" on page 274

Using the terminal applets on the HMC

This section applies to both the line-mode terminal and the full-screen mode terminal on the HMC:
- On an HMC you can only open each applet once.
- Within an LPAR, there can only be one active terminal session for each applet, even if multiple HMCs are used.
- A particular Linux instance supports only one active terminal session for each applet.
- Security hint: Always end a terminal session by explicitly logging off (for example, type "exit" and press Enter). Simply closing the applet leaves the session active and the next user opening the applet resumes the existing session without a logon.
- Slow performance of the HMC is often due to a busy console or increased network traffic.

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

Switching the views of the 3270 terminal device driver

The 3270 terminal device driver provides three different views. Use function key 3 (PF3) to switch between the views.
The Linux kernel messages view is available only if the terminal device has been activated for Linux kernel messages. The full-screen application view is available only if there is an application that uses this view, for example, the ned editor.

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

For the Linux kernel messages view and the terminal I/O view you can use the PF7 key to scroll backward and the PF8 key to scroll forward. The scroll buffers are fixed at 4 pages (16 KB) for the Linux kernel messages view and 5 pages (20 KB) for the terminal I/O view. When the buffer is full and more terminal data needs to be printed, the oldest lines are removed until there is enough room. The number of lines in the history, therefore, vary. Scrolling in the full-screen application view depends on the application.

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

Setting a CCW terminal device online or offline

This section applies to Linux instances that run as z/VM guest operating systems.

The 3270 terminal device driver uses CCW devices and provides them as CCW terminal devices. A CCW terminal device can be:

- The tty3270 terminal device that is can be activated for receiving Linux kernel messages.
  If this device exists, it comes online early during the Linux boot process. In a default z/VM environment, the device number for this device is 0009. In sysfs it is represented as /sys/bus/ccw/drivers/3270/0.0.0009. You need not set this device online and you must not set it offline.
- CCW terminal devices through which users can log in to Linux with the CP DIAL command.
  These devices are defined with the CP DEF GRAF command. They are represented in sysfs as /sys/bus/ccw/drivers/3270/0.<n>.<devno> where <n> is the subchannel set ID and <devno> is the virtual device number. By setting these devices online you enable them for user logins. If you set a device offline it can no longer be used for user login.

See z/VM CP Commands and Utilities Reference for more information about the DEF GRAF and DIAL commands.
You can use the chccwdev command (see "chccwdev - Set a CCW device online" on page 334) to set a CCW terminal device online or offline. Alternatively, you can write "1" to the device’s online attribute to set it online, or "0" to set it offline.

**Examples**

- To set a CCW terminal device 0.0.7b01 online issue:
  
  ```
  # chccwdev -e 0.0.7b01
  
  or
  
  # echo 1 > /sys/bus/ccw/drivers/3270/0.0.7b01/online
  ```

- To set a CCW terminal device 0.0.7b01 offline issue:
  
  ```
  # chccwdev -d 0.0.7b01
  
  or
  
  # echo 0 > /sys/bus/ccw/drivers/3270/0.0.7b01/online
  ```

**Entering control and special characters on line-mode terminals**

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

Another problem on line-mode terminals is how to enter a character string without a newline character at the end. Pressing the Enter key adds a newline character to your string which is not expected by some applications.

**Table 44** summarizes how you can use the caret character (^) to enter some control characters and to enter strings without appended newline characters.

**Table 44. Control and special characters on line-mode terminals**

<table>
<thead>
<tr>
<th>For the key combination</th>
<th>Type this</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+C</td>
<td>^c</td>
<td>Cancel the process that is currently running in the foreground of the terminal.</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>^d</td>
<td>Generate an end of file (EOF) indication.</td>
</tr>
<tr>
<td>Ctrl+Z</td>
<td>^z</td>
<td>Stop a process.</td>
</tr>
<tr>
<td>n/a</td>
<td>^n</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

**Using the magic sysrequest functions on line-mode terminals**

The two characters “^-” (caret and hyphen) 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.
Table 45. Magic sysrequest commands

<table>
<thead>
<tr>
<th>Enter</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>^-b</td>
<td>Re-IPL immediately (see <code>lsreipl - List IPL settings</code> on page 376).</td>
</tr>
<tr>
<td>^-s</td>
<td>Emergency sync all file systems.</td>
</tr>
<tr>
<td>^-u</td>
<td>Emergency remount all mounted file systems read-only.</td>
</tr>
<tr>
<td>^-t</td>
<td>Show task info.</td>
</tr>
<tr>
<td>^-m</td>
<td>Show memory.</td>
</tr>
<tr>
<td>^- followed by a digit (0 to 9)</td>
<td>Set the console log level.</td>
</tr>
<tr>
<td>^-e</td>
<td>Terminate all tasks except init.</td>
</tr>
<tr>
<td>^-i</td>
<td>Kill all tasks except init.</td>
</tr>
</tbody>
</table>

Activating and deactivating the magic sysrequest function

From a Linux terminal or a command prompt, enter the following command to activate the magic sysrequest function:

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

Enter the following command to deactivate the magic sysrequest function:

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

You need root authority to activate or deactivate the magic sysrequest functions.

**Tip:** You can use YaST to activate and deactivate the magic sysrequest function. Go to `yast -> system -> Kernel Settings`, select or clear the **enable SYSREQ** option and leave YaST with **OK**.

Using an emulated HMC Operating System Messages applet under z/VM

The preferred terminal devices for Linux instances that run as z/VM guests are the devices provided by the 3215 or 3270 terminal device drivers. If you need to use the "Operating System Messages" applet emulation, for example, because the 3215 terminal is not operational, you must use the **CP VINPUT** command to prefix any input.

The **VINPUT** command accesses the ttyS0 terminal device. **VINPUT** requires that this device is provided by the SCLP line-mode terminal device driver. To be able to use, **VINPUT**, you have to override the default device driver for z/VM environments (see "Console kernel parameter syntax" on page 264).

**VINPUT** is a z/VM CP command. It can be abbreviated to **VI** but must not be confused with the Linux command **vi**.

If you use the SCLP console driver running under z/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 z/VM. The following examples are written at the input line of a 3270 terminal or terminal emulator (for example, x3270).

If you are in the CP READ mode, omit the leading “#CP” from the commands.

For more information on **VINPUT** refer to *z/VM CP Commands and Utilities Reference*, SC24-6008.

**Priority and non-priority commands**

**VINPUT** commands require a VMSG (non-priority) or PVMSG (priority) specification. Operating systems that honour this specification process priority commands with a higher priority than non-priority commands.

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

Linux does not distinguish priority and non-priority commands.

**Example:** The specifications:

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

and

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

are equivalent.

**Case conversion**

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

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

Linux and bash are case sensitive and require some specifications with uppercase characters. To allow uppercase characters to be passed to Linux, the hardware console uses an escape character “%” under z/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:** In the following examples, the first line shows the user input, the second line shows what the device driver receives after the case conversion by CP, and the third line shows the command processed by bash:

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

```
#cp vinput vmsg echo $PATH
CP VINPUT VMG 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 VMG 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 [Using a 3270 terminal in 3215 mode](page 274), 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 [Using a 3270 terminal in 3215 mode](page 274), 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
```
Simulating the Enter and Spacebar keys

If you are using an emulation of the HMC “Operating System Messages” applet under z/VM, 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```

Using a 3270 terminal in 3215 mode

When running under z/VM, the control program (CP) defines five characters as line editing symbols. Use the CP QUERY TERMINAL command to see the current settings.

The default line editing symbols depend on your terminal emulator. You can reassign the symbols by changing the settings of LINEND, TABCHAR, CHARDEL, LINEDEL, or ESCAPE with the CP TERMINAL command. Table 46 shows the most commonly used settings:

Table 46. Line edit characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Symbol</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>LINEND</td>
<td>The end of line character allows you to enter several logical lines at once.</td>
</tr>
<tr>
<td>[ or \c</td>
<td>LINEDEL</td>
<td>The line delete symbol deletes everything back to and including the previous LINEND symbol or the start of the input. &quot;[&quot; is common for ASCII terminals and &quot;\c&quot; for EBCDIC terminals.</td>
</tr>
<tr>
<td>&quot;</td>
<td>ESCAPE</td>
<td>The escape character allows you to enter a line edit symbol as a normal character.</td>
</tr>
<tr>
<td>I</td>
<td>TABCHAR</td>
<td>The logical tab character.</td>
</tr>
<tr>
<td>@</td>
<td>CHARDEL</td>
<td>The character delete symbol deletes the preceding character.</td>
</tr>
</tbody>
</table>

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

Examples

The following examples assume the settings of Table 46 with the opening bracket character (\[) as the delete line character.

- To specify a tab character specify:
  
  "\["

- To specify a the double quote character specify:
  
  ""

- If you type the character string:
  
  `#CP HALT#CP ZIPL 190[#CP IPL 10290 PARM vmpoff="MSG OP REBOOT#IPL 290"

  the actual commands received by CP are:

  CP HALT
  CP IPL 290 PARM vmpooff="MSG OP REBOOT#IPL 290"`
Chapter 33. Initial program loader for System z - zipl

zipl can be used to prepare a device for one of the following purposes:

- Booting Linux (as a Linux program loader)
- Dumping

For more information on the dump tools that zipl installs and on using the dump functions, refer to Linux on System z Using the Dump Tools, SC33-8412.
- 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 278).

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>
<thead>
<tr>
<th>Base function</th>
<th>Command line short option</th>
<th>Command line long option</th>
<th>Configuration file option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install a boot loader</td>
<td>-i</td>
<td>--image=</td>
<td>image=</td>
</tr>
<tr>
<td>Prepare a DASD or tape dump device</td>
<td>-d</td>
<td>--dumpto=</td>
<td>dumpto=</td>
</tr>
<tr>
<td>Prepare a list of ECKD volumes for a multi-volume dump</td>
<td>-M</td>
<td>--mvdump</td>
<td>mvdump=</td>
</tr>
<tr>
<td>Prepare a SCSI dump device</td>
<td>-D</td>
<td>--dumptofs=</td>
<td>dumptofs=</td>
</tr>
</tbody>
</table>

See “Preparing a boot device” on page 279 for details.

See “Preparing a DASD or tape dump device” on page 281 for details.

See “Preparing a multi-volume dump on ECKD DASD” on page 282 for details.

See “Preparing a dump device on a SCSI disk” on page 284 for details.
### Table 47. zipl base functions (continued)

<table>
<thead>
<tr>
<th>Base function</th>
<th>Command line short option</th>
<th>Command line long option</th>
<th>Configuration file option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prepare a device to load a file to initialize discontiguous named saved segments</td>
<td>-s</td>
<td>--segment</td>
<td>segment=</td>
</tr>
</tbody>
</table>

See "Installing a loader to initialize a discontiguous named saved segment (DCSS)" on page 286 for details.

| Install a menu configuration                                                 | -m                        | --menu                   | (None)                   |

See "Installing a menu configuration" on page 287 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 287), the entire configuration must be defined using command-line parameters. See the following base functions for how to specify command-line parameters:

- "Preparing a boot device" on page 279
- "Preparing a DASD or tape dump device" on page 281
- "Preparing a multi-volume dump on ECKD DASD" on page 282
- "Preparing a dump device on a SCSI disk" on page 284
- "Installing a loader to initialize a discontiguous named saved segment (DCSS)" on page 286

**Configuration-file mode**

If a zipl command is issued either without a base function or to install a menu configuration, a configuration file is accessed. See "Configuration file structure" on page 291 for more information.
zipl syntax overview

parameters when omitting base function:

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 291).
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 280 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 286.
page 286 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 bootstrap file. Use this option when these files are spread across multiple disks to ensure that they are available at IPL time. Specifying this option significantly increases the size of the bootmap file created in the target directory.

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

-V provides verbose command output.

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

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

-v displays version information.

-h displays help information.

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

See "Parameters" on page 288 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 ZIPCONF) 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> -t <directory> -r <ramdisk> -p <parmfile> -a
```

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

**The location** `<image>`
- The location 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 280 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
Use this option when these files are spread across multiple disks to ensure that they are available at IPL time. This option is available on the command line only. Specifying this option significantly increases the size of the bootmap file created in the target directory.

See "Parameters" on page 288 for a summary of the parameters including the long options you can use on the command line.

Figure 49 summarizes how you can specify a boot configuration within a configuration file section. Required specifications are shown in bold. See "Configuration file structure" on page 291 for a more comprehensive discussion of the configuration file.

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.

**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 z Using the Dump Tools*, SC33-8412 for details on how to process these dumps.

See [Parameters](#) on page 288 for a summary of the parameters including the long options you can use on the command line.

Figure 50 summarizes how you can specify a DASD or tape dump configuration in a configuration file. See [Configuration file structure](#) on page 291 for a more comprehensive discussion of the configuration file.

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

Figure 50. 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:

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

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

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

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

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

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

### Preparing a multi-volume dump on ECKD DASD

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

```
zipl [-f] [-M <dump_device_list>] [-s <size>] [-n]
```

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

**A file `-M <dump_device_list>`**

containing the device nodes of the dump partitions, separated by one or more line feed characters (0x0a). `zipl` writes a dump signature to each involved partition and installs the stand-alone multi-volume dump tool on each involved volume. Duplicate partitions are not allowed. A maximum of 32 partitions can be listed. The volumes must be formatted with cdl. You can use any block size, even mixed block sizes. However, to speed up the dump process and to reduce wasted disk space, use block size 4096.
Optionally, you can also specify:

**An option -f or --force**

to force that no signature checking will take place when dumping. Any data on all involved partitions will be overwritten without warning.

**An option -n**

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

**A limit <size>**

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

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

DASD or tape dump devices are not formatted with a file system so no target directory can be specified. Refer to *Linux on System z Using the Dump Tools*, SC33-8412 for details on how to process these dumps.

See "Parameters" on page 288 for a summary of the parameters including the long options you can use on the command line.

**Figure 51** summarizes how you can specify a multi-volume DASD dump configuration in a configuration file. See "Configuration file structure" on page 291 for a more comprehensive discussion of the configuration file.

```plaintext
[<section_name>]
mvdump=<dump_device_list>,<size>
```

**Example**

The following command prepares two DASD partitions /dev/dasdc1, /dev/dasdd1 for a multi-volume dump and suppresses confirmation prompts that require an operator response:

```
# zipl -M sample_dump_conf -n
```

where the sample_dump_conf file contains the two partitions separated by line breaks:

/dev/dasdc1
/dev/dasdd1

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

```plaintext
[multi_volume_dump]
mvdump=sample_dump_conf
```

There is no configuration file equivalent for option -n. To use this option for a multi-volume DASD dump configuration in a configuration file it needs to be specified with the `zipl` command that processes the configuration.

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

```bash
# zipl -c /etc/myxmp.conf multi_volume_dump -n
```

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 z Using the Dump Tools, SC33-8412 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 Linux on System z Using the Dump Tools, SC33-8412 for a discussion of how multiple parameter specifications are combined.

  ```bash
  dump_dir=/<directory>
  ```

  Path to the directory (relative to the root of the dump partition) where the dump file is to be written. This directory is specified with a leading slash. The directory must exist when the dump is initiated.

  **Example:** If the dump partition is mounted as `/dumps`, and the parameter “dump_dir=/mydumps” is defined, the dump directory would be accessed as `/dumps/mydumps`.
  The default is “/” (the root directory of the partition).
dump_compress=gzip|none
Dump compression option. Compression can be time-consuming on slower systems with a large amount of memory.

The default is “none”.

dump_mode=interactive|auto
Action taken if there is no room on the file system for the new dump file. “interactive” prompts the user to confirm that the dump with the lowest number is to be deleted. “auto” automatically deletes this file.

The default is “interactive”.

If you provide multiple parameters, separate them with a blank and enclose them within single quotes (’) or double quotes (“”).

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

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

See “Parameters” on page 288 for a summary of the parameters including the long options you can use on the command line.

Figure 52 summarizes how you can specify a SCSI dump configuration in a configuration file. Required specifications are shown in bold. See “Configuration file structure” on page 291 for a more comprehensive discussion of the configuration file.

[<section_name>]
dumptofs=<dump_partition>
parmfile=<parmfile>,<parm_addr>
parameters=<parameters>
target=<directory>

Figure 52. zipl syntax for preparing a SCSI dump device — configuration file mode

Example
The following command prepares a SCSI partition /dev/sda2 as a dump device and a directory /boot as the target directory. Dumps are to be written to a directory mydumps, relative to the mount point. There is to be no compression but instead the oldest dump will be automatically deleted if there is not enough space for the new dump.

```
# zipl -D /dev/sda2 -P 'dumpdir=/mydumps dump_compress=none dump_mode=auto' -t /boot
```

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

```
[dumpscsi]
dumptofs=/dev/sda2
parmfile='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:

```bash
# 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

```bash
zipl command line syntax for loading a DCSS

```

To prepare a device for loading a data file to initialize discontiguous named saved segments, you must specify:

**The source file** `<segment_file>`

- to be loaded at IPL time.

**The segment address** `<seg_addr>`

- to which the segment is to be written at IPL time.

**A target** `<directory>`

- `zipl` installs the boot loader code on the device containing the specified directory `<directory>`.
After the segment has been loaded, the system is put into the disabled wait state. No Linux instance is started.

See "Parameters on page 288" for a summary of the parameters including the long options you can use on the command line.

Figure 53 summarizes how you can specify a file to be loaded to a DCSS within a configuration file section. See "Configuration file structure on page 291" for a more comprehensive discussion of the configuration file.

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:

```bash
# zipl -s /boot/segment,0x40000000 -t /boot
```

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

```plaintext
[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:

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

Installing a menu configuration
To prepare a menu configuration you need a configuration file that includes at least one menu.

zipl syntax for installing a menu configuration

```bash
zipl --m <menu_name> -c /etc/zipl.conf -c <config_file> -a
```

Notes:
1. You can change the default configuration file with the ZIPCONF 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 bootstrap 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 bootstrap file created in the target directory.

Example
Using the example of a configuration file in [Example on page 293,] you could
install a menu configuration with:

```
# zipl -m menu1
```

## Parameters

This section provides an overview of the options and how to specify them on the
command line or in the configuration file.

<table>
<thead>
<tr>
<th>Command line short option</th>
<th>Command line long option</th>
<th>Configuration file option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c &lt;config_file&gt;</td>
<td>--config=&lt;config_file&gt;</td>
<td>n/a</td>
<td>Specifies the configuration file. You can change the default configuration file /etc/zipl.conf with the environment variable ZIPLCONF.</td>
</tr>
<tr>
<td>n/a</td>
<td></td>
<td>&lt;configuration&gt;</td>
<td>Specifies a configuration section to be read and processed from the configuration file.</td>
</tr>
<tr>
<td>n/a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-i &lt;image&gt;[,&lt;image_addr&gt;]</td>
<td>--image=&lt;image&gt;[,&lt;image_addr&gt;]</td>
<td>n/a</td>
<td>Specifies the location of the Linux kernel image on the file system and, optionally, in memory after IPL. The default memory address is 0x10000.</td>
</tr>
<tr>
<td>image=&lt;image&gt;[,&lt;image_addr&gt;]</td>
<td></td>
<td></td>
<td>See &quot;Preparing a boot device&quot; on page 279 for details.</td>
</tr>
<tr>
<td>-r &lt;ramdisk&gt;[,&lt;initrd_addr&gt;]</td>
<td>--ramdisk=&lt;ramdisk&gt;[,&lt;initrd_addr&gt;]</td>
<td>n/a</td>
<td>Specifies the location of the initial RAM disk (initrd) on the file system and, optionally, in memory after IPL. The default memory address is 0x80000.</td>
</tr>
</tbody>
</table>
Command line short option | Command line long option | Explanation
---|---|---
-p | --parmfile | In conjunction with a boot configuration, specifies the location of a kernel parameter file.
parmfile | parmfile= | 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 284).
parm_addr | parm_addr= | 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 286 and [How kernel parameters from different sources are combined](#) on page 280 for more information.

<parmfile> 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 | In conjunction with a boot configuration, specifies kernel parameters.
parameters | parameters= | In conjunction with a SCSI dump configuration, specifies SCSI system dumper parameters (see [Preparing a dump device on a SCSI disk](#) on page 284).
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 286 and [How kernel parameters from different sources are combined](#) on page 280 for more information.

dump_device | dump_device= | Specifies the DASD partition or tape device to which a dump is to be written after IPL.
size | 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 281 and Linux on System z Using the Dump Tools, SC33-8412 for details.

-M | --mvdump | Specifies a file with a list of DASD partitions to which a dump is to be written after IPL.
mvdump | mvdump= | The optional size specification limits the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary. If you limit the dump size below the amount of memory used by the system to be dumped, the resulting dump is incomplete. If no limit is provided, all of the available physical memory is dumped.

See [Preparing a multi-volume dump on ECKD DASD](#) on page 282 and Linux on System z Using the Dump Tools, SC33-8412 for details.
<table>
<thead>
<tr>
<th>Command line short option</th>
<th>Command line long option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-D</strong></td>
<td>&lt;dump_partition&gt;[,&lt;size&gt;] or <strong>dumptofs</strong>=&lt;dump_partition&gt;[,&lt;size&gt;]</td>
<td>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 284 and Linux on System z Using the Dump Tools, SC33-8412 for details.</td>
</tr>
<tr>
<td><strong>-s</strong></td>
<td>&lt;segment_file&gt;,&lt;seg_addr&gt; or <strong>segment</strong>=&lt;segment_file&gt;,&lt;seg_addr&gt;</td>
<td>Specifies the segment file to load at IPL time and the memory location for the segment.</td>
</tr>
<tr>
<td><strong>-t</strong></td>
<td>&lt;directory&gt; or <strong>target</strong>=&lt;directory&gt;</td>
<td>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).</td>
</tr>
<tr>
<td><strong>-T</strong></td>
<td>&lt;tape_node&gt; or <strong>tape</strong>=&lt;tape_node&gt;</td>
<td>Specifies the tape device where zipl installs the boot loader code.</td>
</tr>
<tr>
<td><strong>-m</strong></td>
<td>&lt;menu_name&gt; or <strong>menu</strong>=&lt;menu_name&gt;</td>
<td>Specifies the name of the menu that defines a menu configuration in the configuration file (see “Menu configurations” on page 292).</td>
</tr>
<tr>
<td><strong>-n</strong></td>
<td><strong>--noninteractive</strong></td>
<td>Suppresses all confirmation prompts (for example, when preparing a DASD or tape dump device).</td>
</tr>
<tr>
<td><strong>-a</strong></td>
<td><strong>--add-files</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>-V</strong></td>
<td><strong>--verbose</strong></td>
<td>Provides more detailed command output.</td>
</tr>
<tr>
<td><strong>-v</strong></td>
<td><strong>--version</strong></td>
<td>Prints version information.</td>
</tr>
</tbody>
</table>
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 279 for details.

- **dumpto=**<dump_device>
  Defines a DASD or tape dump configuration. See "Preparing a DASD or tape dump device" on page 281 for details.

- **mvdump=**<dump_device_list>

- **dumptofs=**<dump_partition>
  Defines a SCSI dump configuration. See "Preparing a dump device on a SCSI disk" on page 284 for details.

- **segment=**<segment_file>
  Defines a DCSS load configuration. See "Installing a loader to initialize a discontiguous named saved segment (DCSS)" on page 286 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 54 on page 294](#) 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/root ro'
image=/boot/image-1
target=/boot

# Second boot configuration (SCSI)
[boot2]
image=/boot/mnt/image-2
ramdisk=/boot/mnt/initrd,0x900000
parmfile=/boot/mnt/parmf-2
target=/boot

# Third boot configuration (DASD)
[boot3]
image=/boot/mnt/image-3
ramdisk=/boot/mnt/initrd
parmfile=/boot/mnt/parmf-3
target=/boot

# Configuration for dumping to tape
[dumptape]
dumpto=/dev/rtibm0

# Configuration for dumping to DASD
[dumpdasd]
dumpto=/dev/dasdc1

# Configuration for multi-volume dumping to DASD
[multi_volume_dump]
mvdump=sample_dump_conf

# Configuration for dumping to SCSI disk
# Separate IPL and dump partitions
[dumpscsi]
target=/boot
dumptofs=/dev/sda2
parameters="dump_dir=/mydumps dump_compress=none dump_mode=auto"

# Menu containing the SCSI boot and SCSI dump configurations
:menu1
1=dumpscsi
2=boot2
target=/boot
default=2

# Menu containing two DASD boot configurations
:menu2
1=boot1
2=boot3
target=/boot
default=1
prompt=1
timeout=30

# Configuration for initializing a DCSS
[segment]
segment=/boot/segment,0x800000
target=/boot

Figure 54. /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:

  ```shell
  # 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 287):

  ```shell
  # 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 302 and "Example for a DASD menu configuration (LPAR)" on page 308 illustrate what this menu looks like when it is displayed.

- Call `zipl` to install a boot loader for boot configuration `[boot2]`:

  ```shell
  # 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:

  ```shell
  # zipl dumptape
  ```

  **Result:** `zipl` selects the `[dumptape]` section and prepares a dump tape on `/dev/rtibm0`.

- Call `zipl` to prepare a DASD dump device:

  ```shell
  # 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:

  ```shell
  # 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/sdal`. 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:
Result: zipl installs segment loader that will load the contents of file
/boot/segment to address 0x800000 at IPL time and then put the processor into
the disabled wait state.
Chapter 34. Booting Linux

This chapter provides a general overview of how to boot Linux in an LPAR or as a z/VM guest. For details on how to define a Linux virtual machine, see z/VM Getting Started with Linux on System z, SC24-6096, the chapter on creating your first Linux virtual machine.

IPL and booting

On System z, you usually start booting Linux by performing an Initial Program Load (IPL). Figure 55 summarizes the main steps.

Figure 55. IPL and boot process

The IPL process accesses the IPL device and loads the Linux boot loader code to the mainframe memory. The boot loader code then gets control and loads the Linux kernel. At the end of the boot process Linux gets control.

If your Linux instance is to run in an LPAR, you can circumvent the IPL and use the service element (SE) to copy the Linux kernel to the mainframe memory (see "Loading Linux from a DVD or from an FTP server" on page 308).

Apart from starting a boot process, an IPL can also be used for:

- Writing out system storage (dumping)
  
  Refer to Linux on System z 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/documentation_dev.html
The `zipl` tool allows you to prepare DASD, SCSI, and tape devices as IPL devices for booting Linux, for dumping, or for loading a DCSS. See Chapter 33, “Initial program loader for System z - zipl,” on page 275 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 48 provides an overview of the possibilities:

<table>
<thead>
<tr>
<th></th>
<th>DASD</th>
<th>tape</th>
<th>SCSI</th>
<th>NSS</th>
<th>VM reader</th>
<th>CD-ROM/FTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM guest</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LPAR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

DASDs, tapes on channel-attached tape devices, and SCSI device that are attached through an FCP channel can be used for both LPAR and VM guests. A SCSI device can be a disk or an FC-attached CD-ROM or DVD drive. Named saved systems (NSS) and the VM reader are 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

In SUSE Linux Enterprise Server, you use `zipl` to prepare a DASD or SCSI boot disk. You can also define a menu configuration. A boot device with a menu configuration can hold the code for multiple boot configurations. For SCSI disks, the menu can also include one or more SCSI system dumpers.

Each boot and dump configuration in a menu is associated with a configuration number. At IPL time, you can specify a configuration number to select the configuration to be used.

For menu configurations on DASD, you can display a menu with the configuration numbers (see “Example for a DASD menu configuration on VM” on page 302 and “Example for a DASD menu configuration (LPAR)” on page 308). For menu configurations on SCSI disks, you need to know the configuration numbers without being able to display the menus.

See “Menu configurations” on page 292 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

On SUSE Linux Enterprise Server, kernel images are installed into the /boot directory and are named image-<version>. See SUSE Linux Enterprise Server 11 Deployment Guide for information about where to find the iamges and how to start an installation.

Boot loader code

SUSE Linux Enterprise Server kernel images are compiled to contain boot loader code for IPL from VM reader devices.

If you want to boot a kernel image from a device that does not correspond to the included boot loader code, you can provide alternate boot loader code separate from the kernel image.

Use zipl to prepare boot devices with separate DASD, SCSI, or tape boot loader code. You can then boot from DASD, SCSI, or tape regardless of the boot loader code in the kernel image.

Kernel parameters

The kernel parameters are in form of an ASCII text string of up to 895 characters. If the boot device is tape or the 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.

Using the zipl command to prepare your boot device, you can provide kernel parameters on the command line, in a parameter file, and in a zipl configuration file. See Chapter 33, “Initial program loader for System z - zipl,” on page 275 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.

You can use the following kernel parameters to set up the console:

**conmode=<mode> and condev=<cuu>**

See “Console kernel parameter syntax” on page 264 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.

SUSE Linux Enterprise Server provides a ramdisk located in /boot and named initrd-<kernel version>. When a ramdisk is installed or modified, you must call zipl to update the boot record.

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 Getting Started with Linux on System z, SC24-6096.

Using tape

Before you start:

- You need a tape that is prepared as a boot device.

A tape boot device must contain the following in the specified order:
1. Tape boot loader code
   The tape boot loader code is included in the s390-tools package on developerWorks.
2. Tape mark
3. Kernel image
4. Tape mark
5. Kernel parameters (optional)
6. Tape mark
7. Initial RAM disk (optional)
8. Tape mark
9. Tape mark

All tape marks are required even if an optional item is omitted. For example, if you do not provide an initial RAM disk image, the end of the boot information is marked with three consecutive tape marks. zipl prepared tapes conform to this layout.

Perform these steps to start the boot process:

1. Establish a CMS 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> parm <kernel_parameters>`

where

<devno>
is the device number of the boot device as seen by the guest.
**parm** `<kernel_parameters>`

is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters used by your boot configuration (see ["Preparing a boot device" on page 279](#) for information about the boot configuration). To replace the existing kernel parameters, use:

```
parm =<kernel_parameters>
```

See also ["Using a VM parameter string for Linux re-IPL" on page 313](#).

**Using DASD**

**Before you start:**

- You need a DASD boot device prepared with **zipl** (see ["Preparing a boot device" on page 279](#)).

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> parm <kernel_parameters>
```

where:

- `<devno>` specifies the device number of the boot device as seen by the guest.

**loadparm** `<n>`

is applicable to menu configurations only. Omit this parameter if you are not working with a menu configuration.

Configuration number "0" specifies the default configuration. Depending on the menu configuration, omitting this option might display the menu or select the default configuration. Specifying "prompt" instead of a configuration number forces the menu to be displayed.

Displaying the menu allows you to specify additional kernel parameters (see ["Example for a DASD menu configuration on VM" on page 302](#)). 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 292](#) for more details on menu configurations.

**parm** `<kernel_parameters>`

is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters used by your boot configuration (see ["Preparing a boot device" on page 279](#) for information about the boot configuration). To replace the existing kernel parameters, use:

```
parm =<kernel_parameters>
```

See also ["Using a VM parameter string for Linux re-IPL" on page 313](#).
Example for a DASD menu configuration on VM

This example illustrates how menu2 in the sample configuration file in Figure 54 on page 294 displays on the VM console:

```
00: ziptl v1.8.0 interactive boot menu
00: 0. default (boot1)
00: 1. boot1
00: 2. boot3
00: Note: VM users please use '#cp vi vmsg <input>'
00: Please choose (default will boot in 30 seconds):
```

You choose a configuration by specifying its configuration number. For example, to boot configuration boot3, issue:

```
#cp vi vmsg 2
```

You can also specify additional kernel parameters by appending them to this command. For example:

```
#cp vi vmsg 2 maxcpus=1 mem=64m
```

Using a SCSI device

A SCSI device can be a disk or an FC-attached CD-ROM or DVD drive.

**Before you start:**

- If you use a SCSI disk, you must prepare it with `zipl` (see "Preparing a boot device" on page 279).

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 eight from the final eight digits.

- `<lun>` is the LUN of the SCSI boot disk. Specify the LUN in hexadecimal format with a blank separating the first eight from the final eight 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. For example, an installation from DVD is typically done with boot program selector 2.

See “Menu configurations” on page 292 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 a named saved system**

**Before you start:**

The z/VM guest virtual machine that is to boot Linux from the NSS must run on a z/VM system on which a Linux kernel with kernel sharing support has been saved to an NSS.

To boot your z/VM guest from an NSS, `<nss_name>`, enter an IPL command of this form:

```
#cp i <nss_name> parm <kernel_parameters>
```

where:

- **<nss_name>**
  
  The NSS name can be one to eight characters long and must consist of alphabetic or numeric characters. Examples of valid names include: 73248734, NSSCSITE, or NSS1234.

- **parm <kernel_parameters>**

  is an optional 56-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters used by your boot configuration (see “Preparing a boot device” on page 279 for information about the boot configuration). To replace the existing kernel parameters, use:

  ```
  parm =<kernel_parameters>
  ```

See also “Using a VM parameter string for Linux re-IPL” on page 313.
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](http://ibm.com/redbooks/redpapers/pdfs/redp0120.pdf).

Tip: On the SUSE Linux Enterprise Server DVD under /boot/s390x there is a sample script (REXX™ EXEC) for booting from the VM reader.

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. This is the case for the default SUSE Linux Enterprise Server kernel.
- 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:
      
      \[ \text{spool pun * rdr} \]

   c. Use the CMS PUNCH command to transfer each of the required files to the reader. Be sure to use the "no header" option to omit the file headers.
      First transfer the kernel image.
      Second transfer the kernel parameters.
      Third transfer the initial RAM disk image, if present.

      For each file, issue a command of this form:
      
      \[ \text{pun <file_name> <file_type> <file_mode> (noh)} \]

   d. Optionally, ensure that the contents of the reader remain fixed.
      
      \[ \text{change rdr all keep nohold} \]

      If you omit this step, all files are deleted from the reader during the IPL that follows.
4. Issue the IPL command:
   
   \[ \text{ipl 000c clear parm <kernel_parameters>} \]

   where:
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 279).
- For booting from a SCSI boot device, you need to have the SCSI IPL feature (FC9904) installed.

Perform these steps to boot from a DASD, tape, or SCSI boot device:
1. In the left navigation pane of the HMC expand Systems Management and Servers and select the mainframe system you want to work with. A table of LPARs is displayed in the upper content area on the right.
2. Select the LPAR where you want to boot Linux.
3. In the Tasks area, expand Recovery and click Load (see Figure 56 on page 306).
4. Proceed according to your boot device.

   For booting from tape:
   a. Select **Load type** “Normal” (see Figure 57).

   b. Enter the device number of the tape boot device in the **Load address** field.

   For booting from DASD:
   a. Select **Load type** “Normal” (see Figure 57).
b. Enter the device number of the DASD boot device in the **Load address** field.

c. If the boot configuration is part of a zipl created menu configuration, enter the configuration number that identifies your DASD boot configuration within the menu in the **Load parameter** field.

Configuration number "0" specifies the default configuration. Depending on the menu configuration, omitting this option might display the menu or select the default configuration. Specifying "prompt" instead of a configuration number forces the menu to be displayed.

Displaying the menu allows you to specify additional kernel parameters (see "Example for a DASD menu configuration (LPAR)" on page 308). 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 292 for more details on menu configurations.

**For booting from a SCSI device:**

A SCSI device can be a disk or an FC-attached CD-ROM or DVD drive.

a. Select **Load type** "SCSI" (see Figure 58).

b. Enter the device number of the FCP channel through which the SCSI device is accessed in the **Load address** field.

c. Enter the WWPN of the SCSI device in the **World wide port name** field.

d. Enter the LUN of the SCSI device in the **Logical unit number** field.

e. If the boot configuration is part of a zipl created menu configuration, enter the configuration number that identifies your SCSI boot configuration within the menu in the **Boot program selector** field. Configuration number "0" specifies the default configuration. For example, an installation from DVD is typically done with boot program selector 2.
See “Menu configurations” on page 292 for more details on menu configurations.

f. Leave the fields Load parameter, Boot record logical block address, and Operating System specific load parameters blank.

5. Click OK to start the boot process.

Check the output on the preferred console (see “Console kernel parameter syntax” on page 264) to monitor the boot progress.

Example for a DASD menu configuration (LPAR)

This example illustrates how menu2 in the sample configuration file in Figure 54 on page 294 displays on the hardware console:

```
ziPL v1.3.0 interactive boot menu
0. default (boot1)
1. boot1
2. boot3

Please choose (default will boot in 30 seconds):
```

You choose a configuration by specifying the configuration number. For example, to boot configuration boot3, issue:

```
# 2
```

You can also specify additional kernel parameters by appending them to this command. For example:

```
# 2 maxcpus=1 mem=64m
```

Loading Linux from a DVD or from an FTP server

You can use the SE to copy the Linux kernel image directly to your LPARs memory. This process bypasses IPL and does not require a boot loader. The SE performs the tasks that are normally done by the boot loader code. When the Linux kernel has been loaded, Linux is started using restart PSW.

As a source, you can use the SE's CD-ROM/DVD drive or any device on a remote system that you can access through FTP from your SE. If you access the SE remotely from an HMC, you can also use the CD-ROM drive of the system where your HMC runs.

The installation process requires a file with a mapping of the location of installation data in the file system of the DVD or FTP server and the memory locations where the data is to be copied. For SUSE Linux Enterprise Server this file is called suse.ins and located in the root directory of the file system on the DVD 1.

1. In the left navigation pane of the HMC expand Systems Management and Servers and select the mainframe system you want to work with. A table of LPARs is displayed in the upper content area on the right.
2. Select the LPAR where you want to boot Linux.
3. In the Tasks area, expand Recovery and click Load from CD-ROM, DVD, or Server (see Figure 59 on page 309).
4. Specify the source of the code to be loaded.

   For loading from a CD-ROM drive:
   a. Select **Hardware Management Console CD-ROM/DVD** (see Figure 60).

   ![Figure 60. Load from CD-ROM or Server panel](image)

   **Load from CD-ROM or Server - H42:H42LP05**

   Use this task to load operating system software or utility programs from a CD-ROM/DVD or a server that can be accessed using FTP.

   Select the source of the software:
   - Hardware Management Console CD-ROM/DVD
   - FTP Source
   
   Insert host computer:

   User ID:

   Password:

   Account (optional):

   File location (optional):

   ![Figure 60. Load from CD-ROM or Server panel](image)

   b. Leave the **File location** field blank.

   For loading from an FTP server:
   a. Select the **FTP Source** radio button.
b. Enter the IP address or host name of the FTP server where the install code resides in the **Host computer** entry field.

c. Enter your user ID for the FTP server in the **User ID** entry field.

d. Enter your password for the FTP server in the **Password** entry field.

e. If required by your FTP server, enter your account information in the **Account** entry field.

f. Enter the path for the directory where the suse.ins resides in the file location entry field. You can leave this field blank if the file resides in the FTP server’s root directory.

5. Click **Continue** to display the “Select the software to load” panel (Figure 61).

6. Select the **suse.ins**.

7. Click **OK** to start loading Linux.

At this point the kernel has started and the SUSE Linux Enterprise Server boot process continues.

### Displaying current IPL parameters

To display the IPL parameters, use the command `lsreipl` (see **lsreipl - List IPL settings** on page 376). Alternatively, a sysfs user-space interface is available:

```
/sys/firmware/ipl/ipl_type
```

The `/sys/firmware/ipl/ipl_type` ASCII file contains the device type from which the kernel was booted. The following values are possible:

- **ccw**: The IPL device is a CCW device.
- **fcp**: The IPL device is an FCP device.
- **unknown**: The IPL device is not known.

Depending on the IPL type, additional files might reside in `/sys/firmware/ipl/`. If the device is CCW, the additional files `device` and `loadparm` are present.

- **device**: Contains the bus ID of the CCW device used for IPL, for example:

```
# cat /sys/firmware/ipl/device
0.0.1234
```

- **loadparm**: Contains the eight-character loadparm used for IPL, for example:
# cat /sys/firmware/ipl/loadparm

parm
Contains the current VM parameter string:

# cat /sys/firmware/ipl/parm
init=/bin/bash

See also "Using a VM parameter string for Linux re-IPL" on page 313.

If the device is FCP, a number of additional files are present (also see Chapter 4, "SCSI-over-Fibre Channel device driver," on page 41 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.

---

Re-booting from an alternative source

When you re-boot Linux, the system conventionally boots from the last used location. However, you can configure an alternative device to be used for re-IPL instead of the last used IPL device. When the system is re-IPLed, the alternative device is used to boot the kernel.

The SUSE Linux Enterprise Server kernel includes this feature.

Prerequisites

You need sysfs to use this feature; SUSE Linux Enterprise Server mounts sysfs by default.

The machine must have zfcp IPL support for re-IPL from SCSI devices. This support is available only for LPARs on System z machines.
Configuring the re-IPL device

To configure the re-IPL device, use the `chreipl` tool (see “chreipl - Modify the re-IPL configuration” on page 338).

Alternatively, you can use a sysfs interface. The virtual configuration files are located under `/sys/firmware/reipl`. To configure, write strings into the configuration files. The following re-IPL types can be set with the `/sys/firmware/reipl/reipl_type` attribute:

- ccw: For ccw devices such as ESCON- or FICON-attached DASDs.
- fcp: For FCP SCSI devices, including SCSI disks and CD or DVD drives (Hardware support is required.)
- nss: For Named Saved Systems (z/VM only)

For each supported re-IPL type a sysfs directory is created under `/sys/firmware/reipl` that contains the configuration attributes for the device. The directory name is the same as the name of the re-IPL type.

When Linux is booted, the re-IPL attributes are set by default to the values of the boot device, which can be found under `/sys/firmware/ipl`.

Attributes for ccw

The attributes for re-IPL type ccw under `/sys/firmware/reipl/ccw` are:

- device: Device number of the re-IPL device. For example 0.0.4711.
- loadparm: An eight-character loadparm used to select the boot configuration in the zipl menu (if available). The loadparm parameter can be set only under z/VM.
- parm: A 64-byte string containing kernel parameters that is concatenated to the boot command line. See also “Using a VM parameter string for Linux re-IPL” on page 313.

Attributes for fcp

The attributes for re-IPL type fcp under `/sys/firmware/reipl/fcp` are:

- device: Device number of the fcp adapter used for re-IPL. For example 0.0.4711.
- wwpn: World wide port number of the FCP re-IPL device.
- lun: Logical unit number of the FCP re-IPL device.
- bootprog: Boot program selector. Used to select the boot configuration in the zipl menu (if available).
- br_lba: Boot record logical block address. Master boot record. Is always 0 for Linux.

Attributes for nss

The attributes for re-IPL type nss under `/sys/firmware/reipl/nss` are:

- name: Name of the NSS. The NSS name can be 1-8 characters long and must consist of alphabetic or numeric characters. Examples of valid names include: 73248734, NSSCSITE, or NSS1234.
- parm: A 56-byte string containing kernel parameters that is concatenated to the boot command line. (Note the difference in length compared to ccw.) See also “Using a VM parameter string for Linux re-IPL” on page 313.

Kernel panic settings

Set the attribute `/sys/firmware/shutdown_actions/on_panic` to reipl to make the system re-IPL with the current re-IPL settings in case of a kernel panic. See also the `dumpconf` tool described in Linux on System z Using the Dump Tools on the developerWorks Web site at:
Examples

To configure an FCP re-IPL device 0.0.4711 with a LUN 0x4711000000000000 and a WWPN 0x5005076303004711:

```bash
# echo 0.0.4711 > /sys/firmware/reipl/fcp/device
# echo 0x5005076303004711 > /sys/firmware/reipl/fcp/wwpn
# echo 0x4711000000000000 > /sys/firmware/reipl/fcp/lun
# echo 0 > /sys/firmware/reipl/fcp/bootprog
# echo 0 > /sys/firmware/reipl/fcp/br_lba
# echo fcp > /sys/firmware/reipl/reipl_type
```

**Note:** IPL is possible only from subchannel set 0.

To re-IPL a Linux NSS with different parameters:

1. Change to the reipl sysfs directory:
   ```bash
   # cd /sys/firmware/reipl/
   ```
2. Set the reipl_type to nss:
   ```bash
   # echo nss > reipl_type
   ```
3. Setup the attributes in the nss directory:
   ```bash
   # echo LNXNSS > name
   # echo "dasd=0150 root=/dev/dasda1" > parm
   ```

To set a VM parameter string for Linux re-IPL, follow these steps:

1. Change to the sysfs directory appropriate for the next re-IPL:
   ```bash
   # cd /sys/firmware/reipl/$(cat /sys/firmware/reipl/reipl_type)
   ```
2. Use the echo command to output the parameter string into the parmfile:
   ```bash
   # echo "init=/bin/bash" > parm
   ```

---

**Using a VM parameter string for Linux re-IPL**

**Before you begin:** Your system must have either:

- z/VM 5.3 with PTFs for APAR VM64402 and APAR VM64442
- z/VM 5.4

The z/VM IPL process allows to specify a parameter string up to 64 bytes long that is passed to the guest virtual machine. The parameter string is stored in the general purpose registers at completion of IPL. After IPL, the kernel retrieves the parameter string and appends the string to the boot command line.

Later, the boot command line is parsed for specific kernel options to configure various kernel components. The VM parameter string helps you to control the Linux start and initialization behavior in a flexible way. You can:

- Add new kernel options using the IPL command and without modifying the content of the zipl parameter file, see "Booting a z/VM Linux guest" on page 300.
- Temporarily overwrite the kernel options that are provided by zipl. (See "How kernel parameters from different sources are combined" on page 280.)
- Replace the entire boot command line with the VM parameter string; thereby bypassing any kernel options from the zipl parameter file.
- Create new Linux Named Saved Systems (NSS) on the CP/CMS command line, see "Creating a Linux NSS from the CP command line" on page 210.

Further information
See z/VM CP Commands and Utilities Reference, SC24-6081, for information about how VM parameters can be specified using the IPL command.
Chapter 35. Shutdown actions

You can specify the action to take on shutdown by setting the shutdown actions attributes. The /sys/firmware/ directory has the following structure:

![Firmware directory structure diagram]

The directories contain the following information:

- **ipl**: Information about the IPL device (see “Displaying current IPL parameters” on page 310).
- **reipl**: Information about the re-ipl device (see “Re-booting from an alternative source” on page 311).
- **dump**: Information about the dump device. Attributes are configured by the dumpconf script. For details, see the description of the dumpconf command in Using the Dump Tools, SC33-8412.
- **vmcmd**: CP commands for halt, power off, reboot, and panic.
- **shutdown_actions**: Configuration of actions in case of halt, poff, reboot and panic.

The shutdown_actions directory contains the following files:

- on_halt
- on_poff
- on_reboot
- on_panic

The shutdown_actions attributes can contain the shutdown actions 'ipl', 'reipl', 'dump', 'stop' or 'vmcmd'. These values specify what should be done in case of a
halt, power off, reboot or kernel panic event. Default for on_halt, on_poff and on_panic is 'stop'. Default for reboot is 'reipl'. The attributes can be set by writing the appropriate string into the virtual files.

The vmcmd directory also contains the four files on_halt, on_poff, on_reboot, and on_panic. All these files can contain CP commands.

For example, if CP commands should be executed in case of a halt, the on_halt attribute in the vmcmd directory must contain the CP commands and the on_halt attribute in the shutdown_actions directory must contain the string 'vmcmd'.

CP commands written to the vmcmd attributes must be uppercase. You can specify multiple commands using the newline character "\n" as separator. The maximum command line length is limited to 127 characters.

For CP commands that do not end or stop the virtual machine, halt, power off, and panic will stop the machine after the command execution. For reboot, the system will be rebooted using the parameters specified under /sys/firmware/reipl.

Examples

If the Linux halt command is executed, automatically log off the z/VM guest:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_halt
# echo LOGOFF > /sys/firmware/vmcmd/on_halt
```

If the Linux poweroff command is executed, send a message to guest MASTER and automatically log off the guest. Do not forget the cat command to ensure that the newline is processed correctly:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_poff
# echo -e "MSG MASTER Going down\nLOGOFF" | cat > /sys/firmware/vmcmd/on_poff
```

If a kernel panic occurs, trigger a re-ipl using the IPL parameters under /sys/firmware/ipl:

```
# echo ipl > /sys/firmware/shutdown_actions/on_panic
```

If the Linux reboot command is executed, send a message to guest MASTER and reboot Linux:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_reboot
# echo "MSG MASTER Reboot system" > /sys/firmware/vmcmd/on_reboot
```

Note that VM commands, device addresses, and guest names must be uppercase.
Part 8. Diagnostics and troubleshooting

This section describes device drivers and features that are used in the context of diagnostics and problem solving.

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

Restrictions: For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 release notes at
www.novell.com/linux/releasenotes/s390x/SUSE-SLES/11/

---

Chapter 36. Channel measurement facility ............................................. 319
Features ............................................................................................. 319
Setting up the channel measurement facility ...................................... 319
Working with the channel measurement facility ............................... 320

Chapter 37. Control program identification ............................................. 323
Working with the CPI support .............................................................. 323

Chapter 38. Avoiding common pitfalls .................................................. 327
Ensuring correct channel path status ................................................... 327
Determining channel path usage ......................................................... 327
Configuring LPAR I/O devices ............................................................ 327
Using cio_ignore ............................................................................... 328
Excessive guest swapping ................................................................. 328
Booting stops with disabled wait state ............................................... 328
Preparing a dump disk .................................................................... 328

Chapter 39. Kernel messages ............................................................... 329
Displaying a message man page ........................................................ 329
Chapter 36. Channel measurement facility

The System z architecture provides a channel measurement facility to collect statistical data about I/O on the channel subsystem. Data collection can be enabled for all CCW devices. User space applications can access this data through the sysfs.

Features

The channel measurement facility provides the following features:

- Basic channel measurement format for concurrently collecting data on up to 4096 devices. (Note that specifying 4096 or more channels causes high memory consumption and enabling data collection might not succeed.)
- Extended channel measurement format for concurrently collecting data on an unlimited number of devices.
- Data collection for all channel-attached devices, except those using QDIO (that is, except qeth and SCSI-over-Fibre channel attached devices)

Setting up the channel measurement facility

You can configure the channel measurement facility by adding parameters to the kernel parameter file.

Channel measurement facility kernel parameters

![Channel measurement facility kernel parameters diagram]

Note: If you specify both parameter=value pairs, separate them with a blank.

where:

cmf.format

defines the format, “0” for basic and “1” for extended, of the channel measurement blocks. The default, “-1”, assigns a format depending on the hardware. For System z9 and System z10 mainframes the extended format is used.

cmf.maxchannels=<no_channels>

limits the number of devices for which data measurement can be enabled concurrently with the basic format. The maximum for <no_channels> is 4096. A warning will be printed if more than 4096 channels are specified. The channel measurement facility might still work; however, specifying more than 4096 channels causes a high memory consumption.

For the extended format there is no limit and any value you specify is ignored.
Working with the channel measurement facility

This section describes typical tasks you need to perform when working with the channel measurement facility.

- Enabling, resetting, and switching off data collection
- Reading data

Enabling, resetting, and switching off data collection

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:

```bash
# echo 1 > /sys/bus/ccw/devices/<device_bus_id>/cmb_enable
```

where `/sys/bus/ccw/devices/<device_bus_id>` represents the device in sysfs.

When data collection is enabled for a device, a subdirectory `/sys/bus/ccw/devices/<device_bus_id>/cmf` is created that contains several attributes. These attributes contain the collected data (see "Reading data").

To switch off data collection issue a command of this form:

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

When data collection for a device is switched off, the subdirectory `/sys/bus/ccw/devices/<device_bus_id>/cmf` and its content are deleted.

Example

In this example, data collection for a device `/sys/bus/ccw/devices/0.0.b100` is already active and reset:

```bash
# cat /sys/bus/ccw/devices/0.0.b100/cmb_enable
1
# echo 1 > /sys/bus/ccw/devices/0.0.b100/cmb_enable
```

Reading data

While data collection is enabled for a device, the directories that represent it in sysfs contain a subdirectory, cmf, with several read-only attributes. These attributes hold the collected data. To read one of the attributes issue a command of this form:

```bash
# cat /sys/bus/ccw/devices/<device_bus-id>/cmf/<attribute>
```

where `/sys/bus/ccw/devices/<device-bus-id>` is the directory that represents the device, and `<attribute>` the attribute to be read. Table 49 on page 321 summarizes the available attributes.
Table 49. Attributes with collected I/O data

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssch_rsch_count</td>
<td>An integer representing the ssch rsch count value.</td>
</tr>
<tr>
<td>sample_count</td>
<td>An integer representing the sample count value.</td>
</tr>
<tr>
<td>avg_device_connect_time</td>
<td>An integer representing the average device connect time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_function_pending_time</td>
<td>An integer representing the average function pending time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_device_disconnect_time</td>
<td>An integer representing the average device disconnect time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_control_unit_queuing_time</td>
<td>An integer representing the average control unit queuing time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_initial_command_response_time</td>
<td>An integer representing the average initial command response time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_device_active_only_time</td>
<td>An integer representing the average device active only time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_device_busy_time</td>
<td>An integer representing the average value device busy time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_utilization</td>
<td>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.</td>
</tr>
<tr>
<td>avg_sample_interval</td>
<td>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.</td>
</tr>
<tr>
<td>avg_initial_command_response_time</td>
<td>An integer representing the average time in nanoseconds between the first command of a channel program being sent to the device and the command being accepted. Available in extended format only.</td>
</tr>
<tr>
<td>avg_device_busy_time</td>
<td>An integer representing the average time in nanoseconds of the subchannel being in the “device busy” state when initiating a start or resume function. Available in extended format only.</td>
</tr>
</tbody>
</table>

Example
To read the avg_device_busy_time attribute for a device /sys/bus/ccw/devices/0.0.b100:

```
# cat /sys/bus/ccw/devices/0.0.b100/cmf/avg_device_busy_time
21
```
Chapter 37. 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, or the sysfs interface /sys/firmware/cpi to assign names to your Linux instance and sysplex. The names are used, for example, to identify the Linux instance or the sysplex on the HMC.

Working with the CPI support

This section describes typical tasks that you need to perform when working with CPI support.

- [Loading the CPI module](#)
- "Defining a sysplex name" on page 324
- "Defining a system name"
- "Defining a system type" on page 324
- "Displaying the system level" on page 324
- "Sending system data to the SE" on page 324

Loading the CPI module

You can provide the system name and the sysplex name as parameters when you load the CPI module, but the preferred method is to use the sysfs interface. When loading the CPI module the following is sent to the SE:

- System name (if provided)
- Sysplex name (if provided)
- System type (automatically set to "LINUX")
- System level (automatically set to the value of LINUX_VERSION_CODE)

CPI module parameter syntax

```
modprobe sclp_cpi
```

where:

- **system_name** = `<system>`
  - specifies an 8-character system name of the following set: A-Z, 0-9, $, @, # and blank. The specification is converted to uppercase.

- **sysplex_name** = `<sysplex>`
  - specifies an 8-character sysplex name of the following set: A-Z, 0-9, $, @, # and blank. The specification is converted to uppercase.

Defining a system name

Use the attribute system_name in sysfs to specify a system name:

```
/sys/firmware/cpi/system_name
```

The system name is a string consisting of up to 8 characters of the following set: A-Z, 0-9, $, @, # and blank.
Defining a sysplex name

Use the attribute sysplex_name in sysfs to specify a sysplex name:
/sys/firmware/cpi/sysplex_name

The sysplex name is a string consisting of up to 8 characters of the following set: A-Z, 0-9, $, @, # and blank.

Example:

```bash
# echo SYSPLEX1 > /sys/firmware/cpi/sysplex_name
```

This attribute is intended for setting the name only. To confirm the current sysplex name, check the HMC.

Defining a system type

Use the attribute system_type in sysfs to specify a system type. The system type must be LINUX.
/sys/firmware/cpi/system_type

Example:

```bash
# echo LINUX > /sys/firmware/cpi/system_type
```

Displaying the system level

The attribute system_level in sysfs provides the operating system version:
/sys/firmware/cpi/system_level

The information is displayed in the format:
0x0000000000aabbcc

where:

- **aa** kernel version
- **bb** kernel patch level
- **cc** kernel sublevel

Example: Linux kernel 2.6.27 displays as

```bash
# cat /sys/firmware/cpi/system_level
0x000000000002061b
```

Sending system data to the SE

Use the attribute set in sysfs to send data to the service element:
/sys/firmware/cpi/set
To send the data in attributes sysplex_name, system_level, system_name, and, system_type to the SE, write an arbitrary string to the set attribute.

Example:

```bash
# echo 1 > /sys/firmware/cpi/set
```
Chapter 38. 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:

```bash
echo off > /sys/devices/css0/chp0.<chpid>/status
```

After the operation has finished and the path is available again, vary the path online using:

```bash
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:

```bash
echo off > /sys/devices/css0/chp0.<chpid>/status
echo on > /sys/devices/css0/chp0.<chpid>/status.
```

Rationale: Linux does not always receive a notification (machine check) when the status of a path changes (especially a path becoming online again). To make sure Linux has up-to-date information on the usable paths, path verification is triggered through the Linux vary operation.

Determining channel path usage

To determine the usage of a specific channel path on LPAR, for example, to check whether traffic is distributed evenly over all channel paths, use the channel path measurement facility. See "Channel path measurement" on page 13 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 435 and "Changing the list of devices to be ignored" on page 436.

Rationale: Numerous unused devices can cause:

- Unnecessary high memory usage due to device structures being allocated.
• Unnecessary high load on status changes, since hot-plug handling must be done for every device found.

Using cio_ignore

With cio_ignore, essential devices might have been hidden. For example, if Linux does not boot under z/VM and does not show any message except

```
HCPGIR450W CP entered; disabled wait PSW 00020001 00000000 00000000 00144D7A
```

check if cio_ignore is used and verify that the console device, which is typically device number 0.0.0009, is not ignored.

Excessive guest swapping

If a Linux guest seems to be swapping and not making any progress, you might try to set the timed page pool size to zero:

```
# echo 0 > /proc/sys/vm/cmm_timed_pages
```

If you see a temporary relief, the guest does not have enough memory. Try increasing the guest memory.

If the problem persists, z/VM might be out of memory.

If the Linux guest is critical to your business, unload the cooperative memory management module:

```
# rmmod cmm
```

See Chapter 24, “Cooperative memory management,” on page 221 for more details on CMM.

Booting stops with disabled wait state

On some distributions, a processor type check is automatically run at every kernel start up. If the check determines that the distribution used is not compatible with the hardware, it stops the boot process with a disabled wait PSW with an address of zero.

If this happens, ensure that you are running SUSE Linux Enterprise Server 11 on supported hardware. See the SUSE Linux Enterprise Server 11 release notes at www.novell.com/linux/releasenotes/s390x/SUSE-SLES/11/.

Preparing a dump disk

You might want to consider setting up your system to automatically create a dump after a kernel panic. Configuring and using “dump on panic” has the following advantages:

• You have a dump disk prepared ahead of time.
• You do not have to reproduce the problem since a dump will be triggered automatically right after the failure.

Chapter 39. Kernel messages

System z specific kernel modules issue messages on the console and write them to the syslog. SUSE Linux Enterprise Server issues these messages with message numbers. Based on these message numbers, you can display man pages to obtain message details.

The message numbers consist of a module identifier, a dot, and six hexadecimal digits. For example, xpram.ab9aa4 is a message number.

Displaying a message man page

Before you start: Ensure that the RPM with the message man pages is installed on your Linux system. This RPM is called kernel-default-man-<kernel-version>.s390x.rpm and shipped on DVD1.

System z specific kernel messages have a message identifier. For example, the following message has the message identifier xpram.ab9aa4:

xpram.ab9aa4: 50 is not a valid number of XPRAM devices

Enter a command of this form, to display a message man page:

man <message_identifier>

Example: Enter the following command to display the man page for message xpram.ab9aa4:

# man xpram.ab9aa4

The corresponding man page looks like this:

xpram.ab9aa4(9) xpram.ab9aa4(9)
Message
xpram.ab9aa4: 50 is not a valid number of XPRAM devices
Severity
Error
Parameters
@1: number of partitions
Description
The number of XPRAM partitions specified for the 'devs' module parameter or with the 'xpram.parts' kernel parameter must be an integer in the range 1 to 32. The XPRAM device driver created a maximum of 32 partitions that are probably not configured as intended.
User action
If the XPRAM device driver has been compiled as a separate module, unload the module and load it again with a correct value for the 'devs' module parameter. If the XPRAM device driver has been compiled into the kernel, correct the 'xpram.parts' parameter in the kernel parameter line and restart Linux.

LINUX Linux Messages xpram.ab9aa4(9)
This section describes commands, kernel parameters, kernel options, and Linux use of z/VM DIAG calls.

**Newest version:** You can find the newest version of this book at

**Restrictions:** For prerequisites and restrictions see the System z architecture specific information in the SUSE Linux Enterprise Server 11 release notes at

---

**Chapter 40. Useful Linux commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>chccwdev</td>
<td>334</td>
</tr>
<tr>
<td>chchp</td>
<td>336</td>
</tr>
<tr>
<td>chreipl</td>
<td>338</td>
</tr>
<tr>
<td>chshut</td>
<td>340</td>
</tr>
<tr>
<td>chzcrypt</td>
<td>342</td>
</tr>
<tr>
<td>cpuplugd</td>
<td>344</td>
</tr>
<tr>
<td>dasdfmt</td>
<td>347</td>
</tr>
<tr>
<td>dasdview</td>
<td>350</td>
</tr>
<tr>
<td>fdasd</td>
<td>359</td>
</tr>
<tr>
<td>lschp</td>
<td>367</td>
</tr>
<tr>
<td>lsess</td>
<td>369</td>
</tr>
<tr>
<td>lsdsd</td>
<td>371</td>
</tr>
<tr>
<td>lslluns</td>
<td>373</td>
</tr>
<tr>
<td>lsqeth</td>
<td>374</td>
</tr>
<tr>
<td>lsreipl</td>
<td>376</td>
</tr>
<tr>
<td>lsshut</td>
<td>377</td>
</tr>
<tr>
<td>lstape</td>
<td>378</td>
</tr>
<tr>
<td>lszcrypt</td>
<td>381</td>
</tr>
<tr>
<td>lszfcp</td>
<td>383</td>
</tr>
<tr>
<td>mon_fsstatd</td>
<td>385</td>
</tr>
<tr>
<td>mon_procd</td>
<td>390</td>
</tr>
<tr>
<td>qetharp</td>
<td>398</td>
</tr>
<tr>
<td>qethconf</td>
<td>400</td>
</tr>
<tr>
<td>scsi_logging_level</td>
<td>403</td>
</tr>
<tr>
<td>snipl</td>
<td>406</td>
</tr>
<tr>
<td>tape390_crypt</td>
<td>415</td>
</tr>
<tr>
<td>tape390_display</td>
<td>419</td>
</tr>
<tr>
<td>tunedasd</td>
<td>421</td>
</tr>
<tr>
<td>vmcp</td>
<td>424</td>
</tr>
<tr>
<td>vmur</td>
<td>426</td>
</tr>
</tbody>
</table>

---

**Chapter 41. Selected kernel parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>additional_cpus</td>
<td>434</td>
</tr>
<tr>
<td>cio_ignore</td>
<td>435</td>
</tr>
<tr>
<td>maxcpus</td>
<td>438</td>
</tr>
<tr>
<td>mem</td>
<td>439</td>
</tr>
<tr>
<td>noinitrd</td>
<td>440</td>
</tr>
<tr>
<td>possible_cpus</td>
<td>441</td>
</tr>
<tr>
<td>ramdisk_size</td>
<td>442</td>
</tr>
</tbody>
</table>
Chapter 40. Useful Linux commands

This chapter describes commands to configure and work with the SUSE Linux Enterprise Server 11 for System z device drivers and features.

For the zipl command, see Chapter 33, “Initial program loader for System z - zipl,” on page 275.

snipl is provided as a separate package snipl-<version>.s390x.rpm. All other commands are included in the s390-tools RPM.

Some commands come with an init script or a configuration file or both. These files are installed under /etc/init.d/ and /etc/sysconfig/ respectively. You can extract any missing files from the etc subdirectory in the s390-tools package.

Note: For tools related to taking and analyzing system dumps, see Linux on System z 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:

```
<command> [other command options] [-h | --help | --version]
```

where command can be any of the commands described in this section.

See “Understanding syntax diagrams” on page ix for general information on reading syntax diagrams.
chccwdev

chccwdev - Set a CCW device online
This command is used to set CCW devices (See "Device categories" on page 7) online or offline.

Format

chccwdev syntax

```
chccwdev [-e] [-d] [-f] [-a <name>=<value>] <device_bus_id>
```

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

```
# chccwdev -d 0.0.b100,0.0.b200-0.0.b2ff

```

- To set several CCW devices in different ranges and different subchannel sets offline, issue:
```
# chccwdev -a online=0 0.0.1000-0.0.1100,0.1.7000-0.1.7010,0.0.1234,0.1.4321
```
chchp

chchp - Change channel path status

Use this command to set channel paths online or offline. The actions are equivalent to performing a Configure Channel Path Off or Configure Channel Path On operation on the hardware management console.

The channel path status that results from a configure operation is persistent across IPLs.

Note: Changing the configuration state of an I/O channel path might affect the availability of I/O devices as well as trigger associated functions (such as channel-path verification or device scanning) which in turn can result in a temporary increase in processor, memory and I/O load.

Format

chchp syntax

```
  chchp -c 0 0.<id> -0.<id> -v 0 1 -a <key>=<value>
```

Where:

- `-c` or `--configure <value>`
  sets the device to configured (1) or standby (0). Note that setting the configured state to standby may cause a currently running I/O operation to be aborted.

- `-v` or `--vary <value>`
  changes the logical channel-path state to online (1) or offline (0). Note that setting the logical state to offline may cause a currently running I/O operation to be aborted.

- `-a` or `--attribute <key>=<value>`
  changes the channel-path sysfs attribute `<key>` to `<value>`. The `<key>` can be the name of any available channel-path sysfs attribute (that is, "configure" or "status"), while `<value>` can take any valid value that can be written to the attribute (for example, "0" or "offline"). This is a more generic way of modifying the state of a channel-path through the sysfs interface. It is intended for cases where sysfs attributes or attribute values are available in the kernel but not in chchp.

- `0.<id>` and `0.<id> - 0.<id>`
  where `<id>` is a hexadecimal, two-digit, lower-case identifier for the channel path. An operation can be performed on more than one channel path by specifying multiple identifiers as a comma-separated list, or a range, or a combination of both.

- `--version`
  Prints the version number of chchp and exits.
Chchk

-h or --help
Prints out a short help text, then exits.

Examples

- To set channel path 0.19 into standby state issue:
  
  # chchp -a configure=0 0.19

- To set the channel path with the channel path ID 0.40 to the standby state, write "0" to the configure file using the chchp command:

  # chchp --configure 0 0.40
  Configure standby 0.40... done.

- To set a channel-path to the configured state, write "1" to the configure file using the chchp command:
  
  # chchp --configure 1 0.40
  Configure online 0.40... done.

- To set channel-paths 0.65 to 0.6f to the configured state issue:

  # chchp -c 1 0.65-0.6f

- To set channel-paths 0.12, 0.7f and 0.17 to 0.20 to the logical offline state issue:

  # chchp -v 0 0.12,0.7f,0.17-0.20
chreipl

**chreipl - Modify the re-IPL configuration**

Use the chreipl command to configure a disk or change an entry in the boot menu for the next boot cycle.

**Format**

```
chreipl syntax

chreipl

ccw -d <bus ID> -L <parameter>
node -d <DASD device node>

fcp -d <bus ID> -l <LUN> -w <wwpn> -b <n>
node -d <FCP device node>
```

Where:

- **ccw** selects a ccw device (DASD) for configuration.
- **fcp** selects an FCP device (device) for configuration.
- **node** specifies a boot target based on a device file.
- **-d** or **--device** *<bus ID>* specifies the device bus-ID of the re-IPL device.
- **<device node>**
  - the device node of the re-IPL device, either an FCP node (/dev/sd...) or a DASD node (/dev/dasd...).
- **-b** or **--bootprog** *<n>* specifies an optional configuration number that identifies the entry in the boot menu to use for the next reboot. The bootprog parameter only works in an FCP environment.
- **-L** or **--loadparm** *<parameter>* specifies an optional entry in the boot menu to use for the next reboot. This parameter must be an alphanumeric character or blank (""") and only works if a valid zipl boot menu is present. The loadparm parameter only works in a DASD environment.
- **-l** or **--lun** *<LUN>* logical unit number (LUN) of the FCP re-IPL device.
- **-w** or **--wwpn** *<wwpn>* world-wide port name (WWPN) of the FCP re-IPL device.
- **-h** or **--help** displays short information on command usage.
- **-v** or **--version** displays version information.
**Examples**

This section illustrates common uses for `chreipl`.

- To boot from device `/dev/dasda` at next system start:
  
  ```bash
  # chreipl node /dev/dasda
  ```

- To use `/dev/sda` as the boot device for the next boot:
  
  ```bash
  # chreipl node /dev/sda
  ```

- To use the ccw device with the number 0.0.7e78 for the next system start:
  
  ```bash
  # chreipl ccw 0.0.7e78
  ```

- After reboot, IPL from the ccw device with the number 0.0.7e78 using the first entry of the boot menu:
  
  ```bash
  # chreipl ccw -d 0.0.7e78 -L 1
  ```

- Re-IPL from the fcp device number 0.0.1700 using WWPN 0x500507630300c562 and LUN 0x401040B300000000
  
  ```bash
  # chreipl fcp --wwpn 0x500507630300c562 --lun 0x401040B300000000 -d 0.0.1700
  ```
chshut - Control the system behavior

The kernel configuration is controlled through entries below the /sys/firmware directory structure. Use the chshut command to change the entries pertaining to shutdown. Also see Chapter 35, “Shutdown actions,” on page 315 for more information on shutdown options.

The chshut command controls the system behavior in the following system states:
- Halt
- Power off
- Reboot

The chshut command handles up to three parameters. The first specifies the system state to which you want to change. The second argument specifies the action you want to execute in the previously specified system state. Valid arguments are ipl, reipl, stop and vmcmd.

If you have chosen vmcmd as action, a third parameter is used for the command to be executed inside z/VM.

Format

chshut syntax

```
chshut [halt | poff | reboot] [ipl | reipl | stop] [vmcmd <z/VM command>]
```

Where:
- **halt** specifies a system state of halt. In SUSE Linux Enterprise Server, by default, "halt" is mapped to "poff". This can be changed by editing the file /etc/sysconfig/shutdown and replacing HALT="auto" with HALT="halt".
- **poff** specifies a system state of power off.
- **reboot** specifies a system state of reboot.
- **ipl** sets the action to be taken to IPL.
- **reipl** sets the action to be taken to re-IPL.
- **stop** sets the action to be taken to stop.
- **vmcmd <z/VM command>** sets the action to be taken to run the specified z/VM command. The command must be in upper case. To issue several commands, repeat the vmcmd attribute with each command. The command string must be enclosed in quotation marks.
- **-h or --help** displays short information on command usage.
-v or --version
displays version information.

**Examples**

This section illustrates common uses for **chshut**.

- To make the system start again after a halt:
  ```
  # chshut halt ipl
  ```

- To log off the z/VM guest if the Linux halt command was executed successfully:
  ```
  # chshut halt vmcmd LOGOFF
  ```

- To send a message to guest MASTER and automatically log off the guest if the Linux power off command is executed:
  ```
  # chshut poff vmcmd "MSG MASTER Going down" vmcmd "LOGOFF"
  ```
chzcrypt

**chzcrypt - Modify the zcrypt configuration**

Use the `chzcrypt` command to configure cryptographic adapters managed by zcrypt and modify zcrypt’s AP bus attributes. To display the attributes, use `lkszcrypt -D Display zcrypt devices` on page 381.

**Before you start:**
- You must have root permissions.
- The sysfs file system must be mounted.

**Format**

```
chzcrypt syntax

```

Where:

- `-e` or `--enable`
  sets the given cryptographic adapters online.

- `-d` or `--disable`
  sets the given cryptographic adapters offline.

- `-a` or `--all`
  sets all available cryptographic adapters online or offline.

  `<device ID>`
  specifies a cryptographic adapter which will be set online or offline. A cryptographic adapter can be specified either in decimal notation or hexadecimal notation using a '0x' prefix.

- `-p` or `--poll-thread-enable`
  enables zcrypt’s poll thread.

- `-n` or `--poll-thread-disable`
  disables zcrypt’s poll thread.

- `-c <timeout>` or `--config-time <timeout>`
  sets configuration timer for re-scanning the AP bus to `<timeout>` seconds.

- `-t <time>` or `--poll-timeout=<time>`
  sets the high resolution polling timer to `<time>` nanoseconds. To display the value, use `lkszcrypt -b`.

- `-V` or `--verbose`
  prints verbose messages.

- `-h` or `--help`
  displays short information on command usage.
chzcrypt

-v or --version
  displays version information.

Examples

This section illustrates common uses for chzcrypt.

- To set the cryptographic adapters 0, 1, 4, 5, and 12 online (in decimal notation):
  
  \[ \text{chzcrypt -e 0 1 4 5 12} \]

- To set all available cryptographic adapters offline:

  \[ \text{chzcrypt -d -a} \]

- To set the configuration timer for re-scanning the AP bus to 60 seconds and disable zcrypt's poll thread:

  \[ \text{chzcrypt -c 60 -n} \]
Use the `cpuplugd` command to:

- Enable or disable CPUs based on a set of rules. This increases the performance of single threaded applications within a z/VM or LPAR environment with multiple CPUs. The rules can incorporate certain system load variables.
- Manage memory under z/VM.

**Before you start:**
- The sysfs file system must be mounted to /sys.
- The proc file system needs to be available at /proc.

**Format**

```
cpuplugd syntax
```

```
[[cpuplugd] -c <config file> -f -V]
```

Where:

- `c` or `--config <config file>`
  sets the path to the configuration file. The file can contain the following variables:
  - loadavg
  - idle
  - onumcpus
  - runnable_proc (for hotplug)
  - apcr
  - freemem
  - swaprate (for memplug)
  To create rules you can use the operators +, *, (, ), /, -, <, >, &l, and !
  See ["Examples" on page 345](#) below for details.

- `f` or `--foreground`
  runs in foreground.

- `V` or `--verbose`
  prints verbose messages.

- `h` or `--help`
  displays short information on command usage.

- `v` or `--version`
  displays version information.
Examples

Enabling and disabling CPUs
The following shows an example configuration file that dynamically adds or takes away CPUs according to the rules given:

```
CPU_MIN=2
CPU_MAX=10
UPDATE=60

HOTPLUG = "(loadavg > onumcpus +0.75) & (idle < 10.0)"
HOTUNPLUG = "(loadavg < onumcpus -0.25) | (idle > 50)"
```

The first two lines specify the minimum and maximum numbers of CPUs. This example ensures that at least two CPUs and no more than ten CPUs are active at any time. Every 60 seconds the daemon checks if a given rule matched against the current system state. If the CPU_MAX variable equals zero, the maximum number of CPUs is equivalent to the number of CPUs detected.

The hotplug line enables a CPU if the current load average (loadavg) is greater than the number of online CPUs (onumcpus) plus 0.75 and the current idle percentage (idle) is below 10 percent.

The hotunplug line disables a CPU if one of the following conditions is true:
- The load is below the number of active CPUs minus 0.25
- The idle percentage is above 50 percent.

You can also use the variable runable_proc, which represents the current number of running processes. For example:

```
HOTPLUG = "RUNABLE_PROC > (onumcpus+2)"
```

The idle percentage is extracted from /proc/stat, whereas the load average and the number of runnable processes is extracted from /proc/loadavg. Information on the current CPUs and their state can be found in the directories below /sys/devices/system/cpu (see Chapter 25, "Managing CPUs," on page 225).

See the man page for more details.

Managing memory
You can use the cpudiant command to react dynamically to changing requirements of the amount of main memory used within a Linux instance running on z/VM.

Before you begin:
- The sys filesystem needs to be mounted to /sys and proc needs to be available at /proc.
- You must load the cmm kernel module. For information about how to load the module, see Chapter 24, "Cooperative memory management," on page 221.

An example configuration file might look like:
The example above illustrates the syntactic format of a memplug and memunplug rule. These two variables must be adjusted depending on the usage and workload of your SUSE Linux Enterprise Server instance. No general or all purpose example configuration can be provided as this does not provide a useful setup for production systems.

Every 60 seconds the daemon checks if a given rule matches against the current system state.

The cmm_min and cmm_max variables define the minimum and maximum size of the cmm static page pool respectively. For an explanation of the cmm page pools, see "Cooperative memory management background" on page 171.

The cmm_inc variable specifies the amount of pages the static page pool is increased (decreased) if a memplug (memunplug) rule is matched.

The memplug rule in the example is matched when:
1. The current swaprate (as shown in the output of the vmstat command) is greater than the current amount of free memory (in megabytes) plus 10.
2. The sum of the free memory (in megabytes) plus 10 is less than the current amount of page cache reads (apcr).

The amount of page-cache reads equals the sum of the bi and bo values shown in the output of `vmstat 1`. The swaprate equals the sum of the si and so fields of the same command. The size of the free memory is retrieved from `/proc/meminfo`.

For further details, see the man page.
dasdfmt - Format a DASD

Use this tool to low-level format ECKD-type direct access storage devices (DASD). Note that this is a software format. To hardware format a raw DASD you must use another System z 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

```
$ dasdfmt -h
```

```
-d cdl -l (default)
-b <blocksize>
-d ldl -l <volser>
-L -k
-p -y -F -v -t 10
-m <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 <blocksize>` or `-blocksize=<blocksize>`
  
  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 `<blocksize>` 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 25 for more details on device nodes.
**dasdfmt**

- **d** `<disklayout>` or `--disk_layout=<disklayout>`
  Formats the device with the compatible disk layout (cdl) or the Linux disk layout (ldl). If the parameter is not specified the default (cdl) is used.

- **L** or `--no_label`
  Valid for `-d ldl` only, where it suppresses the default LNX1 label.

- **l** `<volser>` or `--label=<volser>`
  Specifies the volume serial number (see [VOLSER” on page 21]) 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 21]). This is useful, for example, if the volume serial number has been written with a z/VM tool and should not be overwritten.

- **p** or `--progressbar`
  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 z/VM minidisk with the standard Linux disk layout and a 4 KB blocksize with device node `/dev/dasdc`:
# dasdfmt -b 4096 -d ld1 -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

dasdview - Display DASD structure

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.
- Whether the data on the DASD is encrypted.

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 20 for further information on partitioning.)

Before you start: You need root permissions.

Format

<table>
<thead>
<tr>
<th>dasdview syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>-b 0</td>
</tr>
<tr>
<td>-b &lt;begin&gt;</td>
</tr>
<tr>
<td>-i</td>
</tr>
<tr>
<td>-j</td>
</tr>
<tr>
<td>-c</td>
</tr>
<tr>
<td>&lt;node&gt;</td>
</tr>
</tbody>
</table>

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 cdi. 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:
dasdview

- 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.

-c or --storage-class
  Displays "0" if the data on the disk is encrypted, "1" otherwise.

-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.
dasdview

Specifies the device node of the device for which you want to display information, for example, /dev/dasdzzz. See “DASD naming scheme” on page 25 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:

```bash
# dasdview -x -f /dev/dasdzzz
```

This displays:

```
--- general DASD information ------------------------------------------
device node : /dev/dasdzzz
busid : 0.0.0.0193
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 900a5f08 df72024 0064000f
e000e5a2 05940222 13090674 00000000
00000000 00000000 24241502 df6e0001
0677080f 0074a001 b3500000 00000000
e000e5a2 05940222 13090674 00000000
00000000 00000000 24241502 df6e0001
0677080f 0074a001 b3500000 00000000
800000a1 00001e00 51400009 0909x180
0140c009 7cb7efb7 00000000 00000000
```

#
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 000000101
  (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
reserved : ascii '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'
  : ebcDIC '
  : hex 40404040 40404040 40404040 40404040 40
To display partition information:

```bash
# dasdview -t info -f /dev/dasdzzz
```

This displays:

```plaintext
--- 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:

<table>
<thead>
<tr>
<th>data set</th>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINUX.V0X0193.PART0001.NATIVE</td>
<td>trk</td>
<td>trk</td>
</tr>
<tr>
<td>data set serial number : '0X0193'</td>
<td>2</td>
<td>500</td>
</tr>
<tr>
<td>system code : 'IBM LINUX '</td>
<td>cyl/trk</td>
<td>cyl/trk</td>
</tr>
<tr>
<td>creation date : year 2001, day 317</td>
<td>0/ 2</td>
<td>33/ 5</td>
</tr>
<tr>
<td>LINUX.V0X0193.PART0002.NATIVE</td>
<td>trk</td>
<td>trk</td>
</tr>
<tr>
<td>data set serial number : '0X0193'</td>
<td>501</td>
<td>900</td>
</tr>
<tr>
<td>system code : 'IBM LINUX '</td>
<td>cyl/trk</td>
<td>cyl/trk</td>
</tr>
<tr>
<td>creation date : year 2001, day 317</td>
<td>33/ 6</td>
<td>60/ 1</td>
</tr>
<tr>
<td>LINUX.V0X0193.PART0003.NATIVE</td>
<td>trk</td>
<td>trk</td>
</tr>
<tr>
<td>data set serial number : '0X0193'</td>
<td>901</td>
<td>1499</td>
</tr>
<tr>
<td>system code : 'IBM LINUX '</td>
<td>cyl/trk</td>
<td>cyl/trk</td>
</tr>
<tr>
<td>creation date : year 2001, day 317</td>
<td>60/ 1</td>
<td>99/ 14</td>
</tr>
</tbody>
</table>
```
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)
DS4SREC : dec 7, hex 0007
DS4CCHH : 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
DS4SCYL : 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
DS4DEVYL : dec 0, hex 0000
DS4DEVDT : dec 12, hex 0c
DS4DEVD : dec 0, hex 00
DS4AMTIM : 0000000000000000
DS4AMCAT : 000000
DS4R2TIM : 0000000000000000
DS4F6PRT : 0000000000
DS4VTOCE : 0100000000010100000001
  typeind : dec 1, hex 01
  seqno : dec 0, hex 00
  llimit : hex 00000000 (cyl 0, trk 1)
  ulimit : hex 00000000 (cyl 0, trk 1)
res2 : 0000000000000000
DS4FLVL : dec 0, hex 00
DS4FPPTR : 0000000000 (cyl 0, trk 0, blk 0)
res3 : 0000000000000000
```
To print the contents of a disk to the console starting at block 2 (volume label):

```
# dasdview -b 2 -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 | ?????????????. | 0000000000000000 |
| 40404040 40404040 40404040 40404040 | ?????????????. | 0000000000000000 |
| 40404040 88001000 10000000 00008000 | ??????......... | 0000000000000000 |
| 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 14 -s 128 -2 -f /dev/dasdzzz
```

This displays:

```
+---------------+---------------+----------------------+----------+----------+
| BYTE | BYTE | HEXADECIMAL | EBCDIC | ASCII |
| DECIMAL | HEXADECIMAL |1234 5678 | 12345678 | 12345678 |
+---------------+---------------+----------------------+----------+----------+
| 57344 | E000 | D3C0D5E4 E748E5F0 | LINUX.V0 | ?????K?? |
| 57352 | E008 | E7F0F1F9 F348D7C1 | X0193.PA | ?????K?? |
| 57360 | E010 | D9E3F0F0 F0F148D5 | RT001.N | ?????K?? |
| 57368 | E018 | C1E3C9E5 C5404040 | ATIVE?? | ?????000 |
| 57376 | E020 | 40404040 40404040 | ???????? | 00000000 |
| 57384 | E028 | 40404040 F1F0E7F0 | ????10X0 | 0000?? |
| 57392 | E030 | F1F9F300 0165013D | ATIVE??? | ?????@@@ |
| 57400 | E038 | 63016D01 0000C9C2 | ??????? | @@@@@@@@ |
| 57408 | E040 | D40403C9 D5E4E740 | MLINUX? | ?0?????? |
| 57416 | E048 | 40404066 01300000 | ??????. | 00??e=.. |
| 57424 | E050 | 00000000 88001000 | ....h:. | ....??. |
| 57432 | E058 | 10000000 00000000 | ??.??. | ??.?.??. |
| 57440 | E060 | 00000000 00000000 | .......... | .......... |
| 57448 | E068 | 00010000 00000200 | ?.??? | ?.??? |
| 57456 | E070 | 21000500 00000000 | ??.?. | 1.???? |
| 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 | ................ | ................ |
+----------------------------------------+------------------+------------------+
```

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 | ???????????????? | @@@@@@@@@@@@@@@@ |
+----------------------------------------+------------------+------------------+
```

---

dasdview

- To see what is at block 1234 (in this example there is nothing there):
- To try byte 0 instead:
The compatible disk layout allows you to split DASD into several partitions. Use \texttt{fdasd} to manage partitions on a DASD. You can use \texttt{fdasd} to create, change and delete partitions, and also to change the volume serial number.

- \texttt{fdasd} checks that the volume has a valid volume label and VTOC. If either is missing or incorrect, \texttt{fdasd} recreates it.
- Calling \texttt{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.

\textbf{Note:} To partition a SCSI disk, use \texttt{fdisk} rather than \texttt{fdasd}.

\textbf{Before you start:}
- You must have root permissions.
- The disk must be formatted with \texttt{dasdfmt} with the (default) -d cds1 option.

For more information on partitions see \texttt{"The IBM label partitioning scheme" on page 20}.

\textbf{Attention:} Careless use of \texttt{fdasd} can result in loss of data.

\begin{center}
\textbf{Format}
\end{center}

\begin{center}
\texttt{fdasd} syntax
\end{center}

\begin{center}
\begin{itemize}
\item \texttt{-s} \texttt{-r} \texttt{partitioning options}
\end{itemize}
\end{center}

\begin{center}
\texttt{partitioning options:}
\end{center}

\begin{center}
\begin{itemize}
\item \texttt{-h} \texttt{-v} \texttt{-a} \texttt{-k} \texttt{-l} \texttt{-c} \texttt{-i} \texttt{-p}
\end{itemize}
\end{center}

\begin{center}
\texttt{Notes:}
\end{center}

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:
fdasd

-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 21). This is useful, for example, if the volume serial number has been written with a z/VM tool and should not be overwritten.

-l <volser> or --label=<volser>
Specifies the volume serial number (see "VOLSER" on page 21).
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:
-1 'a\b\$c#'
to get:
A$B$C#

VOLSER is interpreted as an ASCII string and is automatically converted to uppercase, padded with blanks and finally converted to EBCDIC before being written to disk.

Do not use the following reserved volume serials:
• SCRTCH
• PRIVAT
• MIGRAT
• Lnnnnn (L followed by a five digit number)
These are used as keywords by other operating systems (z/OS).

Omitting this parameter causes fdasd to prompt for it, if it is needed.

-c <conf_file> or --config <conf_file>
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 25 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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>print this menu</td>
</tr>
<tr>
<td>p</td>
<td>print the partition table</td>
</tr>
<tr>
<td>n</td>
<td>add a new partition</td>
</tr>
<tr>
<td>d</td>
<td>delete a partition</td>
</tr>
<tr>
<td>v</td>
<td>change volume serial</td>
</tr>
<tr>
<td>t</td>
<td>change partition type</td>
</tr>
<tr>
<td>r</td>
<td>re-create VTOC and delete all partitions</td>
</tr>
<tr>
<td>u</td>
<td>re-create VTOC re-using existing partition sizes</td>
</tr>
<tr>
<td>s</td>
<td>show mapping (partition number - data set name)</td>
</tr>
<tr>
<td>q</td>
<td>quit without saving changes</td>
</tr>
<tr>
<td>w</td>
<td>write table to disk and exit</td>
</tr>
</tbody>
</table>

**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
fdasd

- 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 21 for the format.

t  Changes the partition type. You will be asked to identify the partition to be changed. You will then be asked for the new partition type (Linux native or swap). Note that this type is a guideline; the actual use Linux makes of the partition depends on how it is defined with the mkswap or mkxxfs tools. The main function of the partition type is to describe the partition to other operating systems so that, for example, swap partitions can be skipped by backup programs.

r  Recreates the VTOC and thereby deletes all partitions.

u  Recreates all VTOC labels without removing all partitions. Existing partition sizes will be reused. This is useful to repair damaged labels or migrate partitions created with older versions of fdasd.

s  Displays the mapping of partition numbers to data set names. For example:

Command (m for help): s

device ........: /dev/dasdzzz
volume label ...: VOL1
volume serial ...: 0X0193

WARNING: This mapping may be NOT up-to-date,
if you have NOT saved your last changes!

/dev/dasdzzz1 - LINUX.V0X0193.PART0001.NATIVE
/dev/dasdzzz2 - LINUX.V0X0193.PART0002.NATIVE
/dev/dasdzzz3 - LINUX.V0X0193.PART0003.NATIVE

q  Quits fdasd without updating the disk. Any changes you have made (in this session) will be discarded.

w  Writes your changes to disk and exits. After the data is written Linux will reread the partition table.

Examples

Example using the menu
This section gives an example of how to use fdasd to create two partitions on a z/VM minidisk, change the type of one of the partitions, save the changes and check the results.

In this example, we will format a z/VM minidisk with the compatible disk layout. The minidisk has device number 193.

1. Call fdasd, specifying the minidisk:

   # fdasd /dev/dasdzzz

   fdasd reads the existing data and displays the menu:
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 -------------------------------

<table>
<thead>
<tr>
<th>Device</th>
<th>start</th>
<th>end</th>
<th>length</th>
<th>Id</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/dasdzzz1</td>
<td>2</td>
<td>700</td>
<td>699</td>
<td>1</td>
<td>Linux native</td>
</tr>
<tr>
<td>/dev/dasdzzz2</td>
<td>701</td>
<td>1100</td>
<td>400</td>
<td>2</td>
<td>Linux native</td>
</tr>
<tr>
<td>1101</td>
<td>1499</td>
<td>399</td>
<td>unused</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 -------------------------------

<table>
<thead>
<tr>
<th>Device</th>
<th>start</th>
<th>end</th>
<th>length</th>
<th>Id</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/dasdzzz1</td>
<td>2</td>
<td>700</td>
<td>699</td>
<td>1</td>
<td>Linux native</td>
</tr>
<tr>
<td>/dev/dasdzzz2</td>
<td>701</td>
<td>1100</td>
<td>400</td>
<td>2</td>
<td>Linux native</td>
</tr>
<tr>
<td>1101</td>
<td>1499</td>
<td>399</td>
<td>unused</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:
8. Write the results to disk using the \texttt{w} option:

Command (m for help): \texttt{w}

writing VTOC...
rereading partition table...

\textbf{Example using options}

You can partition using the \texttt{-a} or \texttt{-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 \texttt{-a} parameter you can create one large partition on a DASD:

\begin{verbatim}
# fdasd -a /dev/dasdzzz
auto-creating one partition for the whole disk...
writing volume label...
writing VTOC...
rereading partition table...
\end{verbatim}

This will create a partition as follows:

\begin{verbatim}
Device start end length Id System
/dev/dasdzzz1 2 1499 1498 1 Linux native
\end{verbatim}

Using a configuration file you can create several partitions. For example, the following configuration file, \texttt{config}, creates three partitions:

\begin{verbatim}
[first,500]
[501,1100]
[1101,last]
\end{verbatim}

Submitting the command with the \texttt{-c} option creates the partitions:

\begin{verbatim}
# fdasd -c config /dev/dasdzzz
 parsing config file 'config'...
 writing volume label...
 writing VTOC...
 rereading partition table...
\end{verbatim}

This creates partitions as follows:
<table>
<thead>
<tr>
<th>Device</th>
<th>start</th>
<th>end</th>
<th>length</th>
<th>Id</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/dasdzz1</td>
<td>2</td>
<td>500</td>
<td>499</td>
<td>1</td>
<td>Linux native</td>
</tr>
<tr>
<td>/dev/dasdzz2</td>
<td>501</td>
<td>1100</td>
<td>600</td>
<td>2</td>
<td>Linux native</td>
</tr>
<tr>
<td>/dev/dasdzz3</td>
<td>1101</td>
<td>1499</td>
<td>399</td>
<td>3</td>
<td>Linux native</td>
</tr>
</tbody>
</table>
Ischp - List channel paths

Use this command to display information about channel paths.

Format

```
lschp syntax

lschp
--help
--version
```

where:
- `-v` or `--version`
  Prints the version number of `lschp` and exits.
- `-h` or `--help`
  Prints out a short help text, then exits.

Output column description:

CHPID
Channel-path identifier.

Vary
Logical channel-path state:
- 0 = channel-path is not used for I/O.
- 1 = channel-path is used for I/O.

Cfg.
Channel-path configure state:
- 0 = stand-by
- 1 = configured
- 2 = reserved
- 3 = not recognized

Type
Channel-path type identifier.

Cmg
Channel measurement group identifier.

Shared
Indicates whether a channel-path is shared between LPARs:
- 0 = channel-path is not shared
- 1 = channel-path is shared

A column value of "-" indicates that a facility associated with the respective channel-path attribute is not available.
Ischp

Examples

- To query the configuration status of channel path ID 0.40 issue:

```bash
# lschp
CHPID Vary Cfg. Type Cmg Shared
====================================
 .
 .
 0.40 1 1 1b 2 1
 .
```

The value under Cfg. shows that the channel path is configured (1).
**lscss - List subchannels**

This command is used to gather subchannel information from sysfs and display it in a summary format.

**Format**

```
lscss syntax

lscss [ -s ] [ -t ] <devicetype> [<model>]
```

Where:

- **-s** or **--short**
  strips the “0.n.” from the device bus IDs in the command output.

- **-t** or **--devtype**
  limits the output to information on the specified device types and, if provided, the specified model.

- **<devicetype>**
  specifies a device type.

- **<model>**
  is a specific model of the specified device type.

**Examples**

- This command lists all subchannels:

  ```
  # ls css
  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
 [data truncated for brevity]
  ```

- 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

```bash
# 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
```
Isdasm - List DASD devices

This command is used to gather information on DASD devices from sysfs and display it in a summary format.

Format

```
Isdasm syntax

Isdasm [options] <device_bus_id>
```

Where:

- `-a` or `--offline`
  - includes devices that are currently offline.

- `-b` or `--base`
  - omits PAV alias devices. Lists only base devices.

- `-s` or `--short`
  - strips the “0.n.” from the device bus IDs in the command output.

- `-v` or `--verbose`
  - Obsolete. This option has no effect on the output.

- `-l` or `--long`
  - extends the output to include UID and attributes.

- `-c` or `--compat`
  - creates output of this command as with versions earlier than 1.7.0.

- `-u` or `--uid`
  - includes and sorts output by UID.

- `--version`
  - prints the version of the s390-tools package and the command.

- `<device_bus_id>`
  - limits the output to information on the specified devices only.

- `-h` or `--help`
  - Prints out a short help text, then exits.
Examples

- The following command lists all DASD (including offline DASDS):

```
# lsdasd -a
```

<table>
<thead>
<tr>
<th>Bus-ID</th>
<th>Status</th>
<th>Name</th>
<th>Device</th>
<th>Type</th>
<th>BlkSz</th>
<th>Size</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0190</td>
<td>offline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0.0191</td>
<td>offline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0.019d</td>
<td>offline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0.019e</td>
<td>offline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0.0592</td>
<td>offline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0.4711</td>
<td>offline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0.4712</td>
<td>offline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0.472c</td>
<td>offline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0.4d80</td>
<td>active</td>
<td>dasda</td>
<td>94:0</td>
<td>ECKD</td>
<td>4096</td>
<td>4695MB</td>
<td>1202040</td>
</tr>
<tr>
<td>0.0.4f19</td>
<td>active</td>
<td>dasdb</td>
<td>94:4</td>
<td>ECKD</td>
<td>4096</td>
<td>23034MB</td>
<td>5896800</td>
</tr>
<tr>
<td>0.0.4d81</td>
<td>active</td>
<td>dasdc</td>
<td>94:8</td>
<td>ECKD</td>
<td>4096</td>
<td>4695MB</td>
<td>1202040</td>
</tr>
<tr>
<td>0.0.4d82</td>
<td>active</td>
<td>dasdd</td>
<td>94:12</td>
<td>ECKD</td>
<td>4096</td>
<td>4695MB</td>
<td>1202040</td>
</tr>
<tr>
<td>0.0.4d83</td>
<td>active</td>
<td>dasde</td>
<td>94:16</td>
<td>ECKD</td>
<td>4096</td>
<td>4695MB</td>
<td>1202040</td>
</tr>
</tbody>
</table>

- The following command shows information only for the DASD with device number 0x4d80 and strips the “0.n.” from the bus IDs in the output:

```
# lsdasd -s 4d80
```

<table>
<thead>
<tr>
<th>Bus-ID</th>
<th>Status</th>
<th>Name</th>
<th>Device</th>
<th>Type</th>
<th>BlkSz</th>
<th>Size</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>4d80</td>
<td>active</td>
<td>dasda</td>
<td>94:0</td>
<td>ECKD</td>
<td>4096</td>
<td>4695MB</td>
<td>1202040</td>
</tr>
</tbody>
</table>

- The following command shows only online DASDs in the previous format:

```
# lsdasd -c
```

0.0.4d80(ECKD) at ( 94: 0) is dasda : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4f19(ECKD) at ( 94: 4) is dasdb : active at blocksize 4096, 5896800 blocks, 23034 MB
0.0.4d81(ECKD) at ( 94: 8) is dasdc : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4d82(ECKD) at ( 94:12) is dasdd : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4d83(ECKD) at ( 94:16) is dasde : active at blocksize 4096, 1202040 blocks, 4695 MB
Isluns - Discover LUNs in Fibre Channel SANs

Use the Isluns command to discover and scan LUNs in Fibre Channel Storage Area Networks (SANs).

Format

```plaintext
Isluns syntax

>>> Isluns
   -c <name>
   -p <name>

Where:

-c or --ccw <name>
  Shows LUNs for a specific adapter. The adapter name is of the form 0.0.XXXX.

-p or --port <name>
  Shows LUNs for a specific port. The port name is an 8-byte hexadecimal value, for example, 0x500500ab0012cd00.

-v or --version
  Prints the version number of Isluns and exits.

-h or --help
  Prints out an overview of the syntax.
```

Examples

• This example shows all LUNs for port 0x50050763030c562:

```bash
# Isluns --port 0x50050763030c562
Scanning for LUNs on adapter 0.0.5922
  at port 0x50050763030c562:
    0x4010400000000000
    0x4010400100000000
    0x4010400200000000
    0x4010400300000000
    0x4010400400000000
    0x4010400500000000
```

• This example shows all LUNs for adapter 0.0.5922:

```bash
# Isluns -c 0.0.5922
  at port 0x50050763030c562:
    0x4010400000000000
    0x4010400100000000
    0x4010400200000000
    0x4010400300000000
    0x4010400400000000
    0x4010400500000000
    0x4010400600000000
    0x4010400700000000
    0x4010400800000000
    0x4010400900000000
    0x4010400a00000000
    0x4010400b00000000
    0x4010400c00000000
    0x4010400d00000000
    0x4010400e00000000
    0x4010400f00000000
```

Chapter 40. Useful Linux commands 373
Isqeth

Isqeth - List qeth based network devices

This command is used to gather information on qeth-based network devices from sysfs and display it in a summary format.

Before you start: To be able to use this command you must also have installed qethconf (see “qethconf - Configure qeth devices” on page 400). You install qethconf and lsqeth with the same RPM.

Format

Isqeth syntax

```
lsqeth [-p] <interface>
```

Where:

- **-p** or **--proc**
  displays the interface information in the former /proc/qeth format. This option can generate input to tools that expect this particular format.

- **<interface>**
  limits the output to information on the specified interface only.

- **-h** or **--help**
  Prints out a short help text, then exits.

Examples

- The following command lists information on interface eth0 in the default format:
  
  ```
  # lsqeth eth0
  Device name : eth0
  -----------------------------
  card_type : OSD_100
  cdev0 : 0.0.f5a2
  cdev1 : 0.0.f5a3
  cdev2 : 0.0.f5a4
  chpid : B5
  online : 1
  portname : OSAPORT
  portno : 0
  route4 : no
  route6 : no
  checksumming : sw checksumming
  state : UP (LAN ONLINE)
  priority_queueing : always queue 2
  fake_broadcast : 0
  buffer_count : 16
  layer2 : 0
  large_send : no
  ```

- The following command lists information on all qeth-based interfaces in the former /proc/qeth format:
<table>
<thead>
<tr>
<th>devices</th>
<th>CHPID</th>
<th>interface</th>
<th>cardtype</th>
<th>port</th>
<th>chksum</th>
<th>prio-q'ing</th>
<th>rtr4</th>
<th>rtr6</th>
<th>fzs</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.833f/0.0.8340/0.0.8341</td>
<td>xFE</td>
<td>hsi0</td>
<td>HiperSockets</td>
<td>0</td>
<td>sw</td>
<td>always_q_2</td>
<td>no</td>
<td>no</td>
<td>n/a</td>
<td>16</td>
</tr>
<tr>
<td>0.0.f5a2/0.0.f5a3/0.0.f5a4</td>
<td>xB5</td>
<td>eth0</td>
<td>OSD_100</td>
<td>0</td>
<td>sw</td>
<td>always_q_2</td>
<td>no</td>
<td>no</td>
<td>n/a</td>
<td>16</td>
</tr>
<tr>
<td>0.0.fba2/0.0.fba3/0.0.fba4</td>
<td>xB5</td>
<td>eth0</td>
<td>OSD_100</td>
<td>0</td>
<td>sw</td>
<td>always_q_2</td>
<td>no</td>
<td>no</td>
<td>n/a</td>
<td>16</td>
</tr>
</tbody>
</table>
lsreipl

lsreipl - List IPL settings

Use this command to see from which device your system will boot after you issue the reboot command. Further you can query the system for information about the current boot device.

Format

lsreipl syntax

where:

- \texttt{-i} or \texttt{--ipl}
  
  Prints the IPL setting.

- \texttt{-v} or \texttt{--version}
  
  Prints the version number of \texttt{lsreipl} and exits.

- \texttt{-h} or \texttt{--help}
  
  Prints out an overview of the syntax. Any other parameters are ignored.

Examples

- This example shows the current IPL settings:

  # lsreipl
  
  Re-IPL type: fcp
  WPN: 0x500507630300c562
  LUN: 0x401040b300000000
  Device: 0.0.1700
  bootprog: 0
  br_lba: 0
Isshut - List the configuration for system states

Use this command to see how the system is configured to behave in the following system states: halt, panic, power off, and reboot.

Format

```bash
Isshut syntax

```

where:

- `-h` or `--help`
  - Prints out a short help text, then exits.

- `-v` or `--version`
  - Prints the version number of Isshut and exits.

Examples

- To query the configuration issue:

  ```bash
  # Isshut
  Trigger  Action
  ********************
  Halt      stop
  Panic     stop
  Power off vmcmd (LOGOFF)
  Reboot    reipl
  ```
**Istape**

**Istape - List tape devices**

This command is used to gather information on CCW-attached tape devices and tape devices attached to the SCSI bus from sysfs (see “Displaying tape information” on page 70) and display it in a summary format.

For information about SCSI tape devices, the command uses the following sources for the information displayed:

- The IBMtape or the open source lin_tape driver.
- The sg_inq command from the scsi/sg3_utils package.
- The st (SCSI tape) device driver in the Linux kernel.

If you use the IBMtape or lin_tape driver, the sg_inq utility is required. If sg_inq is missing, certain information about the IBMtape or lin_tape driver cannot be displayed.

**Format**

```
Istape syntax

![](istape_syntax.png)

Notes:

1. Specify the first device bus-ID with a leading blank.
```

Where:

- `-s` or `--shortid`
  
  strips the “0.n.” from the device bus-IDs in the command output. For CCW-attached devices only.

- `-t` or `--type`
  
  limits the output to information on the specified type or types of CCW-attached devices only.

- `--ccw-only`
  
  limits the output to information on CCW-attached devices only.

- `--scsi-only`
  
  limits the output to information on tape devices attached to the SCSI bus.

- `--online` or `--offline`
  
  limits the output to information on online or offline CCW-attached tape devices only.
Istape

<device_bus_id>
limits the output to information on the specified tape device or devices only.

--verbose
For tape devices attached to the SCSI bus only. Prints the serial of the tape as well as information about the FCP connection as an additional text line below each SCSI tape in the list.

-h or --help
prints a short help text.

--version
prints the version of the command.

Output attributes
The attributes in the output provide this data:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic</td>
<td>SCSI generic device file for the tape drive (for example /dev/sg0). This attribute is empty if the <code>sg_inq</code> command is not available.</td>
</tr>
<tr>
<td>Device</td>
<td>Main device file for accessing the tape drive, for example:</td>
</tr>
<tr>
<td></td>
<td>• <code>/dev/st0</code> for a tape drive attached through the Linux st device driver</td>
</tr>
<tr>
<td></td>
<td>• <code>/dev/sch0</code> for a medium changer device attached through the Linux changer device driver</td>
</tr>
<tr>
<td></td>
<td>• <code>/dev/IBMchanger0</code> for a medium changer attached through the IBMtape or lin_tape device driver</td>
</tr>
<tr>
<td></td>
<td>• <code>/dev/IBMtape0</code> for a tape drive attached through the IBMtape or lin_tape device driver</td>
</tr>
<tr>
<td>Target</td>
<td>The ID in Linux used to identify the SCSI device.</td>
</tr>
<tr>
<td>Vendor</td>
<td>The vendor field from the tape drive.</td>
</tr>
<tr>
<td>Model</td>
<td>The model field from the tape drive.</td>
</tr>
<tr>
<td>Type</td>
<td>&quot;Tapedrv&quot; for a tape driver or &quot;changer&quot; for a medium changer.</td>
</tr>
<tr>
<td>State</td>
<td>The state of the SCSI device in Linux. This is an internal state of the Linux kernel, any state other than &quot;running&quot; can indicate problems.</td>
</tr>
<tr>
<td>HBA</td>
<td>The FCP adapter to which the tape drive is attached.</td>
</tr>
<tr>
<td>WWPN</td>
<td>The WWPN (World Wide Port Name) of the tape drive in the SAN.</td>
</tr>
<tr>
<td>Serial</td>
<td>The serial number field from the tape drive.</td>
</tr>
</tbody>
</table>

Examples

- This command displays information on all tapes found, here one CCW-attached tape and one tape and changer device configured for zFCP:

```
#> lstape
FICON/ESCON tapes (found 1):
TapeNo BusID CuType/Model DevType/Model BlkSize State Op MedState
  0 0.0.0480 3480/01 3480/04 auto UNUSED --- UNLOADED
SCSI tape devices (found 2):
Generic Device Target Vendor Model Type State
sg4 IBMchanger0 0:0:0:0 IBM 03590H11 changer running
sg5 IBMtape0 0:0:0:1 IBM 03590H11 tapedrv running
```
If only the generic tape driver (st) and the generic changer driver (ch) are loaded, the output will list those names in the device section:

```bash
#> lstape
FICON/ESCON tapes (found 1):
TapeNo BusID CuType/Model DevType/Model BlkSize State Op MedState
0 0.0.0480 3480/01 3480/04 auto UNUSED --- UNLOADED

SCSI tape devices (found 2):
Generic Device Target Vendor Model Type State
sg0 sch0 0:0:0:0 IBM 03590H11 changer running
sg1 st0 0:0:0:1 IBM 03590H11 tapedrv running
```

- This command displays information on all available CCW-attached tapes.

```bash
# lstape --ccw-only
TapeNo BusID CuType/Model DevType/DevMod BlkSize State Op MedState
0 0.0.0132 3590/50 3590/11 auto IN_USE --- LOADED
1 0.0.0110 3490/10 3490/40 auto UNUSED --- UNLOADED
2 0.0.0133 3590/50 3590/11 auto IN_USE --- LOADED
3 0.0.012a 3480/01 3480/04 auto UNUSED --- UNLOADED
N/A 0.0.01f8 3480/01 3480/04 N/A OFFLINE --- N/A
```

- This command limits the output to tapes of type 3480 and 3490.

```bash
# 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.

```bash
# 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
N/A 0.0.01f8 3480/01 3480/04 N/A OFFLINE --- N/A
```

- 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.

```bash
# lstape -s 0.0.012a
TapeNo BusID CuType/Model DevType/DevMod BlkSize State Op MedState
3 012a 3480/01 3480/04 auto UNUSED --- UNLOADED
```

- This command limits the output to SCSI devices but gives more details. Note that the serial numbers are only displayed if the `sg_inq` command is found on the system.

```bash
#> lstape --scsi-only --verbose
Generic Device Target Vendor Model Type State
HBA WWPN Serial
sg0 st0 0:0:0:1 IBM 03590H11 tapedrv running
0.0.1708 0x500507630040727b NO/INQ
sg1 sch0 0:0:0:2 IBM 03590H11 changer running
0.0.1708 0x500507630040727b NO/INQ
```

380 Device Drivers, Features, and Commands - SUSE Linux Enterprise Server 11 for System z
lszcrypt - Display zcrypt devices

Use the lszcrypt command to display information about cryptographic adapters managed by zcrypt and zcrypt's AP bus attributes. To set the attributes, use "chzcrypt - Modify the zcrypt configuration" on page 342. The following information can be displayed for each cryptographic adapter:

- The card type
- The online status
- The hardware card type
- The hardware queue depth
- The request count

The following AP bus attributes can be displayed:

- The AP domain
- The configuration timer
- The poll thread status

Before you start:

- The sysfs file system must be mounted.

Format

lszcrypt syntax

```
lszcrypt -V -b <device ID>
```

Where:

- **-V, -VV or --verbose**
  increases the verbose level for cryptographic adapter information. The maximum verbose level is two (-VV). At verbose level one (-V) card type and online status are displayed. At verbose level two card type, online status, hardware card type, hardware queue depth, and request count are displayed.

- **<device ID>**
  specifies the cryptographic adapter which will be displayed. A cryptographic adapter can be specified either in decimal notation or hexadecimal notation using a '0x' prefix. If no adapters are specified information about all available adapters will be displayed.

- **-b or --bus**
  displays the AP bus attributes.

- **-h or --help**
  displays short information on command usage.

- **-v or --version**
  displays version information.
Examples

This section illustrates common uses for `lszcrypt`.

- To display card type and online status of all available cryptographic adapters:
  
  ```
  lszcrypt -V
  ```

- To display card type, online status, hardware card type, hardware queue depth, and request count for cryptographic adapters 0, 1, 10, and 12 (in decimal notation):
  
  ```
  lszcrypt -VV 0 1 10 12
  ```

- To display AP bus information:
  
  ```
  lszcrypt -b
  ```
## lszfcp - List zfcp devices

This command is used to gather information on zfcp adapters, ports, units, and their associated class devices from sysfs and to display it in a summary format.

### Format

#### lszfcp syntax

```
```

Where:

- `-H` or `--hosts`
  shows information about hosts.

- `-P` or `--ports`
  shows information about ports.

- `-D` or `--devices`
  shows information about SCSI devices.

- `-a` or `--attributes`
  shows all attributes (implies `-V`).

- `-V` or `--verbose`
  shows sysfs paths of associated class and bus devices.

- `-b` or `--busid <device_bus_id>`
  limits the output to information on the specified device.

- `-p` or `--wwpn <port_name>`
  limits the output to information on the specified port name.

- `-l` or `--lun <lun_id>`
  limits the output to information on the specified LUN.

- `-s` or `--sysfs <mount_point>`
  specifies the mount point for sysfs.

- `-v` or `--version`
  prints version information.

- `-h` or `--help`
  prints a short help text.

### Examples

- This command displays information on all available hosts, ports, and SCSI devices.
This command limits the output to the SCSI device with device bus-ID 0.0.3d0c:

```bash
# 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
0.0.500c/0x50050763030bc562/0x4010403200000000 1:0:0:0
...
mon_fsstatd – Monitor z/VM guest file system size

The mon_fsstatd command is a user space daemon that collects physical file system size data from a Linux guest and periodically writes the data as defined records to the z/VM monitor stream using the monwriter character device driver. You can start the daemon with a service script /etc/init.d/mon_statd or call it manually. When it is called with the service utility, it reads the configuration file /etc/sysconfig/mon_statd.

Before you start:

- Install the monwriter device driver and set up z/VM to start the collection of monitor sample data. See Chapter 15, "Writing application APPLDATA records," on page 179 for information on the setup for and usage of the monwriter device driver.
- Customize the configuration file /etc/sysconfig/mon_statd if you plan to call it with the service utility.

Format

You can run the mon_fsstatd command in two ways:

- Calling mon_statd with the service utility. This method will read the configuration file /etc/sysconfig/mon_statd. The mon_statd service script also controls other daemons, such as mon_procd.
- Calling mon_fsstatd from a command line.

Service utility syntax

```
mon_statd service utility syntax

service mon_statd
  /etc/init.d/mon_statd
  start
  stop
  status
  restart
```

Where:

start Enables monitoring of guest file system size, using the configuration in /etc/sysconfig/mon_statd.

stop Disable monitoring of guest file system size.

status Show current status of guest file system size monitoring.

restart Stops and restarts monitoring. Useful to re-read the configuration file when it was changed.

Configuration file keywords:

```
FSSTAT_INTERVAL="<n>"
Specifies the desired sampling interval in seconds.

FSSTAT="yes | no"
Specifies whether to enable the mon_fsstatd daemon. Set to "yes" to enable the daemon. Anything other than "yes" will be interpreted as "no".
```
mon_fsstatd

Command-line syntax

```
mon_fsstatd command-line syntax

mon_fsstatd [-i <seconds>] [-a]
```

Where:
- **-i or --interval** `<seconds>` specifies the desired sampling interval in seconds.
- **-a or --attach** runs the daemon in the foreground.
- **-h or --help** displays help information for the command.
- **-v or --version** displays version information for the command.

Examples

Examples of service utility use
Example configuration file for mon_statd (/etc/sysconfig/mon_statd).
- This example sets the sampling interval to 30 seconds and enables the mon_fsstatd daemon:

  ```
  FSSTAT_INTERVAL="30"
  FSSTAT="yes"
  ```

Example of mon_statd use (note that your output may look different and include messages for other daemons, such as mon_procd):
- To enable guest file system size monitoring:

  ```
  > service mon_statd start
  ...
  Starting mon_fsstatd: [ OK ]
  ...
  ```

- To display the status:

  ```
  > service mon_statd status
  ...
  mon_fsstatd (pid 1075, interval: 30) is running.
  ...
  ```

- To disable guest file system size monitoring:

  ```
  > service mon_statd stop
  ...
  Stopping mon_fsstatd: [ OK ]
  ...
  ```

- To display the status again and check that monitoring is now disabled:
mon_fsstatd

To restart the daemon and re-read the configuration file:

> service mon_statd restart

... stopping mon_fsstatd: [ OK ]

starting mon_fsstatd: [ OK ]

... 

Examples of command-line use

• To start mon_fsstatd with default setting:

  > mon_fsstatd

• To start mon_fsstatd with a sampling interval of 30 seconds:

  > mon_fsstatd -i 30

• To start mon_fsstatd and have it run in the foreground:

  > mon_fsstatd -a

• To start mon_fsstatd with a sampling interval of 45 seconds and have it run in the foreground:

  > mon_fsstatd -a -i 45

Usage

Processing monitor data

The feature writes physical file system size data for a Linux guest to the z/VM monitor stream. The following is the format of the file system size data that is passed to the z/VM monitor stream. One sample monitor record is written for each physical file system mounted at the time of the sample interval. The monitor data in each record contains a header, fsstatd_hdr, followed by the length of the file system name, then the file system name, then the length of the mount directory name, followed by the mount directory name, then followed by the size information for that file system (as obtained from statvfs).

Table 51. File system size data format

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__u64</td>
<td>time_stamp</td>
<td>Time at which the file system data was sampled.</td>
</tr>
<tr>
<td>__u16</td>
<td>fsstat_data_len</td>
<td>Length of data following this fsstatd_hdr.</td>
</tr>
<tr>
<td>__u16</td>
<td>fsstat_data_offset</td>
<td>Offset from start of fsstatd_hdr to start of file system data (that is, to the fields beginning with fs__).</td>
</tr>
<tr>
<td>__u16</td>
<td>fs_name_len</td>
<td>Length of the file system name. If the file system name was too long to fit in the monitor record, this is the length of the portion of the name that is contained in the monitor record.</td>
</tr>
</tbody>
</table>
Table 51. File system size data format (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fs_name_len</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fs_name</td>
<td></td>
</tr>
<tr>
<td>__u16</td>
<td>fs_dir_len</td>
<td>Length of the mount directory name. If the mount directory name was too long to fit in the monitor record, this is the length of the portion of the name that is contained in the monitor record.</td>
</tr>
<tr>
<td></td>
<td>fs_dir</td>
<td></td>
</tr>
<tr>
<td>__u16</td>
<td>fs_type_len</td>
<td>Length of the mount type. If the mount type is too long to fit in the monitor record, this is the length of the portion that is contained in the monitor record.</td>
</tr>
<tr>
<td></td>
<td>fs_type</td>
<td></td>
</tr>
<tr>
<td>__u64</td>
<td>fs_bsize</td>
<td>File system block size.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_frsise</td>
<td>Fragment size.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_blocks</td>
<td>Total data blocks in file system.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_bfree</td>
<td>Free blocks in fs.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_bavail</td>
<td>Free blocks avail to non-superuser.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_files</td>
<td>Total file nodes in file system.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_ffree</td>
<td>Free file nodes in fs.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_favail</td>
<td>Free file nodes available to non-superuser.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_flag</td>
<td>Mount flags.</td>
</tr>
</tbody>
</table>

The following data structures map the fixed-length portion of the record above. See /s390-tools/mon_tools/mon_fsstatd.h for these mappings.

```c
struct fsstatd_hdr {
    __u64 time_stamp;
    __u16 fsstat_data_len;
    __u16 fsstat_data_offset;
} __attribute__((packed));

struct fsstatd_data {
    __u64 fs_bsize;
    __u64 fs_frsise;
    __u64 fs_blocks;
    __u64 fs_bfree;
    __u64 fs_bavail;
    __u64 fs_type;
    __u64 fs_files;
    __u64 fs_ffree;
    __u64 fs_favail;
    __u64 fs_flag;
};
```

The time_stamp should be used to correlate all file systems that were sampled in a given interval.
**mon_fsstatd**

**Note:** The data length field (fs_data_len) in the fsstatd_hdr (not the length of the monitor record itself) should be used to determine how much data there is in the monitor record.

**Reading the monitor data**

As described in the monwriter documentation, all records written to the z/VM monitor stream begin with a product identifier. The product ID is a 16-byte structure of the form pppppppffnvvvrrmm, where for records written by mon_fsstatd, these values will be:

- **pppppp** is a fixed ASCII string LNXAPPL.
- **ff** is the application number for mon_fsstatd = x'0001'.
- **n** is the record number = x'00'.
- **vv** is the version number = x'0000'.
- **rr** is reserved for future use and should be ignored.
- **mm** is reserved for mon_fsstatd and should be ignored.

**Note:** Though the mod_level field (mm) of the product ID will vary, there is no relationship between any particular mod_level and file system. The mod_level field should be ignored by the reader of this monitor data.

There are many tools available to read z/VM monitor data. One such tool is the Linux monreader character device driver. See Chapter 16, “Reading z/VM monitor records” for more information about monreader.

**Further information**

- Refer to *z/VM: 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 *z/VM: CP Commands and Utilities Reference*, SC24-6081 for information on the CP commands.
- Refer to *z/VM: Performance*, SC24-6109 for information on monitor APPLDATA.
mon_procd

mon_procd – Monitor Linux guest

The mon_procd command is a user space daemon that writes system summary information and information of each process for up to 100 concurrent processes that are managed by a Linux guest to the z/VM monitor stream using the monwriter character device driver. You can start the daemon with a service script /etc/init.d/mon_statd or call it manually. When it is called with the service utility, it reads the configuration file /etc/sysconfig/mon_statd.

Before you start:

- Install the monwriter device driver and set up z/VM to start the collection of monitor sample data. See Chapter 15, “Writing application APPLDATA records,” on page 179 for information on the setup for and usage of the monwriter device driver.
- Customize the configuration file /etc/sysconfig/mon_statd if you plan to call it with the service utility.
- The z/VM virtual machine in which the Linux guest running this daemon resides must have the OPTION APPLMON statement in its CP directory entry.

Format

You can run the mon_procd command in two ways:

- Calling mon_procd with the service utility. Use this method when you have the mon_statd service script installed in /etc/init.d. This method will read the configuration file /etc/sysconfig/mon_statd. The mon_statd service script also controls other daemons, such as mon_fsstatd.
- Calling mon_procd manually from a command line.

Service utility syntax

```
mon_statd service utility syntax
```

```
/service mon_statd
/etc/init.d/ mon_statd
```

Where:

- **start** Enables monitoring of guest process data, using the configuration in /etc/sysconfig/mon_statd.
- **stop** Disable monitoring of guest process data.
- **status** Show current status of guest process data monitoring.
- **restart** Stops and restarts guest process data monitoring. Useful in order to re-read the configuration file when it has changed.

**Configuration file keywords:**

```
PROC_INTERVAL="<n>"
```

Specifies the desired sampling interval in seconds.
**PROC=“yes | no”**

Specifies whether to enable the mon_procd daemon. Set to “yes” to enable the daemon. Anything other than "yes" will be interpreted as "no".

**Command-line syntax**

```
mon_procd command-line syntax
```

```
mon_procd
 -i 60
     -i <seconds>
 -a
```

Where:

- **-i or --interval <seconds>**
  specifies the desired sampling interval in seconds.

- **-a or --attach**
  runs the daemon in the foreground.

- **-h or --help**
  displays help information for the command.

- **-v or --version**
  displays version information for the command.

**Examples**

**Examples of service utility use**

Example configuration file for mon_statd (/etc/sysconfig/mon_statd).

- This example sets the sampling interval to 30 seconds and enables the mon_procd:

```
PROC_INTERVAL="30"
PROC="yes"
```

Example of mon_statd use (note that your output might look different and include messages for other daemons, such as mon_fsstatd):

- **To enable guest process data monitoring:**

  ```
  > service mon_statd start
  ...
  Starting mon_procd: [ OK ]
  ...
  ```

- **To display the status:**

  ```
  > service mon_statd status
  ...
  mon_procd (pid 1075, interval: 30) is running.
  ...
  ```

- **To disable guest process data monitoring:**
mon_procd

> service mon_statd stop
... $opping mon_procd: [ OK ]
... 

• To display the status again and check that monitoring is now disabled:

> service mon_statd status
... mon_procd is not running
... 

• To restart the daemon and re-read the configuration file:

> service mon_statd restart
... stopping mon_procd: [ OK ]
starting mon_procd: [ OK ]
... 

Examples of command-line use

• To start mon_procd with default setting:

> mon_procd

• To start mon_procd with a sampling interval of 30 seconds:

> mon_procd -i 30

• To start mon_procd and have it run in the foreground:

> mon_procd -a

• To start mon_procd with a sampling interval of 45 seconds and have it run in the foreground:

> mon_procd -a -i 45

Usage

Processing monitor data

The mon_procd daemon writes system summary information and information of each process for up to 100 processes currently being managed by a Linux guest to the z/VM monitor stream. At the time of the sample interval, one sample monitor record is written for system summary data, then one sample monitor record is written for each process for up to 100 processes currently being managed by the Linux guest. If more than 100 processes exist in a Linux guest system at a given time, processes are sorted by the sum of CPU and memory usage percentage values and only the top 100 processes' data is written to the z/VM monitor stream.

The monitor data in each record contains a header, procd_hdr, followed by summary data if the type in the field “record number” of the 16-byte product ID is a summary type or process data if the type in the field “record number” of the 16-byte product ID is a process type. The following is the format of system summary data passed to the z/VM monitor stream.
### Table 52. System summary data format

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__u64</td>
<td>time_stamp</td>
<td>Time at which the process data was sampled.</td>
</tr>
<tr>
<td>__u16</td>
<td>data_len</td>
<td>Length of data following this proc_hdr.</td>
</tr>
<tr>
<td>__u16</td>
<td>data_offset</td>
<td>Offset from start of proc_hdr to start of process data.</td>
</tr>
<tr>
<td>__u64</td>
<td>uptime</td>
<td>Uptime of the Linux guest system.</td>
</tr>
<tr>
<td>__u32</td>
<td>users</td>
<td>Number of users on the Linux guest system.</td>
</tr>
<tr>
<td>char[6]</td>
<td>loadavg_1</td>
<td>Load average over the last one minute.</td>
</tr>
<tr>
<td>char[6]</td>
<td>loadavg_5</td>
<td>Load average over the last five minutes.</td>
</tr>
<tr>
<td>char[6]</td>
<td>loadavg_15</td>
<td>Load average over the last 15 minutes.</td>
</tr>
<tr>
<td>__u32</td>
<td>task_total</td>
<td>Total number of tasks on the Linux guest system.</td>
</tr>
<tr>
<td>__u32</td>
<td>task_running</td>
<td>Number of running tasks.</td>
</tr>
<tr>
<td>__u32</td>
<td>task.sleeping</td>
<td>Number of sleeping tasks.</td>
</tr>
<tr>
<td>__u32</td>
<td>task_stopped</td>
<td>Number of stopped tasks.</td>
</tr>
<tr>
<td>__u32</td>
<td>task_zombie</td>
<td>Number of zombie tasks.</td>
</tr>
<tr>
<td>__u64</td>
<td>num_cpus</td>
<td>Number of CPUs.</td>
</tr>
<tr>
<td>__u16</td>
<td>puser</td>
<td>A number representing (100 * percentage of total CPU time used for normal processes executing in user mode).</td>
</tr>
<tr>
<td>__u16</td>
<td>pnice</td>
<td>A number representing (100 * percentage of total CPU time used for niced processes executing in user mode).</td>
</tr>
<tr>
<td>__u16</td>
<td>psystem</td>
<td>A number representing (100 * percentage of total CPU time used for processes executing in kernel mode).</td>
</tr>
<tr>
<td>__u16</td>
<td>pidle</td>
<td>A number representing (100 * percentage of total CPU idle time).</td>
</tr>
<tr>
<td>__u16</td>
<td>piowait</td>
<td>A number representing (100 * percentage of total CPU time used for I/O wait).</td>
</tr>
<tr>
<td>__u16</td>
<td>pirq</td>
<td>A number representing (100 * percentage of total CPU time used for interrupts).</td>
</tr>
<tr>
<td>__u16</td>
<td>psoftirq</td>
<td>A number representing (100 * percentage of total CPU time used for softirqs).</td>
</tr>
<tr>
<td>__u16</td>
<td>psteal</td>
<td>A number representing (100 * percentage of total CPU time spent in stealing).</td>
</tr>
<tr>
<td>__u64</td>
<td>mem_total</td>
<td>Total memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>mem_used</td>
<td>Used memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>mem_free</td>
<td>Free memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>mem_buffers</td>
<td>Memory in buffer cache in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>mem_pggpin</td>
<td>Data read from disk in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>mem_pgpout</td>
<td>Data written to disk in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>swap_total</td>
<td>Total swap memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>swap_used</td>
<td>Used swap memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>swap_free</td>
<td>Free swap memory in KB.</td>
</tr>
</tbody>
</table>
The following is the format of a process information data passed to the z/VM monitor stream.

### Table 53. Data format passed to the z/VM monitor stream

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_u64</td>
<td>time_stamp</td>
<td>Time at which the process data was sampled.</td>
</tr>
<tr>
<td>_u16</td>
<td>data_len</td>
<td>Length of data following this procd_hdr.</td>
</tr>
<tr>
<td>_u16</td>
<td>data_offset</td>
<td>Offset from start of procd_hdr to start of process data.</td>
</tr>
<tr>
<td>_u32</td>
<td>pid</td>
<td>ID of the process.</td>
</tr>
<tr>
<td>_u32</td>
<td>ppid</td>
<td>ID of the process parent.</td>
</tr>
<tr>
<td>_u32</td>
<td>euid</td>
<td>Effective user ID of the process owner.</td>
</tr>
<tr>
<td>_u16</td>
<td>tty</td>
<td>Device number of the controlling terminal or 0.</td>
</tr>
<tr>
<td>_s16</td>
<td>priority</td>
<td>Priority of the process.</td>
</tr>
<tr>
<td>_s16</td>
<td>nice</td>
<td>Nice value of the process.</td>
</tr>
<tr>
<td>_u32</td>
<td>processor</td>
<td>Last used processor.</td>
</tr>
<tr>
<td>_u16</td>
<td>pcpu</td>
<td>A number representing (100 * percentage of the elapsed cpu time used by the process since last sampling).</td>
</tr>
<tr>
<td>_u16</td>
<td>pmem</td>
<td>A number representing (100 * percentage of physical memory used by the process).</td>
</tr>
<tr>
<td>_u64</td>
<td>total_time</td>
<td>Total cpu time the process has used.</td>
</tr>
<tr>
<td>_u64</td>
<td>ctotal_time</td>
<td>Total cpu time the process and its dead children has used.</td>
</tr>
<tr>
<td>_u64</td>
<td>size</td>
<td>Total virtual memory used by the task in KB.</td>
</tr>
<tr>
<td>_u64</td>
<td>swap</td>
<td>Swapped out portion of the virtual memory in KB.</td>
</tr>
<tr>
<td>_u64</td>
<td>resident</td>
<td>Non-swapped physical memory used by the task in KB.</td>
</tr>
<tr>
<td>_u64</td>
<td>trs</td>
<td>Physical memory devoted to executable code in KB.</td>
</tr>
<tr>
<td>_u64</td>
<td>drs</td>
<td>Physical memory devoted to other than executable code in KB.</td>
</tr>
<tr>
<td>_u64</td>
<td>share</td>
<td>Shared memory used by the task in KB.</td>
</tr>
<tr>
<td>_u64</td>
<td>dt</td>
<td>Dirty page count.</td>
</tr>
<tr>
<td>_u64</td>
<td>maj_flt</td>
<td>Number of major page faults occurred for the process.</td>
</tr>
<tr>
<td>char</td>
<td>state</td>
<td>Status of the process.</td>
</tr>
<tr>
<td>_u32</td>
<td>flags</td>
<td>The process current scheduling flags.</td>
</tr>
<tr>
<td>_u16</td>
<td>ruser_len</td>
<td>Length of real user name of the process owner and should not be larger than 64.</td>
</tr>
<tr>
<td>char[ruser_len]</td>
<td>ruser</td>
<td>Real user name of the process owner. If the name is longer than 64, the name is truncated to the length 64.</td>
</tr>
<tr>
<td>_u16</td>
<td>euser_len</td>
<td>Length of effective user name of the process owner and should not be larger than 64.</td>
</tr>
</tbody>
</table>
### Table 53. Data format passed to the z/VM monitor stream (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char[euser_len]</td>
<td>euser</td>
<td>Effective user name of the process owner. If the name is longer than 64, the name is truncated to the length 64.</td>
</tr>
<tr>
<td>__u16</td>
<td>egroup_len</td>
<td>Length of effective group name of the process owner and should not be larger than 64.</td>
</tr>
<tr>
<td>char[egroup_len]</td>
<td>egroup</td>
<td>Effective group name of the process owner. If the name is longer than 64, the name is truncated to the length 64.</td>
</tr>
<tr>
<td>__u16</td>
<td>wchan_len</td>
<td>Length of sleeping in function's name and should not be larger than 64.</td>
</tr>
<tr>
<td>char[wchan_len]</td>
<td>wchan_name</td>
<td>Name of sleeping in function or '-'. If the name is longer than 64, the name is truncated to the length 64.</td>
</tr>
<tr>
<td>__u16</td>
<td>cmd_len</td>
<td>Length of command name or program name used to start the process and should not be larger than 64.</td>
</tr>
<tr>
<td>char[cmd_len]</td>
<td>cmd</td>
<td>Command or program name used to start the process. If the name is longer than 64, the name is truncated to the length 64.</td>
</tr>
<tr>
<td>__u16</td>
<td>cmd_line_len</td>
<td>Length of command line used to start the process and should not be larger than 1024.</td>
</tr>
<tr>
<td>char[cmd_line_len]</td>
<td>cmd_line</td>
<td>Command line used to start the process. If the name is longer than 1024, the name is truncated to the length 1024.</td>
</tr>
</tbody>
</table>

The following data structures map the fixed-length portion of the record above. See /s390-tools/mon_tools/mon_procd.h for these mappings.

```c
struct procd hdr {
    __u64 time_stamp;
    __u16 data_len;
    __u16 data_offset;
}__attribute__((packed));

struct proc_sum_t {
    __u64 uptime;
    __u32 users;
    char loadavg_1[6];
    char loadavg_5[6];
    char loadavg_15[6];
    struct task_sum_t task;
    struct cpu_t cpu;
    struct mem_t mem;
    struct swap_t swap;
}__attribute__((packed));

struct task_sum_t {
    __u32 total;
    __u32 running;
    __u32 sleeping;
    __u32 stopped;
    __u32 zombie;
};

struct mem_t {
    __u64 total;
    __u64 used;
    __u64 free;
    __u64 buffers;
    __u64 pggpin;
    __u64 ppggout;
};
```
struct swap_t {
    __u64 total;
    __u64 used;
    __u64 free;
    __u64 cached;
    __u64 pswpin;
    __u64 pswpout;
};

struct cpu_t {
    __u32 num_cpus;
    __u16 puser;
    __u16 pnice;
    __u16 psystem;
    __u16 pidle;
    __u16 piowait;
    __u16 pirq;
    __u16 psoftirq;
    __u16 psteal;
};

struct task_t {
    __u32 pid;
    __u32 ppid;
    __u32 euid;
    __u16 tty;
    __s16 priority;
    __s16 nice;
    __u32 processor;
    __u16 pcpu;
    __u16 pmem;
    __u64 total_time;
    __u64 ctotal_time;
    __u64 size;
    __u64 swap;
    __u64 resident;
    __u64 trs;
    __u64 drs;
    __u64 share;
    __u64 dt;
    __u64 majflt;
    char state;
    __u32 flags;
} __attribute__((packed));

The time_stamp should be used to correlate all process information that were sampled in a given interval.

**Note:** The data length field (data_len) in the procd_hdr (not the length of the monitor record itself) should be used to determine how much data there is in the monitor record.

**Reading the monitor data**

As described in the monwriter documentation, all records written to the z/VM monitor stream begin with a product identifier. The product ID is a 16-byte structure of the form pppppppppnnvmm, where for records written by mon_procd, these values will be:

pppppppp
   is a fixed ASCII string LNXAPPL.

ff
   is the application number for mon_procd = x'0002'.

n
   is the record number as follows:
   • x'00' indicates summary data.
   • x'01' indicates task data.
**mon_procd**

**vv** is the version number = x'0000'.

**rr** is the release number, which can be used to mark different versions of process APPLDATA records.

**mm** is reserved for mon_procd and should be ignored.

**Note:** Though the mod_level field (mm) of the product ID will vary, there is no relationship between any particular mod_level and process. The mod_level field should be ignored by the reader of this monitor data.

This item uses at most 101 monitor buffer records from the monwriter device driver. Since a maximum number of buffers is set when a monwriter module is loaded, the maximum number of buffers must not be less than the sum of buffer records used by all monwriter applications.

There are many tools available to read z/VM monitor data. One such tool is the Linux monreader character device driver. See Chapter 16, "Reading z/VM monitor records" for more information about monreader.

**Further information**
- Refer to z/VM: Saved Segments Planning and Administration, SC24-6116 for general information on DCSSs.
- Refer to z/VM: CP Commands and Utilities Reference, SC24-6081 for information on the CP commands.
- Refer to z/VM: Performance, SC24-6109 for information on monitor APPLDATA.
qetharp

qetharp - Query and purge OSA and HiperSockets ARP data

Use the qetharp command to query and purge address data such as MAC and IP addresses from the ARP cache of the OSA and HiperSockets hardware. You cannot use this command in conjunction with the layer2 option. For z/VM guest LAN and VSWITCH interfaces in non-layer2 mode, note that only the --query option is supported.

Format

```
qetharp parameters

qetharp [options] <interface>

Options:
- -q or --query
  Shows the address resolution protocol (ARP) information found in the ARP cache of the OSA or HiperSockets, which depends on <interface>. If it is an OSA device, it shows the ARP entries stored in the OSA feature's ARP cache, otherwise, the ones from the HiperSockets ARP cache. If the IP address is an IPv4 address, qetharp tries to determine the symbolic host name. If it fails, the IP address will be shown. In case of IPv6, there is currently no attempt to determine host names, so that the IP address will be shown directly.

- -n or --numeric
  Shows numeric addresses instead of trying to determine symbolic host names. This option can only be used in conjunction with the -q option.

- -c or --compact
  Limits the output to numeric addresses only. This option can only be used in conjunction with the -q option.

- -a <interface>
  Adds a static ARP entry to the OSA adapter card.

- -i <ip_address>
  IP address to be added to the OSA adapter card.

- -d <interface>
  Deletes a static ARP entry from the OSA adapter card.

- -m <mac_address>
  MAC address to be added to the OSA adapter card.

- -p <interface>
  Flushes the ARP cache of the OSA, causing the hardware to regenerate the addresses. This option works only with OSA devices. qetharp returns immediately.
```

The meanings of the parameters of this command are as follows:

- **-q or --query**
  Shows the address resolution protocol (ARP) information found in the ARP cache of the OSA or HiperSockets, which depends on `<interface>`. If it is an OSA device, it shows the ARP entries stored in the OSA feature’s ARP cache, otherwise, the ones from the HiperSockets ARP cache. If the IP address is an IPv4 address, qetharp tries to determine the symbolic host name. If it fails, the IP address will be shown. In case of IPv6, there is currently no attempt to determine host names, so that the IP address will be shown directly.

- **-n or --numeric**
  Shows numeric addresses instead of trying to determine symbolic host names. This option can only be used in conjunction with the -q option.

- **-c or --compact**
  Limits the output to numeric addresses only. This option can only be used in conjunction with the -q option.

- `<interface>`
  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 --version
  Shows version information and exits

-h or --help
  Shows usage information and exits

Examples

- Show all ARP entries of the OSA defined as eth0:
  ```
  # qetharp -q eth0
  ```

- Show all ARP entries of the OSA defined as eth0, without resolving host names:
  ```
  # qetharp -nq eth0
  ```

- Flush the OSA's ARP cache for eth0:
  ```
  # qetharp -p eth0
  ```

- Add a static entry for eth0 and IP address 1.2.3.4 to the OSA’s ARP cache, using MAC address aa:bb:cc:dd:ee:ff:
  ```
  # qetharp -a eth0 -i 1.2.3.4 -m aa:bb:cc:dd:ee:ff
  ```

- Delete the static entry for eth0 and IP address 1.2.3.4 from the OSA’s ARP cache, using MAC address aa:bb:cc:dd:ee:ff:
  ```
  # qetharp -d eth0 -i 1.2.3.4
  ```
qethconf

qethconf - Configure qeth devices

The qethconf configuration tool is a bash shell script that simplifies configuring qeth devices (see Chapter 7, “qeth device driver for OSA-Express (QDIO) and HiperSockets,” on page 81) 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 qethconf command has these function keywords:

ipa
  Configure qeth for IP address takeover (IPA).

vipa
  Configure qeth for virtual IP address (VIPA).

parp or rxip
  Configure qeth for proxy ARP.

The qethconf command has these action keywords:

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_addr>
IP address. Can be specified in one of these formats:
- IP version 4 format, for example, 192.168.10.38
- IP version 6 format, for example, FE80::1:800:23e7:f5db
- 8- or 32-character hexadecimals prefixed with -x, for example, -xc0a80a26

<mask_bits>
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.

-h or --help
displays help information.

-v or --version
displays version information.

Examples

- List existing proxy ARP definitions:

```
# qethconf parp list
parp add 1.2.3.4 eth0
```

- Assume responsibility for packages destined for 1.2.3.5:

```
# qethconf parp add 1.2.3.5 eth0
qethconf: Added 1.2.3.5 to /sys/class/net/eth0/device/rxip/add4.
quethconf: 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.
quethconf: 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

# qethconf vipa add 10.99.3.3 eth1
qethconf: Added 10.99.3.3 to /sys/class/net/eth1/device/vipa/add4.
qethconf: Use "qethconf vipa list" to check for the result

Display the new VIPA definitions:

# qethconf vipa list
vipa add 10.99.3.3 eth1

• List all existing IPA, VIPA, and proxy ARP definitions.

# qethconf list_all
parp add 1.2.3.4 eth0
parp add 1.2.3.5 eth0
ipa add 192.168.10.0/24 eth0
vipa add 10.99.3.3 eth1
scsi_logging_level - Set and get the SCSI logging level

This command is used to create, set, or get the SCSI logging level.

The SCSI logging feature is controlled by a 32 bit value – the SCSI logging level. This value is divided into 3-bit fields describing the log level of a specific log area. Due to the 3-bit subdivision, setting levels or interpreting the meaning of current levels of the SCSI logging feature is not trivial. The scsi_logging_level script helps with both tasks.

Format

```
scsi_logging_level [options] [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 `--l owlevel <level>`  
specifies SCSI_LOG_LLQUEUE and SCSI_LOG_LLCOMPLETE.
scsi_logging_level

--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.

```bash
#> 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:

```bash
#> scsi_logging_level -s 3
```

404  Device Drivers, Features, and Commands - SUSE Linux Enterprise Server 11 for System z
This command sets \texttt{SCSI\_LOG\_HLQUEUE}=3, \texttt{SCSI\_LOG\_HLCOMPLETE}=2 and assigns all other \texttt{SCSI\_LOG} fields the value 1.
**snipl**

---

**snipl – Simple network IPL (Linux image control for LPAR and z/VM)**

*snipl* (simple network 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**.

**Note:** Be aware that incautious use of *snipl* can result in loss of data.

### LPAR mode

In LPAR mode, *snipl* allows you to:

- *Load* (IPL) an LPAR from a device, for example, a DASD device or a SCSI device.
- *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, which is obtainable from the following Web site:


### 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. 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 z/VM host, the VSMSERVE server must be configured and the vsmapi service must be registered on the target z/VM host. Also, 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, SC24-6122 obtainable from the following Web site:

[ibm.com/vm/](ibm.com/vm/)

Usage

Command line syntax (LPAR mode)

```
snipl [options] [parameters]
```

Options:
- `-f <defaultfile>`
- `-f <filename>`
- `--timeout <timeout>`
- `--profilename <defaultprofile>`
- `--msgtimeout <interval>`
- `--msgfilename <filename>`
- `-l loadparms`
- `-x`
- `-i`
- `-s` scsiloadparms

Parameters:
- `<image_name>`
- `<ip_address>`
- `<community>`
- `<ip_address>`
- `<image_name>`

Notes:
1. See description of the `-f` option.
2. See description of the `--profilename` option.
snipl command (LPAR mode) cont.

loadparms:

- `--load_timeout <timeout>`
- `--noclear`
- `--storestatus`

snipl command (LPAR mode) cont.

scsiloadparms:

- `--wwpn_scsiload <portname>`
- `--lun_scsiload <unitnumber>`
- `--bps_scsiload <selector>`
- `--ossparms_scsiload <string>`
- `--bootrecord_scsiload <hexaddress>`

Command line syntax (VM mode)

snipl command (VM mode)

- `snipl`
- `<image_name>`
- `-V <ip_address>`
- `-u <userid>`
- `-p <password>`
- `-f <filename>`
- `-a`
- `-d`
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 z/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.

-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 z/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.

--timeout <timeout>
LPAR only: Specifies the timeout in milliseconds for general management API calls. The default is 60000 ms.

-a or --activate
Issues an activate command for the targeted LPAR or z/VM guest.

--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.
-d or --deactivate
  Issues a deactivate command for the target LPAR or z/VM guest.

-F or --force
  Forces the image operation.
  • 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.

-r or --reset
  Issues a reset command for the targeted LPAR(s) or z/VM guest(s).

-x or --listimages
  Lists all available images for the specified server.
  • For z/VM this may be specified with image, server, server+user or image+user according to the uniqueness in the configuration file. In case of z/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.

-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.

--msgtimeout <interval>
  LPAR only: In conjunction with --dialog this option specifies the interval in milliseconds for management API calls that retrieve operating system messages. The default value is 5000 ms.

-M <filename> or --msgfilename <filename>
  LPAR only: In conjunction with --dialog this option specifies the name of a file to which the operating system messages are written as well as to stdout. If none is given, operating system messages are written to stdout only.

-l or --load
  LPAR only: Issues a load command for the target LPAR.

-A <loadaddress> or --address_load <loadaddress>
  LPAR only: In conjunction with --load and --scsiload this option specifies the load address as four 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 and --scsiload 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.

-s or --scsiload
    LPAR only: Issues a SCSI load command for the target LPAR.

--wwpn_scsiload <portname>
    LPAR only: Specifies the worldwide port name (WWPN) to be used for scsiload. It identifies the Fibre Channel port of the SCSI target device and consists of 16 hexadecimal characters. Smaller specifications are padded with zeroes at the end. If none is given, the worldwide port name of the previous scsiload is used.

--lun_scsiload <unitnumber>
    LPAR only: Specifies the logical unit number (LUN) defined by FCP to be used for scsiload. It consists of 16 hexadecimal characters. Smaller specifications are padded with zeroes at the end. If none is given, the logical unit number of the previous scsiload is used.

--bps_scsiload <selector>
    LPAR only: Specifies the boot program selector to be used for scsiload. It identifies the program to load from the FCP-load device. Valid values range from 0 to 30. This option provides the possibility of having up to 31 different boot configurations on a single SCSI disk device. If none is given, the boot program selector of the previous scsiload is used.

--ossparms_scsiload <string>
    LPAR only: Specifies an operating-system specific load parameter string for scsiload. The field contains a variable number of characters to be used by the program that is loaded during SCSI IPL. This information is given to the IPLed operating system and ignored by the machine loader. The IPLed operating system must support this. If none is given, the parameter string of the previous scsiload is used.

--bootrecord_scsiload <hexaddress>
    LPAR only: Specifies the boot record logical block address for scsiload, if your file system supports dual boot or booting from one of multiple partitions. It consists of 16 hexadecimal characters. Smaller specifications are padded with zeroes at the end. If none is given, the address of the previous scsiload is used.

-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. z/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.

An alias name to specify a hostname for an image can optionally be specified using the slash-character as a separator in the image name. Both are valid:

image = <imagename>
image = <imagename>/<alias>

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>/<alias>
image = <imagename>/<alias>
Server = <IP-address>
type = <host-type>
user = <username>
password = <password>
image = <imagename>
image = <imagename>/<alias>
image = <imagename>/<alias>
image = <imagename>

Blanks and \n are separators. The keywords are not case-sensitive.

snipl command examples

**LPAR mode - Activate:**

```bash
# snipl LPARLN1 -L 9.164.70.100 -a -P
Enter password: Warning : No default configuration file could be found/opened.
processing......
LPARLN1: acknowledged.
```

**LPAR mode - Load using configuration file:**

```bash
# snipl LPARLN1 -f xcfg -l -A 5119
Server 9.99.99.99 from config file xcfg is used
processing......
LPARLN1: acknowledged.
```

**LPAR mode - SCSI load using configuration file:**

```bash
snipl LPARLN1 -s -A 5000 --wwpn_scsiload 500507630303c562 --lun_scsiload 4010490000000000
Server 9.99.99.99 from config file /etc/snipl.conf is used
processing...
LPARLN1: acknowledged.
```

**z/VM mode - Activate using configuration file:**

```bash
# 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
**snipl**

LPARLNX1: not acknowledged – command was not successful – rc is 135921664

are described in “Appendix B” of the HWMCAPI 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/](http://linux-ha.org/).

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

Chapter 40. Useful Linux commands 413
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 411.

**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 this publication:
  
  *System z Application Programming Interfaces, SB10-7030*

  You can obtain this publication from the following Web site: [ibm.com/servers/resourcelink/](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, SC24–6122*

  You can obtain this publication from the following Web site: [ibm.com/vm/](ibm.com/vm/)

**Using STONITH:** The following examples show how you can invoke STONITH.

- Sample call that passes a configuration file:
  
  ```
  stonith -t lic_vps -p "snipl_file snipl.conf" -T reset t293043
  ```

- Equivalent call passing a parameter string:
  
  ```
  stonith -t lic_vps -p "snipl_param server=boet2930,type=vm,user=t2930043,password=passw0rd,image=t2930043" -T reset t2930043
  ```
tape390_crypt - manage tape encryption

Use this command to enable and disable tape encryption for a channel attached tape device, as well as to specify key encrypting keys (KEK) by means of labels or hashes.

For 3592 tape devices, it is possible to write data in an encrypted format. The encryption keys are stored on an encryption key manager (EKM) server, which can run on any machine with TCP/IP and Java™ support. The EKM communicates with the tape drive over the tape control unit using TCP/IP. The control unit acts as a proxy and forwards the traffic between the tape drive and the EKM. This type of setup is called "out of band" control-unit based encryption.

The EKM creates a data key that encrypts data. The data key itself is encrypted with KEKs and is stored in so called external encrypted data keys (EEDKs) on the tape medium.

You can store up to two EEDKs on the tape medium. The advantage of having two EEDKs is that one EEDK can contain a locally available KEK and the other can contain the public KEK of the location or company to where the tape is to be transferred. Then the tape medium can be read in both locations.

When the tape device is mounted, the tape drive sends the EEDKs to the EKM, which tries to unwrap one of the two EEDKs and sends back the extracted data key to the tape drive.

Linux can address KEKs by specifying either hashes or labels. Hashes and labels are stored in the EEDKs.

Note: If a tape has been encrypted, it cannot be used for IPL.

Prerequisites
To use tape encryption you need:
- A 3592 crypto-enabled tape device and control unit configured as system-managed encryption.
- A crypto-enabled 3590 channel-attached tape device driver. See Chapter 5, "Channel-attached tape device driver," on page 65.
tape390_crypt

Format

```
tape390_crypt syntax

-tape390_crypt -q -e on off

Keys:

(1) -k <value> <char>label -d :
     <char>hash -d -f

Notes:

1. The -k or --key operand can be specified maximally twice.
```

where:

- **-q or --query**
  displays information on the tape's encryption status. If encryption is active
  and the medium is encrypted, additional information about the encryption
  keys is displayed.

- **-e or --encryption**
  sets tape encryption on or off.

- **-k or --key**
  sets tape encryption keys. You can only specify the -k option if the tape
  medium is loaded and rewound. While processing the -k option, the tape
  medium is initialized and all previous data contained on the tape medium is
  lost.

You can specify the -k option twice, because the tape medium can store
two EEDKs. If you specify the -k option once, two identical EEDKs are
stored.

- **<value>**
  specifies the key encrypting key (KEK), which can be up to 64
  characters long. The keywords label or hash specify how the KEK in
  <value> is to be stored on the tape medium. The default store type is
  label.

- **-d or --delimiter**
  specifies the character that separates the KEK in <value> from the store
  type (label or hash). The default delimiter is ":" (colon).

- **<char>**
  is a character separating the KEK in <value> from the store type (label
  or hash).

- **-f or --force**
  specifies that no prompt message is to be issued before writing the KEK
  information and initializing the tape medium.
<node>
specifies the device node of the tape device.

-h or --help
prints help text. For more information, enter the command man
tape390_crypt.

-v or --version
prints information about the version.

Examples
This example shows a query of tape device /dev/ntibm0. Initially, encryption for this
device is off. Encryption is then turned on, and the status is queried again.

```
tape390_crypt -q /dev/ntibm0
ENCRIPTION: OFF
MEDIUM: NOT ENCRYPTED
```
```
tape390_crypt -e on /dev/ntibm0
```
```
tape390_crypt -q /dev/ntibm0
ENCRIPTION: ON
MEDIUM: NOT ENCRYPTED
```

Then two keys are set, one in label format and one in hash format. The status is
queried and there is now additional output for the keys.

```
tape390_crypt -k my_first_key:label -k my_second_key:hash /dev/ntibm0
<---- ATTENTION! <----
All data on tape /dev/ntibm0 will be lost.
Type "yes" to continue: yes
SUCCESS: key information set.
```
```
tape390_crypt -q /dev/ntibm0
ENCRIPTION: ON
MEDIUM: ENCRYPTED
KEY1:
value: my_first_key
type: label
ontape: label
KEY2:
value: my_second_key
type: label
ontape: hash
```

Usage scenarios
The following scenarios illustrate the most common use of tape encryption. In all
examples /dev/ntibm0 is used as the tape device.

Using default keys for encryption:
1. Load the cartridge. If the cartridge is already loaded:
   - Switch encryption off:
     tape390_crypt -e off /dev/ntibm
   - Rewind:
     mt -f /dev/ntibm0 rewind
2. Switch encryption on:
   tape390_crypt -e on /dev/ntibm0
3. Write data.

Using specific keys for encryption:
**tape390_crypt**

1. Load the cartridge. If the cartridge is already loaded, rewind:
   
   ```
   mt -f /dev/ntibm0 rewind
   ```
2. Switch encryption on:
   
   ```
   tape390_crypt -e on /dev/ntibm0
   ```
3. Set new keys:
   
   ```
   tape390_crypt -k key1 -k key2 /dev/ntibm0
   ```
4. Write data.

**Writing unencrypted data:**

1. Load the cartridge. If the cartridge is already loaded, rewind:
   
   ```
   mt -f /dev/ntibm0 rewind
   ```
2. If encryption is on, switch encryption off:
   
   ```
   tape390_crypt -e off /dev/ntibm0
   ```
3. Write data.

**Appending new files to an encrypted cartridge:**

1. Load the cartridge
2. Switch encryption on:
   
   ```
   tape390_crypt -e on /dev/ntibm0
   ```
3. Position the tape.
4. Write data.

**Reading an encrypted tape:**

1. Load the cartridge
2. Switch encryption on:
   
   ```
   tape390_crypt -e on /dev/ntibm0
   ```
3. Read data.
tape390_display - display messages on tape devices and load tapes

This command is used to display messages on a physical tape device’s display unit, optionally in conjunction with loading a tape.

Format

```
tape390_display syntax
```

```
-t standard
-t load
-t unload
-t reload
-t noop
```

Notes:

1. With reload, two messages must be specified and option -b is not permitted.

where:

-t or --type
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.
tape390_display

<message2>
is a second message to be displayed alternately with the first, at 2 second intervals. The message can be up to 8 byte.

<node>
is a device node of the target tape device.

-q or --quiet
suppresses all error messages.

-h or --help
prints help text.

-v or --version
prints information about the version.

Notes:
1. Symbols that can be displayed include:

   Alphabetic characters:
   A through Z (uppercase only) and spaces. Lowercase letters are converted to uppercase.

   Numeric characters:
   0 1 2 3 4 5 6 7 8 9

   Special characters:
   @ $ # , . / ( ) * & + - = % _ < > ?
   The following are included in the 3490 hardware reference but might not display on all devices: I C

2. If only one message is defined, it remains displayed until the tape device driver next starts to move or the message is updated.

3. If the messages contain spaces or shell-sensitive characters, they must be enclosed in quotation marks.

Examples

The following examples assume that you are using standard devices nodes and not device nodes created by udev:

• Alternately display "BACKUP" and "COMPLETE" at two second intervals until device /dev/ntibm0 processes the next tape movement command:
  tape390_display BACKUP COMPLETE /dev/ntibm0

• Display the message "REM TAPE" while a tape is in the physical tape device followed by the message "NEW TAPE" until a new tape is loaded:
  tape390_display --type reload "REM TAPE" "NEW TAPE" /dev/ntibm0

• Attempts to unload the tape and load a new tape automatically, the messages are the same as in the previous example:
  tape390_display -l -t reload "REM TAPE" "NEW TAPE" /dev/ntibm0
tunedasd - Adjust DASD performance

Use tunedasd to:

- Display and reset DASD performance statistics
- Query and set a DASD’s cache mode
- Reserve and release DASD
- Breaking the lock of a known DASD (for accessing a boxed DASD while booting Linux see "Accessing DASD by force" on page 33)

Before you start:

- You must have root permissions.
- For the performance statistics:
  - Data gathering must have been switched on by writing “on” to /proc/dasd/statistics.

Format

```
<table>
<thead>
<tr>
<th>tunedasd syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>┌─────────┬─────────┬─┐</td>
</tr>
<tr>
<td>│tunedasd │ -h     │   │</td>
</tr>
<tr>
<td>├─────────┼─────────┼─┐</td>
</tr>
<tr>
<td>│        │ -g         │   │</td>
</tr>
<tr>
<td>├─────────┼─────────┼─┐</td>
</tr>
<tr>
<td>│        │ -c &lt;mode&gt; │   │</td>
</tr>
<tr>
<td>├─────────┼─────────┼─┐</td>
</tr>
<tr>
<td>│        │ -S         │   │</td>
</tr>
<tr>
<td>├─────────┼─────────┼─┐</td>
</tr>
<tr>
<td>│        │ -L         │   │</td>
</tr>
<tr>
<td>├─────────┼─────────┼─┐</td>
</tr>
<tr>
<td>│        │ -D         │   │</td>
</tr>
<tr>
<td>├─────────┼─────────┼─┐</td>
</tr>
<tr>
<td>│        │ -P         │   │</td>
</tr>
<tr>
<td>└─────────┴─────────┴─┘</td>
</tr>
</tbody>
</table>
```

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.
tunedasd

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 --prof_item <row>
Prints the usage profile item specified by <row>. <row> can be one of:
  reqs   number of DASD I/O requests
  sects  number of 512 byte sectors
  sizes  histogram of sizes
  total  histogram of I/O times
  totsect histogram of I/O times per sector
  start  histogram of I/O time till ssch
  irq    histogram of I/O time between ssch and irq
  irqsect histogram of I/O time between ssch and irq per sector
  end    histogram of I/O time between irq and end
  queue  number of requests in the DASD internal request queue at enqueueing

-v or --version
  displays version information.

-h or --help
  displays help information.

Examples

- This example first queries the current setting for the cache mode of a DASD with device node /dev/dasdzzz and then sets it to 1 cylinder “prestage”.

  # tunedasd -g /dev/dasdzzz
  normal (0 cyl)
  # tunedasd -c prestige -n 2 /dev/dasdzzz
  Setting cache mode for device </dev/dasdzzz>...
  Done.
  # tunedasd -g /dev/dasdzzz
  prestige (2 cyl)

- In this example two device nodes are specified. The output is printed for each node in the order in which the nodes were specified.
The following command prints the usage profile of a DASD.

```
# tunedasd -P /dev/dasdzzz
```

```
19617 dasd I/O requests
with 4841336 sectors(512B each)

+1632641282565121M2M4M8M16M32M64M128M256M512M1G2G4G8G16G32G64G128G
Histogram of sizes (512B sects)
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Histogram of I/O times (microseconds)
 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 per sector
 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 time till ssch
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Histogram of I/O time between ssch and irq
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Histogram of I/O time between irq and end
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

# of req in chanq at enqueuing (1..32)
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...)
267| 18544| 224| 3| 4| 4| 0| 0| 0| 0| 0| (cont...)
0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| (cont...)
```

Chapter 40. Useful Linux commands 423
**vmcp - Send CP commands to the z/VM hypervisor**

**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 issue **vmcp** commands using the /dev/vmcp device node (see Chapter 22, "z/VM CP interface device driver," on page 217) or with the user space tool vmcp. In both cases, you must load the vmcp module.

**Before you start:**
- You must have root permissions.
- Ensure that vmcp is loaded by issuing: `modprobe vmcp`

**Format**

```
vmcp syntax
```

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
  ```
The `vmur` command provides all functions required to work with z/VM spool file queues:

**Receive**
Read data from the z/VM reader file queue. The command performs the following steps:
- Places the reader queue file to be received at the top of the queue.
- Changes the reader queue file attribute to NOHOLD.
- Closes the z/VM reader after reading the file.

**Punch or print**
Write data to the z/VM punch or printer file queue and transfer it to another user’s virtual reader, optionally on a remote z/VM node. The data is sliced up into 80-byte or 132-byte chunks (called records) and written to the punch or printer device. If the data length is not an integer multiple of 80 or 132, the last record is padded with 0x00.

**List**
Display detailed information about one or all files on the specified spool file queue.

**Purge**
Remove one or all files on the specified spool file queue.

**Order**
Position a file at the top of the specified spool file queue.

The `vmur` command provides strict serialization of all its functions other than list, which does not affect a file queue’s contents or sequence. Thus concurrent access to spool file queues is blocked in order to prevent unpredictable results or destructive conflicts.

For example, this serialization prevents a process from issuing `vmur purge -f` while another process is executing `vmur receive 1234`. However, `vmur` is not serialized against concurrent CP commands issued through `vmcp`: if one process is executing `vmur receive 1234` and another process issues `vmcp purge rdr 1234`, then the received file might be incomplete. To avoid such unwanted effects use `vmur` exclusively when working with z/VM spool file queues.

The `vmur` command detects z/VM reader queue files in:
- VMDUMP format as created by CP VMDUMP.
- NETDATA format as created by CMS SENDFILE or TSO XMIT.

**Before you start:**
- You must have root permissions.
- Ensure that `vmcp` module is loaded by issuing `modprobe vmcp`
- To use the receive, punch, and print functions, the `vmur` device driver must be loaded and the respective unit record devices must be set online.
Format

vmur syntax

```
vmur receive OptA <spoolid> -n <name> -o <outfile>
vmur punch OptB <spoolid> -n <name> -o <outfile>
vmur print OptC -r -N <name> <file>
vmur list -q rdr
vmur purge -q pun <spoolid>
vmur order -q rdr -q punt -q pr
```

OptA:
```
-f -t -b <sep>,<pad> -d /dev/vmrd-0.0.000c -d <device_node>
```

OptB:
```
-f -t -b <sep>,<pad> -d /dev/vmpun-0.0.000d -d <device_node>
```

OptC:
```
-f -t -b <sep>,<pad> -d /dev/vmprt-0.0.000e -d <device_node>
```

Where:

- **re** or **receive**
  - specifies that a file on the z/VM reader queue is to be received.

- **pun** or **punch**
  - specifies that a file is to be written to the z/VM punch queue.

- **li** or **list**
  - specifies that information on one or all files on a z/VM spool file queue is to be listed.

- **pur** or **purge**
  - specifies that one or all files on a z/VM spool file queue is to be purged.

- **or** or **order**
  - specifies that a file on a z/VM spool file queue is to be ordered, that is to be placed on top of the queue.
Note: The short forms given for receive, punch, print, list, purge, and order are the shortest forms possible. As is common in z/VM, you can use any form of these keywords that contain the minimum form. For example, vmur re, vmur rec, or vmur rece are all equivalent.

-d or --device
specifies the device node of the virtual unit record device.
- If omitted in the receive function, /dev/vmrdr-0.0.000c is assumed.
- If omitted in the punch function, /dev/vmpun-0.0.000d is assumed.
- If omitted in the print function, /dev/vmprt-0.0.000e is assumed.

-q or --queue
specifies the z/VM spool file queue to be listed, purged or ordered. If omitted, the reader file queue is assumed.

-t or --text
specifies a text file requiring EBCDIC-to-ASCII conversion (or vice versa) according to character sets IBM037 and ISO-8859-1.
- For the receive function: specifies to receive the reader file as text file, that is, perform EBCDIC-to-ASCII conversion and insert an ASCII line feed character (0x0a) for each input record read from the z/VM reader. Trailing EBCDIC blanks (0x40) in the input records are stripped.
- For the punch or print function: specifies to punch the input file as text file, that is, perform ASCII-to-EBCDIC conversion and pad each input line with trailing blanks to fill up the record. The record length is 80 for a punch and 132 for a printer. If an input line length exceeds 80 or 132 for punch or print, respectively, an error message is issued.

The --text and the --blocked attributes are mutually exclusive.

-b <sep, pad> or --blocked <sep, pad>
specifies that the file has to be received or written using the blocked mode. As parameter for the -b option, specify the hex codes of the separator and the padding character. Example:
  --blocked 0x55,0x40

Use this option if you need to use character sets other than IBM037 and ISO-8859-1 for conversion.
- For the receive function: All trailing padding characters are removed from the end of each record read from the virtual reader and the separator character is inserted afterwards. The receive function's output can be piped to `iconv` using the appropriate character sets. Example:

```bash
# vmur rec 7 -b 0x25,0x40 -O | iconv -f EBCDIC-US -t ISO-8859-1 > myfile
```

- For the punch or print function: The separator is used to identify the line end character of the file to punch or print. If a line has less characters than the record length of the used unit record device, the residual of the record is filled up with the specified padding byte. If a line exceeds the record size, an error is printed. Example:

```bash
# iconv test.txt -f ISO-8859-1 -t EBCDIC-US | vmur pun -b 0x25,0x40 -N test
```

-r or --rdr
specifies that the punch or print file is to be transferred to a reader.
-u <user> or --user <user>
specifies the z/VM user ID to whose reader the data is to be transferred. If user is omitted, the data is transferred to your own machine's reader. The use option is only valid if the -r option has been specified.

-n <node> or --node <node>
specifies the z/VM node ID of the z/VM system to which the data is to be transferred. Remote Spooling Communications Subsystem (RSCS) must be installed on the z/VM systems and the specified node ID must be defined in the RSCS machine's configuration file. If node is omitted, the data is transferred to the specified user at your local z/VM system. The node option is only valid, if the -u option has been specified.

-f or --force
suppresses confirmation messages.

-1 or --stdout
specifies that the reader file's contents are written to standard output.

-N or --name
specifies a name and, optionally, a type for the z/VM spool file to be created by the punch or print option. To specify a type, after the file name enter a period followed by the type. For example:

```
# vmur pun -r /boot/parmfile -N myname.mytype
```

Both the name and the type must comply to z/VM file name rules (that is, must be one to eight characters long).

If omitted, the Linux input file name (if any) is used instead. Use the --force option to enforce valid spool file names and types.

-H or --hold
specifies that the spool file to be received remains in the reader queue. If omitted, the spool file is purged.

<spoolid>
denotes the spool ID that identifies a file belonging to z/VM's reader, punch or printer queue. The spool ID must be a decimal number in the range 0-9999. If the spool ID is omitted in the list or purge function, all files on the respective queue are listed or purged.

<outfile>
specifies the name of the output file to receive the reader spool file's data. If both <outfile> and --stdout are omitted, name and type of the spool file to be received (see the NAME and TYPE columns in vmur list output) are taken to build the output file <name>,<type>. If the spool file to be received is an unnamed file, an error message is issued.

<file>
specifies the file data to be punched or printed. If file is omitted, the data is read from standard input.
vmur

-h or --help
displays short information on command usage. To view the man page, issue
man vmur.

-v or --version
displays version information.

Examples

This section illustrates common scenarios for unit record devices. In all examples
the following device nodes are used:
- /dev/vmrdr-0.0.000c as virtual reader.
- /dev/vmpun-0.0.000d as virtual punch.

Besides the vmur device driver and the vmur command these scenarios require
that:
- The vmcp module must be loaded.
- The vmcp and vmconvert commands from the s390-tools package must be
  available.

Produce and read Linux guest machine dump
1. Produce guest machine dump:

   # vmcp vmdump /* Patience required ... */

2. Find spool ID of VMDUMP spool file in the output of the vmur li
   command:

   # vmur li

```
ORIGINID FILE CLASS RECORDS CPY HOLD DATE TIME NAME TYPE DIST
T6360025 0463 V DMP 00020222 001 NONE 06/11 15:07:42 VMDUMP FILE T6360025
```

In the example above the required VMDUMP file spool ID is 463.

3. Move vmdump file to top of reader queue with the vmur order
   command:

   # vmur or 463

4. Read and convert the vmdump file to a file on the Linux file system in the
   current working directory:

   # vmconvert /dev/vmrdr-0.0.000c linux_dump

5. Close virtual reader:

   # vmcp cl c

Alternatively to the last three steps you can first receive and then convert the
received file:

   # vmur re 463 vmdump
   # vmconvert vmdump linux_dump
Log and read Linux guest machine console

1. Begin console spooling:
   ```sh
   # vmcp sp cons start
   ```

2. Produce output to VM console (for example, with CP TRACE).

3. Close the console file and transfer it to the reader queue, find the spool ID behind the FILE keyword in the corresponding CP message. In the example below, the spool ID is 398:
   ```sh
   # vmcp sp cons clo \* rdr
   
   RDR FILE 0398 SENT FROM T6360025 CON WAS 0398 RECS 1872 CPY 001 T NOHOLD NOKEEP
   ```

4. Read the guest machine console file into a file on the Linux file system in the current working directory:
   ```sh
   # vmur re -t 398 linux_cons
   ```

Prepare z/VM reader to IPL Linux image

1. Send parmfile to VM punch and transfer it to the reader queue:
   ```sh
   # vmur pun -r /boot/parmfile
   ```

2. Find the parmfile spool ID in message:
   ```
   Reader file with spoolid 0465 created.
   ```

3. Send image to VM punch and transfer it to reader queue:
   ```sh
   # vmur pun -r /boot/vmlinuz -N image
   
   Find the image spool ID in message:
   ```
   ```
   Reader file with spoolid 0466 created.
   ```

4. (Optional) Check the spool IDs of image and parmfile in the reader queue:
   ```sh
   # vmur li
   ```
   ```
   ORIGINID FILE CLASS RECORDS CPY HOLD DATE TIME NAME TYPE DIST
   T6360025 0463 V DMP 00020222 001 NONE 06/11 15:07:42 VMDDMP FILE T6360025
   T6360025 0465 A PUN 00000002 001 NONE 06/11 15:30:31 parmfile T6360025
   T6360025 0466 A PUN 00065200 001 NONE 06/11 15:30:52 image T6360025
   ```

   In this example the parmfile spool ID is 465 and the image spool ID is 466.

5. Move image to first and parmfile to the second position in the reader queue:
   ```sh
   # vmur or 465
   # vmur or 466
   ```

6. Prepare re-IPL from the VM reader:
   ```sh
   # echo 0.0.000c > /sys/firmware/reipl/ccw/device
   ```

7. Boot the Linux image in the VM reader:
Send VM PROFILE EXEC to different Linux guest machines

This scenario describes how to send a file called vmprofile.exe from the Linux file system to other Linux guest machines. The file contains the VM PROFILE EXEC file.

1. Send vmprofile.exe (configuration file containing CP and CMS commands to customize a virtual machine) to two other Linux guest machines: user ID t2930020 at node ID boet2930 and user ID t6360025 at node ID boet6360.

   vmur pun vmprofile.exe -t -r -u t2930020 -n boet2930 -N PROFILE
   vmur pun vmprofile.exe -t -r -u t6360025 -n boet6360 -N PROFILE

2. Logon to t2930020 at boet2930, IPL CMS, and issue the CP command:

   QUERY RDR ALL

   Find the spool ID of PROFILE in the FILE column.

3. Issue the CMS command:

   RECEIVE <spoolid> PROFILE EXEC A (REPL

4. Logon to t6360025 at boet6360, IPL CMS, and issue the CP command:

   QUERY RDR ALL

   Find the spool ID of PROFILE in the FILE column.

5. Issue the CMS command:

   RECEIVE <spoolid> PROFILE EXEC A (REPL

Send VSE job to VSE guest machine

This scenario describes how to send a file containing a VSE job to a VSE guest machine.

To send lserv.job (file containing VSE job control and JECL commands) to user ID vseuser at node ID vse01sys, issue:

   vmur pun lserv.job -t -r -u vseuser -n vse01sys -N LSERV
Chapter 41. 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`
- `maxcpus`
- `mem`
- `noinitrd`
- `possible_cpus`
- `ramdisk_size`
- `rd`
- `root`
- `vmhalt`
- `vmpanic`
- `vmpoff`
- `vmreboot`
additional_cpus

Usage

Note: The additional_cpus parameter is available with Linux kernel versions prior to kernel 2.6.24. In later kernel versions new CPUs can be added to the system without specifying any kernel parameters. To limit the number of CPUs that can be added use the possible_cpus parameter.

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 438 and “possible_cpus” on page 441.

Format

```
additional_cpus=syntax
   ^add<=>tional_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.
- cio_ignore might hide essential devices such as the console under z/VM. The console is typically device number 0.0.0009.

See also "Changing the list of devices to be ignored" on page 436.

Format

cio_ignore syntax

```
cio_ignore syntax

<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
```
cio_ignore

- 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 435) 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. Note that this command only manipulates the list of devices to be ignored, and does not have any immediate effect on already present devices.

- 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. For example, the devices still appear in the output of the lsccs command and can be set online. 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 435.

Examples:

Note: The dynamically changed list of devices to be ignored is only taken into account when a device in this list is newly made available to the system, for example after it has been defined to the system. It does not have any effect on setting devices online or offline within Linux.

- 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
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 434 and "possible_cpus" on page 441.

Format

maxcpus syntax

```plaintext
maxcpus=<number>
```

Examples

maxcpus=2
mem

Usage

Restricts memory usage to the size specified. You can use the K, M, or G suffix to specify the value in kilobyte, megabyte, or gigabyte.

Format

```
mem syntax

mem=<size>  K
            M
            G
```

Examples

```
mem=64M

Restricts the memory Linux can use to 64 MB.
mem=123456K

Restricts the memory Linux can use to 123456 KB.
```
**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 434 and "maxcpus" on page 438.

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:

```bash
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.
vmpanic

Usage

Specifies a command to be issued to CP after a kernel panic. This command is only applicable if the system runs as a VM guest.

Format

```
 vmpanic syntax

 vmpanic=<COMMAND>
```

Examples

This example specifies that a VMDUMP should follow a kernel panic:

```
vmpanic="VMDUMP"
```

Note: The command must be entered in uppercase.
vmpoff

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.
vmreboot

Usage

Specifies a command to be issued to CP on reboot. This command is only applicable if the system runs as a VM guest.

Format

```
vmreboot syntax

vmreboot=<COMMAND>
```

Examples

This example specifies that a message to guest MASTER should be sent in case of a reboot:

```
vmreboot="MSG MASTER Reboot system"
```

Note: The command must be entered in uppercase.
Chapter 42. Linux diagnose code use

SUSE Linux Enterprise Server 11 for System z issues several diagnose instructions to the hypervisor (LPAR or z/VM). Table 54 lists all diagnoses which are used by the Linux kernel or a kernel module.

Linux can fail if you change the privilege class of the diagnoses marked as **required** using the MODIFY diag command in z/VM.

Table 54. Linux diagnoses

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Linux use</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x008</td>
<td>VM/CP command console interface</td>
<td>• The vmcp command • The 3215 and 3270 console drivers • The z/VM recording device driver (vmlogdr) • smsgiucv</td>
<td>Required</td>
</tr>
<tr>
<td>0x010</td>
<td>Release pages</td>
<td>CMM</td>
<td>Required</td>
</tr>
<tr>
<td>0x014</td>
<td>Input spool file manipulation</td>
<td>The vmur device driver</td>
<td>Required</td>
</tr>
<tr>
<td>0x044</td>
<td>Voluntary time-slice end</td>
<td>In the kernel for spinlock and udelay</td>
<td>Required</td>
</tr>
<tr>
<td>0x064</td>
<td>Allows Linux to attach a DCSS</td>
<td>The DCSS block device driver (dcssblk), xip, and the MONITOR record device driver (monreader).</td>
<td>Required</td>
</tr>
<tr>
<td>0x09c</td>
<td>Voluntary time slice yield</td>
<td>Spinlock.</td>
<td>Optional</td>
</tr>
<tr>
<td>0x0dc</td>
<td>Monitor stream</td>
<td>The APPLDATA monitor record and the MONITOR stream application support (monwriter).</td>
<td>Required</td>
</tr>
<tr>
<td>0x204</td>
<td>LPAR Hypervisor data</td>
<td>The hypervisor file system (hypfs).</td>
<td>Required</td>
</tr>
<tr>
<td>0x210</td>
<td>Retrieve device information</td>
<td>• The common I/O layer • The DASD driver DIAG access method • The vmur device driver</td>
<td>Required</td>
</tr>
<tr>
<td>0x224</td>
<td>CPU type name table</td>
<td>The hypervisor file system (hypfs).</td>
<td>Required</td>
</tr>
<tr>
<td>0x250</td>
<td>Block I/O</td>
<td>The DASD driver DIAG access method.</td>
<td>Required</td>
</tr>
<tr>
<td>0x258</td>
<td>Page-reference services</td>
<td>In the kernel, for pfault.</td>
<td>Optional</td>
</tr>
<tr>
<td>0x288</td>
<td>Virtual machine time bomb</td>
<td>The watchdog device driver.</td>
<td>Required</td>
</tr>
<tr>
<td>0x2fc</td>
<td>Hypervisor cpu and memory accounting data</td>
<td>The hypervisor file system (hypfs).</td>
<td>Required</td>
</tr>
<tr>
<td>0x308</td>
<td>Re-ipl</td>
<td>Re-ipl and dump code.</td>
<td>Required</td>
</tr>
</tbody>
</table>

Required means that a function is not available without the diagnose; optional means that the function is available but there might be a performance impact.
Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the
names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

**Trademarks**

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.
Glossary

This glossary includes IBM product terminology as well as selected other terms and definitions. Additional information can be obtained in:

- 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).
- Internet Request for Comments: 1208, Glossary of Networking Terms
- Internet Request for Comments: 1392, Internet Users’ Glossary

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

capauto-detection. Listing the addresses of devices attached to a card by issuing a query command to the card.

C

CCL. The Communication Controller for Linux on System z (CCL) replaces the 3745/6 Communication Controller so that the Network Control Program (NCP) software can continue to provide business critical functions like SNI, XRF, BNN, INN, and SSCP takeover. This allows you to leverage your existing NCP functions on a "virtualized" communication controller within the Linux on System z environment.

cdl. compatible disk layout. A disk structure for Linux on System z which allows access from other System z operating systems. This replaces the older ldl.

CEC. (Central Electronics Complex). A synonym for CPC.

channel subsystem. The programmable input/output processors of the System z, which operate in parallel with the cpu.

checksum. An error detection method using a check byte appended to message data

CHPID. channel path identifier. In a channel subsystem, a value assigned to each installed channel path of the system that uniquely identifies that path to the system.

Console. (1) In Linux, an output device for kernel messages. (2) In the context of IBM mainframes, a device that gives a system programmer control over the hardware resources, for example the LPARs.
CPC.  (Central Processor Complex). A physical collection of hardware that includes main storage, one or more central processors, timers, and channels. Also referred to as a CEC.

CRC.  cyclic redundancy check. A system of error checking performed at both the sending and receiving station after a block-check character has been accumulated.

CSMA/CD.  carrier sense multiple access with collision detection

CTC.  channel to channel. A method of connecting two computing devices.

CUU.  control unit and unit address. A form of addressing for System z devices using device numbers.

D

DASD.  direct access storage device. A mass storage medium on which a computer stores data.

device driver.  (1) A file that contains the code needed to use an attached device. (2) A program that enables a computer to communicate with a specific peripheral device; for example, a printer, a videodisc player, or a CD-ROM drive. (3) A collection of subroutines that control the interface between I/O device adapters and the processor.

E

ECKD.  extended count-key-data device. A disk storage device that has a data transfer rate faster than some processors can utilize and that is connected to the processor through use of a speed matching buffer. A specialized channel program is needed to communicate with such a device.

ESCON.  enterprise systems connection. A set of IBM products and services that provide a dynamically connected environment within an enterprise.

Ethernet.  A 10-Mbps baseband local area network that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and delayed retransmission. Ethernet uses CSMA/CD.

F

Fast Ethernet (FENET).  Ethernet network with a bandwidth of 100 Mbps

FBA.  fixed block architecture. A type of DASD emulated by VM.


FTP.  file transfer protocol. In the Internet suite of protocols, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

G

Gigabit Ethernet (GbE).  An Ethernet network with a bandwidth of 1000-Mbps

H

hardware console.  A service-call logical processor that is the communication feature between the main processor and the service processor.

Host Bus Adapter (HBA).  An I/O controller that connects an external bus, such as a Fibre Channel, to the internal bus (channel subsystem).

HMC.  hardware management console. A console used to monitor and control hardware such as the System z microprocessors.

HFS.  hierarchical file system. A system of arranging files into a tree structure of directories.

I

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.

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.


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 System z.

LVS (Linux virtual server). Network sprayer software used to dispatch, for example, http requests to a set of Web servers to balance system load.

M

MAC. medium access control. In a LAN this is the sub-layer of the data link control layer that supports medium-dependent functions and uses the services of the physical layer to provide services to the logical link control (LLC) sub-layer. The MAC sub-layer includes the method of determining when a device has access to the transmission medium.

Mbps. million bits per second.

MIB (Management Information Base). (1) A collection of objects that can be accessed by means of a network management protocol. (2) A definition for management information that specifies the information available from a host or gateway and the operations allowed.

MTU. maximum transmission unit. The largest block which may be transmitted as a single unit.

Multicast. A protocol for the simultaneous distribution of data to a number of recipients, for example live video transmissions.

N

NIC. network interface card. The physical interface between the IBM mainframe and the network.

O

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-Express. Abbreviation for Open Systems Adapter-Express networking features. These include 10 Gigabit Ethernet, Gigabit Ethernet, and Fast Ethernet.

OSPF. open shortest path first. A function used in route optimization in networks.

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.
Glossary

S

SE. support element. (1) An internal control element of a processor that assists in many of the processor operational functions. (2) A hardware unit that provides communications, monitoring, and diagnostic functions to a central processor complex.

SNA. systems network architecture. The IBM architecture that defines the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. The layered structure of SNA allows the ultimate origins and destinations of information (the users) to be independent of and unaffected by the specific SNA network services and facilities that are used for information exchange.

SNMP (Simple Network Management Protocol). In the Internet suite of protocols, a network management protocol that is used to monitor routers and attached networks. SNMP is an application layer protocol. Information on devices managed is defined and stored in the application’s Management Information Base (MIB).

Sysctl. system control programming manual control (frame). A means of dynamically changing certain Linux kernel parameters during operation.

T

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.

Terminal. A physical or emulated device, associated with a keyboard and display device, capable of sending and receiving information.

U

UNIX. An operating system developed by Bell Laboratories that features multiprogramming in a multuser 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 or 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.
Index

Special characters
/sys, mount point viii
*ACCOUNT, VM record 189
*LOGREC, VM record 189
*SYMPTOM, VM record 189

Numerics
10 Gigabit Ethernet 83
SNMP 129
1000Base-T Ethernet
LAN channel station 139
SNMP 129
1000Base-T, Ethernet 83
1750, control unit 19
2105, control unit 19
2107, control unit 19
3088, control unit 139, 143, 161
31-bit
   z90crypt 246
3270 emulation 267
3370, DASD 19
3380, DASD 19
3390, DASD 19
3480 tape drive 65
3490 tape drive 65
3590 tape drive 65
3592 tape drive 65
3880, control unit 19
3990, control unit 19
6310, control unit 19
9336, DASD 19
9343, control unit 19
9345, DASD 19

A
access control
   FCP LUN 45
access_denied
   zfcp attribute (port) 54
   zfcp attribute (SCSI device) 57, 58
access_shared
   zfcp attribute 58
ACCOUNT, VM record 189
actions, shutdown 315
activating standby CPU 225
adapter_name, CLAW attribute 163
add, DCSS attribute 202
adding and removing cryptographic adapters 250
additional_cpus, kernel parameter 434
Address Resolution Protocol
   See ARP
AF_IUCV protocol support 219
AgentX protocol 129
alias
   DASD attribute 38
   alias device 38
API
   devices 7
   API
   FC-HBA 44
api_type
   CLAW attribute 163
APPDATA, monitor stream 173
ARP 90
   proxy ARP 111
   query/purge OSA-Express ARP cache 398
attributes
   device 9
   for CCW devices 9
   for subchannels 12
   qeth 92, 93
auto-detection
   DASD 29
   LCS 139
autoconfiguration, IPv6 88
autorecording, z/VM recording attribute 191
availability
   common CCW attribute 9
   DASD attribute 33
avg_*, cmf attributes 321
   avg_control_unit_queuing_time, cmf attribute 321
   avg_device_active_only_time, cmf attribute 321
   avg_device_busy_time 321
   avg_device_busy_time, cmf attribute 321
   avg_device_connect_time, cmf attribute 321
   avg_device_disconnect_time, cmf attribute 321
   avg_function_pending_time, cmf attribute 321
   avg_initial_command_response_time, cmf attribute 321
   avg_sample_interval, cmf attribute 321
   avg_utilization, cmf attribute 321

B
base device 38
base name
   network interfaces 4
block device
   tape 65
   block_size, tape attribute 71
bootsize, tape attribute 71
boot devices 298
   preparing 275
boot loader code 299
booting Linux 297
buffer_count, qeth attribute 98
buffer, CTCM attribute 147
buffer, IUCV attribute 156
bus ID 9
control unit 1750 19
2105 19
2107 19
3880 19
3990 19
6310 19
9343 19
cooperative memory management 221
CP Assist for Cryptographic Function 244, 253
CP commands
send to VM hypervisor 424
CP Error Logging System Service 189
CPACF 244
CPI
set attribute 324
sysplex_name attribute 324
system_level attribute 324
system_name attribute 323
system_type attribute 324
CPI (control program identification) 323
CPU capability change 225
CPU configuration 344
CPU, activating standby 225
CPU, deactivating operating 226
cpuplugd, Linux command 344
CRT 243
Crypto Express2 243
cryptographic adapter
attributes 250
cryptographic adapters
adding and removing dynamically 250
cryptographic device driver
See zcrypt
cTC
activating an interface 148
cTC interface
recovery 149
cTCM
buffer attribute 147
device driver 143
group attribute 145
online attribute 146
protocol attribute 146
subchannels 143
type attribute 146
cutype
common CCW attribute 9
tape attribute 71

D
DASD
access by bus-ID 27
access by VOLSER 28
alias attribute 38
availability attribute 33
booting from 301, 305
boxed 33
control unit attached devices 19
device driver 19

DASD (continued)
device names 25
discipline attribute 38
displaying information 350
displaying overview 371
eer_enabled attribute 36
erplog attribute 37
extended error reporting 19
failfast attribute 37
features 19
forcing online 33
formatting ECKD 347
module parameter 28
online attribute 36
partitioning 359
partitions on 20
performance tuning 421
readonly attribute 38
uid attribute 38
use_diag attribute 34, 39
vendor attribute 39
virtual 19
dasd=
module parameter 28
dasdfmt, Linux command 347
dasdview, Linux command 350
dbfs size=, module parameters 47
dCSS
access mode 204
add attribute 202
device driver 199
device names 199
device nodes 199
loader 286
minor number 203
remove attribute 205
save attribute 204
shared attribute 204
dcssblk.segments=, module parameter 200
deactivating operating CPU 226
decryption 243
delete
zfcp attribute (SCSI device) 58
delete, zfcp attribute 61
depth
cryptographic adapter attribute 250
device
cnsole 261
device bus-ID 9
of a qeth interface 101
device driver
CLAW 161
crypto 243
CTCM 143
DASD 19
DCSS 199
HiperSockets 81
in sysfs 10
LCS 139
Index 459
device driver (continued)

- monitor stream application 179
- NETIUCV 153
- network 80
- OSA-Express (QDIO) 81
- overview 8
- pseudo-random number 253
- qeth 81
- SCSI-over-Fibre Channel 41
- tape 65
- vmcp 217
- vmur 197
- watchdog 213
- XPRAM 75
- z/VM *MONITOR record reader 183
- z/VM recording 189
- z90crypt 243
- zfcp 41

device names 3
- DASD 25
- DCSS 199
- random number 253
- tape 66
- vmur 197
- XPRAM 75
- z/VM *MONITOR record 185
- z/VM recording 189

device nodes 3
- console 262
- DCSS 199
- random number 253
- SCSI 43
- tape 68
- vmcp 217
- vmur 197
- watchdog 215
- z/VM *MONITOR record 185
- z/VM recording 189
- z90crypt 248
- zfcp 43

device numbers 3

device special file
- See device nodes

device_blocked
- zfcp attribute (SCSI device) 58

devices
- alias 38
- attributes 9
- base 38
- corresponding interfaces 5
- ignoring 435
- in sysfs 9

devs=, module parameter 76

devtype
- common CCW attribute 9
- tape attribute 71

DHCP 126
- required options 126

Direct Access Storage Device
- See DASD

Direct SNMP 129

disabled wait
- cio_ignore 328

discipline
- DASD attribute 38
- discontiguous saved segments
- See DCSS

- dmesg 5
- domain=
  - module parameter 246

drivers
- See device driver
dump device
- DASD and tape 281
- ECKD DASD 282
- SCSI 284
dumped_frames, zfcp attribute 50

DVD, loading Linux 308

Dynamic Host Configuration Protocol
- See DHCP
dynamic routing, and VIPA 112

E

EBCDIC
- kernel parameters 299

ECKD 19
- devices 19
- EDP, value for geth large_send attribute 96
- edit characters, z/VM console 274
- EEDK 415
eer_enabled
- DASD attribute 36

EKM 415
- enable, qeth IP takeover attribute 109
- encryption 243
- encryption key manager 415
- Enterprise Storage Server 19

environment variables
- LOADDEV 302
- TERM 266
- ZIPLCONF 291

erplog, DASD attribute 37

Error Logging System Service 189
ero_frames, zfcp attribute 50

ESS 19

Ethernet 83
- interface name 87, 139
- LAN channel station 139
-
execution protection feature 255
- expanded memory 75
- extr 199

extended error reporting, DASD 19

external encrypted data key 415

F

failed
- zfcp attribute (channel) 52
- zfcp attribute (port) 54, 55
- zfcp attribute (SCSI device) 58, 60

failed, zfcp attribute 49
failfast, DASD attribute 37
fake_broadcast, qeth attribute 108
Fast Ethernet
   LAN channel station 139
FBA devices 19
FC-HBA 44
FCP 41
discovery 48
traces 48
FCP LUN access control 45
fcp_input_megabytes zfcp attribute 50
fcp_input_requests zfcp attribute 50
fcp_lun
   zfcp attribute (SCSI device) 58
fcp_lun zfcp attribute 57
fcp_output_megabytes zfcp attribute 50
fcp_output_requests zfcp attribute 50
fdasd, Linux command 359
feature
   execution protection 255
Fibre Channel 41
file system
   hugetlbfs 233
file systems
   ext2 199
   ISO9660 66
   sysfs 7
tape 66
   xip option 199
FTP server, loading Linux 308
full-screen mode terminal 266

G
generating random numbers 249
Gigabit Ethernet 83
   SNMP 129
group
   CLAW attribute 162
   CTCM attribute 145
   LCS attribute 140
   qeth attribute 94
group devices
   CLAW 161
   CTCM 143
   LCS 139
   qeth 86
guest LAN sniffer 127

H
Hardware Management Console
   See HMC
hardware status, z90crypt 248
hardware_version, zfcp attribute 49
HBA API 44
hba_id
   zfcp attribute (SCSI device) 58
hba_id zfcp attribute 57
high availability project 413
high resolution polling timer 342
HiperSockets
device driver 81
   interface name 87
HiperSockets Network Concentrator 120
HMC 259
   as terminal 268
   for booting Linux 298
host_name, CLAW attribute 163
hotplug
   CCW devices 15
   memory 229
hugepages=, kernel parameters 233
hugetlbfs
   virtual file system 233
hw_checksumming, value for qeth checksumming
   attribute 107
hwtype
   cryptographic adapter attribute 250
IBM compatible disk layout 21
IBM label partitioning scheme 20
IBM TotalStorage Enterprise Storage Server 19
IDRC compression 72
if_name, qeth attribute 100
ifconfig 4
Improved Data Recording Capability compression 72
in_recovery
   zfcp attribute (channel) 52
   zfcp attribute (port) 54, 55
   zfcp attribute (SCSI device) 58, 60
in_recovery zfcp attribute 49
Initial Program Load
   See IPL
initial RAM disk 300
Inter-User Communication Vehicle
   See IUCV
interface
   MTIO 68
   network 4
interface names
   claw 161
cct 144
IUCV 155
lcs 139
mpc 144
overview 4
qeth 87, 100
versus devices 5
vmur 197
interfaces
   FC-HBA 44
invalid_crc_count zfcp attribute 50
invalid_tx_word_count zfcp attribute 50
iocounterbits
   zfcp attribute 58
iodone_cnt
   zfcp attribute (SCSI device) 58
ioerr_cnt
zfcp attribute (SCSI device) 58
iorequest_cnt
zfcp attribute (SCSI device) 58
IP address
confirming 102
duplicate 103
takeover 108
virtual 112
IP, service types 98
ipa_takeover, qeth attributes 108
IPL 297
displaying current settings 376
NSS 210
IPL devices
for booting 298
preparing 275
IPv6
stateless autoconfiguration 88
support for 87
ISO9660 file systems 66
IUCV
activating an interface 156
buffer attribute 156
connection attribute 155
devices 154
MTU 156
remove attribute 158
user attribute 155
VM enablement 154, 219

J
journaling file systems
write barrier 33

K
KEK 415
kernel image 299
kernel messages 329
kernel module
vmur 197
kernel panic 312
kernel parameters 299
  additional_cpus= 434
  and zipl 280
  channel measurement facility 319
cio_ignore= 435
cmf.format= 319
cmf.maxchannels= 319
conmode= 265
console= 265
general 433
hugepages= 233
kernelcore= 230
maxcpus= 438
mem= 439
noinitrd 440
possible_cpus= 441
ramdisk_size= 442

kernel parameters (continued)
root= 444
savesys= 209
vmhalt= 445
vmpanic= 446
vmpoff= 447
vmreboot= 448
kernel sharing 209
kernel source tree vii
kernelcore=, kernel parameters 230
key encrypting key 415

L
LAN
z/VM guest LAN sniffer 127
LAN channel station
See LCS
LAN, virtual 117
lancmd_timeout, LCS attribute 141
large page support 233
large_send, qeth attribute 96
layer2, qeth attribute 88
lcs
  recover attribute 142
LCS
activating an interface 142
device driver 139
group attribute 140
lancmd_timeout attribute 141
online attribute 141
subchannels 139
libica library 247
lic_version, zfcp attribute 49
line edit characters, z/VM console 274
line-mode terminal 266
control characters 270
special characters 270
link_failure_count, zfcp attribute 50
link_failure_count, zfcp attribute 50
Linux device special file
See device nodes
Linux disk layout 23
Linux guest
  reducing memory of 171
Linux guest, booting 300
Linux in LPAR mode, booting 305
lip_count, zfcp attribute 50
LNX1 labeled disk 23
LOADDEV, environment variable 302
login at terminals 267
LOGREC, VM record 189
long random numbers 249
loss_of_signal_count, zfcp attribute 50
loss_of_sync_count, zfcp attribute 50
LPAR Linux, booting 305
lschp, Linux command 367
lscss, Linux command 369
lsdasd, Linux command 371
lsqeth
  command 101
lsqeth, Linux command 374
lsreipl, Linux command 376
lsshut, Linux command 377
lstape, Linux command 378
lszcrypt, Linux command 381
lszfcp, Linux command 383

M
MAC addresses 88
MAC header
   layer2 for qeth 88
major number 3
   DASD devices 25
   pseudo-random number 253
   tape devices 66
   XPRAM 75
man pages, messages 329
management information base 129
maxcpus=, kernel parameter 438
maxframe_size
   zfcp attribute 49
Media Access Control (MAC) addresses 88
Medium Access Control (MAC) header 89
medium_state, tape attribute 71
mem=, kernel parameter 439
memory
   block_size_bytes attribute 230
   guest, reducing 171
   hotplug 229
memory, expanded 75
memory, state attribute 231
menu configuration 292
   VM example 302
messages 329
MIB (management information base) 129
minor number 3
   DASD devices 25
   DCSS devices 203
   pseudo-random number 253
   tape devices 66
   XPRAM 75
modalias
   cryptographic adapter attribute 251
model
   zfcp attribute (SCSI device) 58
module
   See kernel module
module parameters
   cmd= 214
   conceal= 214
   CPI 323
dasd= 28
dbfsizes= 47
dcssblk.segments= 200
devs= 76
domain= 246
mondcss= 179, 185
nowayout= 214
poll_thread= 246
sizes= 76
system_name= 323
module parameters (continued)
   XPRAM 76
   z90crypt 246
modulus-exponent 243
mon_fsstatd, command 385
mon_procd, command 390
mondcss=, module parameters 179, 185
monitor stream 173
   module activation 174
   on/off 174
   sampling interval 175
monitor stream application
   device driver 179
mount point, sysfs viii
MTIO interface 68
MTU
   IUCV 156
   qeth 101
   multicast-router, value for qeth router attribute 105
   multiple subchannel set 10

N
name
devices
   See device names
network interface
   See base name
named saved system 209
   See NSS
net-snmp 129
NETIUCV
device driver 153
network
   device drivers 80
   interface names 4
Network Concentrator 120
network interfaces 4
no_checksumming, value for qeth checksumming
   attribute 107
no_prio_queueing, value for qeth priority_queueing
   attribute 97
no_router, value for qeth router attribute 106
no, value for qeth large_send attribute 96
node_name
   zfcp attribute 49
   zfcp attribute (port) 54
node, device
   See device nodes
noinitrd, kernel parameter 440
non-priority commands 272
non-rewinding tape device 65
nos_count, zfcp attribute 50
notices 451
nowayout=, module parameters 214
NPIV
   example 51
   FCP channel mode 51
   for FCP channels 46
NSS 303
NSS (named saved system) 209
numbers, random 249

O

object ID 130
offline
  CHPID 14, 15
devices 9
OID (object ID) 130
online
  CHPID 14, 15
  CLAW attribute 164
  common CCW attribute 9
cryptographic adapter attribute 248
CTCM attribute 146
DASD attribute 36
LCS attribute 141
qeth attribute 100
tape attribute 69, 71
TTY attribute 270
zfcp attribute 48
online, zfcp attribute 49
Open Source Development Network, Inc. 129
openCryptoki 247
operating CPU, deactivating 226
operation, tape attribute 71
OSA-Express
  device driver 81
  LAN channel station 139
  SNMP subagent support 129
osasnmqd, OSA-Express SNMP subagent 129
OSDN (Open Source Development Network, Inc.) 129

P

P/390 439
page pool
  static 171
timed 171
parallel access volume (PAV) 38
PARM
  IPL parameter 210
partition
  on DASD 20
  schemes for DASD 20
  table 22
  XPRAM 75
PAV (parallel access volume) 38
PAV enablement, suppression 29
peer_d_id, zfcp attribute 49
peer_wwvn, zfcp attribute 49
peer_wwpn, zfcp attribute 49
permanent_port_name, zfcp attribute 49, 51
physical_s_id, zfcp attribute 51
pm_pampom, subchannel attribute 13
PKCS #11 API 243, 247
planned changes in channel path availability 327
poll thread
  disable using chcrypt 342
  enable using chcrypt 342
poll_thread
cryptographic adapter attribute 249
poll_thread=
  module parameter 246
poll_timeout
cryptographic adapter attribute 249
  set using chcrypt 342
port_id
  zfcp attribute (port) 54
  port_id, zfcp attribute 49
port_name
  zfcp attribute (port) 54
  port_name, zfcp attribute 49
port_remove, zfcp attribute 49, 53
port_rescan, zfcp attribute 49, 53
port_state
  zfcp attribute (port) 54
  port_type, zfcp attribute 49
portno, qeth attribute 99
possible_cpus=, kernel parameter 441
preferred console 265
prerequisites 1, 17, 79, 167, 223, 241, 257, 317, 331
prim_seq_protocol_err_count, zfcp attribute 50
primary_connector, value for qeth router attribute 105
primary_router, value for qeth router attribute 105
prio_queueing, value for qeth priority_queueing attribute 98
priority command 272
priority_queueing, qeth attribute 97
processors
  cryptographic 7
procs
  appldata 173
cio_ignore 436
  magic sysrequest function 271
VLAN 119
protocol, CTCM attribute 146
proxy ARP 111
proxy ARP attributes 93
pseudo-random number
  device driver 253
device names 253
device nodes 253
purge, z/VM recording attribute 192
PVMSG 272

Q

QDIO 86
qeth
  activating an interface 101
  auto-detection 86
  buffer_count attribute 98
card_type attribute 99
checksumming attribute 107
configuration tool 400
device driver 81
displaying device overview 374
enable attribute for IP takeover 109
fake_broadcast attribute 108
group attribute 94
qeth (continued)
  if_name attribute 100
  ipa_takeover attributes 108
  large_send attribute 96
  layer2 attribute 88
  MTU 101
  online attribute 100
  portno attribute 99
  priority_queueing attribute 97
  recover attribute 103
  route4 attribute 105
  route6 attribute 105
  subchannels 86
  summary of attributes 92, 93
  TCP segmentation offload 96
  VIPA attributes 94
  qeth interfaces, mapping 5
  qetharp, Linux command 398
  qethconf, Linux command 400
  queue_depth
  zfcp attribute (SCSI device) 58
  queue_type
  zfcp attribute (SCSI device) 58
  queueing, priority 97
  roles
  zfcp attribute (port) 54
  root=, kernel parameter 444
  route4, qeth attribute 105
  route6, qeth attribute 105
  router
    IPv4 router settings 105
    IPv6 router settings 105
  RSA 243
  RSA exponentiation 243
  RCS 429
  RVA 19
  rx_frames, zfcp attribute 50
  rx_words, zfcp attribute 50

S
  s_id, zfcp attribute 51
  S/390 hypervisor file system 235
    defining access rights 238
  sample_count, cmf attribute 321
  save, DCSS attribute 204
  savesys=, kernel parameters 209
  SCSI
    multipath devices 44
    SCSI devices, in sysfs 57
    SCSI disk, booting from 302
    SCSI system dumper 284
  scsi_host_no, zfcp attribute 56
  scsi_id, zfcp attribute 56
  scsi_level
    zfcp attribute (SCSI device) 58
  scsi_logging_level, Linux command 403
  scsi_lun, zfcp attribute 56
  scsi_target_id
    zfcp attribute (port) 54
  SCSI-over-Fibre Channel
    See zfcp
  SCSI-over-Fibre Channel device driver 41
  SCSI, booting from 305
  SE (Support Element) 298
    secondary_connector, value for qeth router attribute 105
    secondary_router, value for qeth router attribute 105
    seconds_since_last_reset
      zfcp attribute 50
    segmentation offload, TCP 96
  serial_number, zfcp attribute 49
  service types, IP 98
  set, CPI attribute 324
  setsockopt 98
  shared kernel 209
  shared, DCSS attribute 204
  Shoot The Other Node In The Head 413
  shutdown actions 315
  simple network IPL 406
  Simple Network Management Protocol 129
  sizes=, module parameter 76
  sniffer, guest LAN 127
  snipl, Linux command 406
  SNMP 129, 413
special characters
  line-mode terminals 270
  z/VM console 274
special file
  See device nodes
speed, zfcp attribute 49
ssch_rsch_count, cmf attribute 321
standby CPU, activating 225
state
  memory attribute 231
  zfcp attribute (SCSI device) 58
state, tape attribute 71
stateless autoconfiguration, IPv6 88
static page pool 171
static routing, and VIPA 112
status, CHPID attribute 14, 15
STONITH 413
storage
  memory hotplug 229
subchannel
  multiple set 10
subchannel set ID 10
subchannels
  CCW and CCW group devices 7
  CLAW 161
  CTCM 143
  displaying overview 369
  in sysfs 11
  LCS 139
  qeth 86
support
  AF_IUCV protocol 219
Support Element 298
supported_classes
  zfcp attribute (port) 54
supported_classes, zfcp attribute 49
supported_speeds, zfcp attribute 50
sw_checksumming, value for qeth checksumming attribute 107
swapping
  avoiding 171
SYMPOTOM, VM record 189
syntax diagrams ix
sysfs 7
sysplex_name, CPI attribute 324
system states
  displaying current settings 377
System z Application Programming Interfaces 414
system_level, CPI attribute 324
system_name, CPI attribute 323
system_name=, module parameter 323
system_type, CPI attribute 324
Systems Management Application Programming, z/VM 414

Tape (continued)
  character device 65
cmb_enable attribute 71
cutype attribute 71
device names 66
device nodes 68
devtype attribute 71
display support 419
displaying overview 378
encryption support 415
file systems 66
IDRC compression 72
loading and unloading 72
medium_state attribute 71
MTIO interface 68
online attribute 69, 71
operation attribute 71
state attribute 71
tape device driver 65
tape390_crypt, Linux command 415
tape390_display, Linux command 419
TCP segmentation offload 96
TCP/IP
  ARP 90
  checksumming 107
  DHCP 126
  IUCV 153
  point-to-point 143
  service machine 144, 158
TERM, environment variable 266
terminal
  enabling user logins 267
  mainframe versus Linux 260
tgid_bind_type, zfcp attribute 50
timed page pool 171
timeout
  zfcp attribute (SCSI device) 58
timeout for LCS LAN commands 141
trademarks 452
TSO, value for qeth large_send attribute 96
TTY
  console devices 261
  online attribute 270
  routines 262
tunedasd, Linux command 421
tx_frames, zfcp attribute 50
tx_words, zfcp attribute 50
type
  cryptographic adapter attribute 251
  zfcp attribute (SCSI device) 58
type, CTCM attribute 146

U
ucd-snmp 129
uid
  DASD attribute 38
  unit_add, zfcp attribute 55
  unit_remove, zfcp attribute 61
unplanned changes in channel path availability 327
zfcp (continued)
  hardware_version attribute  49
  hba_id attribute  57
  hba_id attribute (SCSI device)  58
  in_recovery attribute  49
  in_recovery attribute (channel)  52
  in_recovery attribute (port)  54, 55
  in_recovery attribute (SCSI device)  58, 60
  invalid_crc_count attribute  50
  invalid_tx_word_count attribute  50
  iocounters attribute  58
  iodone_cnt attribute (SCSI device)  58
  ioerr_cnt attribute (SCSI device)  58
  iorequest_cnt attribute (SCSI device)  58
  lic_version attribute  49
  link_failure_count attribute  50
  lip_count attribute  50
  loss_of_signal_count attribute  50
  loss_of_sync_count attribute  50
  maxframe_size attribute  49
  model attribute (SCSI device)  58
  node_name attribute  49
  node_name attribute (port)  54
  nos_count attribute  50
  online attribute  48, 49
  peer_d_id attribute  49
  peer_wwnn attribute  49
  peer_wwpn attribute  49
  permanent_port_name attribute  49, 51
  physical_s_id attribute  51
  port_id attribute  49
  port_id attribute (port)  54
  port_name attribute  49
  port_name attribute (port)  54
  port_remove attribute  49, 53
  port_rescan attribute  49, 53
  port_state attribute (port)  54
  port_type attribute  49
  prim_seq_protocol_err_count attribute  50
  queue_depth attribute (SCSI device)  58
  queue_type attribute (SCSI device)  58
  rescan attribute (SCSI device)  58
  reset_statistics attribute  50
  rev attribute (SCSI device)  58
  roles attribute (port)  54
  rx_frames attribute  50
  rx_words attribute  50
  s_id attribute  51
  scsi_host_no attribute  56
  scsi_id attribute  56
  scsi_level attribute (SCSI device)  58
  scsi_lun attribute  56
  scsi_target_id attribute (port)  54
  seconds_since_last_reset attribute  50
  serial_number attribute  49
  speed attribute  49
  state attribute (SCSI device)  58
  supported_classes attribute  49
  supported_classes attribute (port)  54
  supported_speeds attribute  50
  tgid_bind_type attribute  50

zfcp (continued)
  timeout attribute (SCSI device)  58
  tx_frames attribute  50
  tx_words attribute  50
  type attribute (SCSI device)  58
  unit_add attribute  55
  unit_remove attribute  61
  vendor attribute (SCSI device)  58
  wwpn attribute  51, 57
  wwpn attribute (SCSI device)  58

zfcp HBA API  44
zfcp traces  48
zipl
  and kernel parameters  280
  base functions  275
  configuration file  291
  Linux command  275
  menu configurations  292
  parameters  288

ZIPLCONF, environment variable  291
Readers’ Comments — We’d Like to Hear from You

Linux on System z
Device Drivers, Features, and Commands
as available with
SUSE Linux Enterprise Server 11

Publication No. SC34-2595-00

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.
Submit your comments using one of these channels:
• Send your comments to the address on the reverse side of this form.
• Send your comments via e-mail to: eservdoc@de.ibm.com

If you would like a response from IBM, please fill in the following information:

Name
Address
Company or Organization
Phone No.
E-mail address