Linux on System z

IBM

Device Drivers, Features, and Commands

Development stream (Kernel 3.16)

SC33-8411-25

Linux on System z

Device Drivers, Features, and Commands

Development stream (Kernel 3.16)

SC33-8411-25

Note

Before using this document, be sure to read the information in "Notices" on page 747.

This edition applies to the Linux on System z Development stream for kernel 3.16, s390-tools version 1.26, and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Summary of changes

This revision reflects changes to the Development stream for kernel 3.16.

Updates for kernel 3.16

This edition contains changes related to the 3.16 kernel release.

New information

- Priority queueing for QDIO devices now supports IPv6. There are also two new values that you can set, prio_queueing_vlan for VLANs and prio_queueing_skb for other cases. See "Using priority queueing" on page 274.
- The **fdasd** command supports a new partition type that can be used in Elastic Storage file system setups. See "fdasd - Partition a DASD" on page 594.

Changed Information

- Parts have been reordered, see "How this document is organized" on page ix.
- The chapter about PCIe was integrated into Chapter 2, see "PCI Express support" on page 21
- The z/VM® watchdog device driver has been replaced by the diag288 watchdog device driver. The new device driver can be used for both Linux on z/VM and Linux in LPAR mode. See Chapter 30, "The diag288 watchdog device driver," on page 405.

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

Deleted Information

- None.

Updates for kernel 3.14

This edition contains changes related to the 3.14 kernel release.

New information

- A new DASD timeout setting limits the overall time that can expire before an I/O request is cancelled, see "Setting the timeout for I/O requests" on page 170.
- A HiperSockets™ port can be configured as a member of a Linux software bridge, see "HiperSockets layer 2 bridge port functionality" on page 264.
- The cryptographic device driver now supports coprocessors for the Enterprise PKCS#11 feature (EP11), see Chapter 42, "Generic cryptographic device driver," on page 481.
- The console chapter now includes sections about enabling user logins with systemd and about preventing getty respawns on unavailable terminals with systemd, see "Enabling a terminal for user logins using systemd" on page 49 and "Using systemd" on page 52.
- The support for the z/Architecture® CPU-measurement facilities now includes the CPU-measurement sampling facility, see Chapter 46, "Using the CPU-measurement facilities," on page 513. A new command helps you to

display details about supported and authorized counters and sampling modes, see “lscpumf - Display information about the CPU-measurement facilities” on page 614.

Changed Information

- None.

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

Deleted Information

- None.

Updates for kernel 3.12

This edition contains changes related to the 3.12 kernel release.

New information

- Source VIPA now supports IPv6 addresses, see “Source VIPA” on page 302
- A new device driver provides support of RDMA over Converged Ethernet (RoCE), see “PCI Express support” on page 21.
- With a new command, sncap, you can remotely control the capacity records of a CPC, see “sncap - Manage CPU capacity” on page 667.
- With a new command, zdsfs, you can mount z/OS DASDs as Linux file systems, see “zdsfs - Mount a z/OS DASD” on page 694

Changed Information

- The default for the FCP hardware data routing feature has changed from disabled to enabled, see“zfcp device driver kernel parameters” on page 192
- The SCSI standalone dump tool and dump no longer require a file system, see“zipl base functions” on page 65.

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

Deleted Information

- In the zipl chapter, the section called "Preparing a dump device on a SCSI disk" has been deleted and the information merged with the DASD information, see “Preparing a DASD, SCSI, or tape dump device” on page 75.

About this document

This document describes the device drivers available for Linux kernel 3.16 to control IBM® System z® devices and attachments. It also describes commands and parameters for configuring Linux for System z.

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

The most recent System z mainframes on which you can run Linux kernel 3.16 are IBM zEnterprise® BC12 (zBC12), IBM zEnterprise EC12 (zEC12), IBM zEnterprise 114 (z114), and IBM zEnterprise 196 (z196).

Unless stated otherwise, the device drivers, features, and commands described in this document are available for the System z 64-bit and 31-bit architectures.

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

For more specific information about the device driver structure, see the documents in the kernel source tree at `linux/Documentation/s390`. On an installed Linux system the absolute path is typically: `/usr/src/linux/Documentation/s390`.

You can find the latest version of this and other publications in the Linux on System z library on the developerWorks® website at www.ibm.com/developerworks/linux/linux390/documentation_dev.html

How this document is organized

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

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

Part three contains chapters specific to individual storage device drivers.

Part four contains chapters specific to individual network device drivers.

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

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

Part seven contains chapters about device drivers and features that support security aspects of Linux on System z.

Part eight contains chapters about assessing the performance of Linux on System z.

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

Part ten contains chapters with reference information about commands, kernel parameters, kernel options, 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 a Linux on System z system.

The following general assumptions are made about your background knowledge:

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

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

Kernel builders: Some sections are of interest primarily to kernel builders who want to build their own Linux kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Distribution specific information

This document does not provide information that is specific to a particular Linux distribution. The device drivers, features, options, and commands it describes are either provided with the Linux kernel on www.kernel.org, on developerWorks, or are commonly available tools.

Your Linux distribution might provide additional utilities for working with System z devices that are not described in this publication. See the documentation that is provided with your distribution to find out what additional utilities you can use.

For versions of this and other documents that have been adapted to a particular distribution, see one of the following web pages:
www.ibm.com/developerworks/linux/linux390/documentation_suse.html
www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html

Conventions and assumptions used in this publication

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

Authority

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

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

Persistent configuration

This document describes how to change settings and options for Linux on System z in sysfs and procfs. In most cases, changes in sysfs or procfs are not persistent. If you need to make your changes persistent, use tools provided by your distribution or use commonly available tools.

For example, use **`sysctl`** and the `/etc/sysctl.conf` configuration file to make persistent changes for settings that are represented in procfs. See procps.sf.net and the `sysctl.conf` man page for more information.

Terminology

In this publication, 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 an NSS. See also "IPL and booting" on page 89.

sysfs and procfs

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

debugfs

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

To mount debugfs, you can use this command:

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

To mount debugfs persistently, add the following to `/etc/fstab`:

```
debugfs /sys/kernel/debug debugfs auto 0 0
```

Number prefixes

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

Hexadecimal numbers

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

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

Highlighting

This publication uses the following highlighting styles:

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

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

Other Linux on System z publications

You can find Linux on System z publications on developerWorks and on the IBM Knowledge Center.

These publications are available on developerWorks at

www.ibm.com/developerworks/linux/linux390/documentation_dev.html

- *Device Drivers, Features, and Commands*, SC33-8411
- *Using the Dump Tools*, SC33-8412
- *How to Improve Performance with PAV*, SC33-8414
- *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413
- *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596
- *Kernel Messages*, SC34-2599
- *libica Programmer's Reference*, SC34-2602
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294
- *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713
- *Linux on System z Troubleshooting*, SC34-2612
- *Linux Health Checker User's Guide*, SC34-2609

These publications are available on the IBM Knowledge Center at

ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_lib.html

- *libica Programmer's Reference*, SC34-2602
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294
- *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713
- *Linux Health Checker User's Guide*, SC34-2609
- *Linux on System z Troubleshooting*, SC34-2612
- *Kernel Messages*, SC34-2599

Finding IBM publications

You can locate the latest versions of the referenced IBM publications through the IBM Publications Center at:

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

Part 1. General concepts

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

Newest version

You can find the newest version of this publication at www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Chapter 1. How devices are accessed by Linux

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

Device nodes and major/minor numbers

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

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

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

Device drivers assign device names to their devices, according to a device driver-specific naming scheme (see, for example, "DASD naming scheme" on page 151). Each device name is associated with a minor number (see Figure 1).

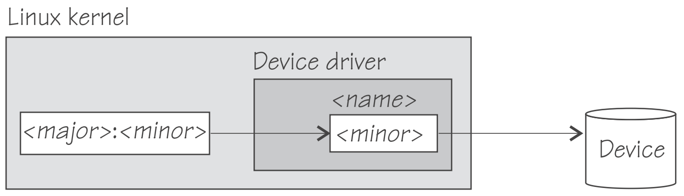


Figure 1. Minor numbers and device names

User space programs access character and block devices through *device nodes* also referred to as *device special files*. When a device node is created, it is associated with a major and minor number.

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

Creating device nodes

You can use the **mknod** command to create device nodes.

To create a device node, use a command of the form:

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

where:

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

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

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

<minor>
: is the minor number that maps to a device name used by the device driver.

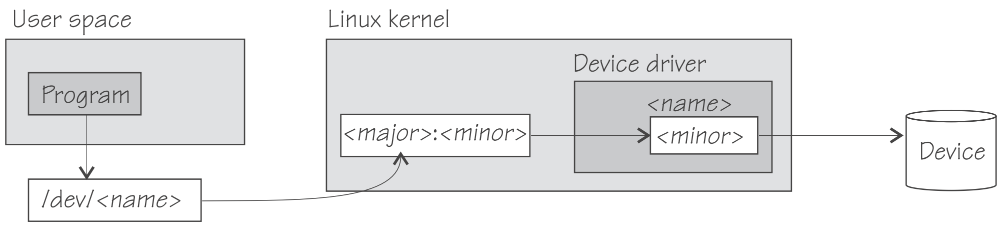


Figure 2. Device nodes

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

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

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

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

Device nodes provided by udev

udev is a utility program that can use the device information in sysfs to create device nodes.

If your distribution provides udev, you can use udev to create device nodes for you. Apart from creating device nodes that are based on the device names, udev can create device nodes that are based on characteristics of the physical devices, for example, on device bus-IDs or VOLSERs.

Unless you change these characteristics of your devices, the device nodes that are based on them remain the same and map to the same device, even if the device name of a device has changed (for example, after rebooting). udev keeps track of the mapping of the device name and the actual devices for you and so helps you ensure that you are addressing the device you intend to.

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

See "Examples for udev-created DASD device nodes" on page 153 and "Examples for udev-created tape device nodes" on page 232 for examples of what udev created device nodes might look like.

See the udev man page for more details.

Network interfaces

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

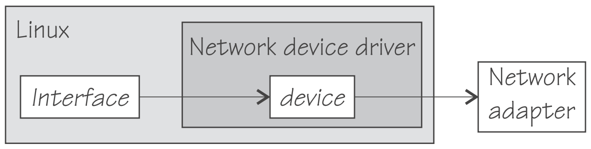


Figure 3. Interfaces

When a network device is defined, it is associated with a real or virtual network adapter (see Figure 3). You can configure the adapter properties for a particular network device through the device representation in sysfs (see "Device directories" on page 11).

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

Interface names

The interface names are assigned by the Linux network stack.

Interface names are of the form <*base_name*><*n*> where <*base_name*> is a base name that is used for a particular interface type. <*n*> is an index number that identifies an individual interface of a particular type.

Table 1 summarizes the base names that are used for the Linux on System z network device drivers for interfaces that are associated with real hardware.

Table 1. Interface base names for real devices.

This table lists interface type and applicable device driver for the available base names.

Base name	**Interface type**	**Device driver module**	**Hardware**
eth	Ethernet	qeth, lcs	OSA-Express features
eth	Ethernet	mlx4_en	RoCE Express feature
osn	ESCON/CDLC bridge	qeth	OSA-Express features
ctc	Channel-to-Channel	ctcm	ESCON channel card, FICON® channel card

Table 1. Interface base names for real devices (continued).

This table lists interface type and applicable device driver for the available base names.

Base name	Interface type	Device driver module	Hardware
mpc	Channel-to-Channel	ctcm	ESCON channel card
claw	CLAW	claw	ESCON channel card

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

Table 2. Interface base names for virtual devices.

This table lists interface type and applicable device driver for the available base names.

Base name	Interface type	Device driver module	Comment
hsi	HiperSockets, virtual NIC	qeth	Real HiperSockets or virtual NIC type HiperSockets coupled to a guest LAN
eth	virtual NIC	qeth	QDIO virtual NIC coupled to a guest LAN or virtual switch
ctc	virtual Channel-to-Channel	ctcm	virtual CTCA
mpc	virtual Channel-to-Channel	ctcm	virtual CTCA
iucv	IUCV	netiucv	IUCV authorizations are required

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

When the first device for a particular interface name is set online, it is assigned the index number 0, the second is assigned 1, the third 2, and so on. For example, the first HiperSockets interface is named hsi0, the second hsi1, the third hsi2, and so on. As an exception, IUCV devices do not need to be set online and the interface names are assigned when the device is created.

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

Matching devices with the corresponding interfaces

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

Your distribution might provide a way to track the mapping or to assign meaningful names to your interfaces. How you can keep track of the mapping yourself differs depending on the network device driver.

For qeth, you can use the **`lsqeth`** command (see "lsqeth - List qeth-based network devices" on page 627) to obtain a mapping.

After setting a device online (or creating an IUCV device), read `/var/log/messages` or issue **`dmesg`** to find the associated interface name in the messages that are issued in response to the device being set online (or created for IUCV).

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

Main steps for setting up a network interface

The main steps apply to all Linux on System z network devices drivers, except 10GbE RoCE Express features. How to perform a particular step can be different for the different device drivers.

Procedure

The main steps are:

1. Define a network device.

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

   Tip: Use the **`znetconf`** command to perform this step. See "znetconf - List and configure network devices" on page 699.
2. Configure the device through its attributes in sysfs. See "Device views in sysfs" on page 13

   Some devices have attributes that can or must be set later when the device is online or when the connection is active.
3. Set the device online (skip this step for IUCV network devices)

   This step 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.
4. Configure and activate the interface.

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

Chapter 2. Devices in sysfs

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

Device categories

There are several Linux on System z specific device categories in the `/sys/devices` directory.

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

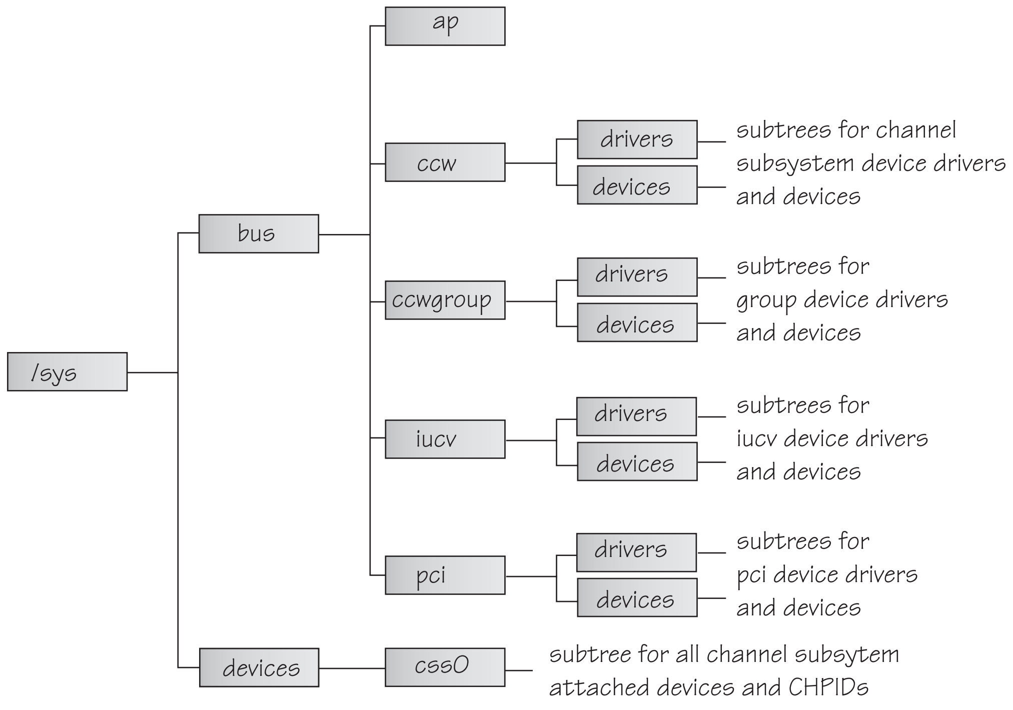

/sys
bus
devices
ap
ccw
ccwgroup
iucv
pci
css0
drivers
devices
subtrees for channel subsystem device drivers and devices
subtrees for group device drivers and devices
subtrees for iucv device drivers and devices
subtrees for pci device drivers and devices
subtree for all channel subsytem attached devices and CHPIDs

Figure 4. sysfs

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

AP devices
: are adjunct processors used for cryptographic operations.

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

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

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

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

Table 3 lists the Linux on System z device drivers that have representation in sysfs:

Table 3. Linux on System z device drivers with representation in sysfs

Device driver	Category	sysfs directories
3215 console	CCW	`/sys/bus/ccw/drivers/3215`
3270 console	CCW	`/sys/bus/ccw/drivers/3270`
DASD	CCW	`/sys/bus/ccw/drivers/dasd-eckd` `/sys/bus/ccw/drivers/dasd-fba`
SCSI-over-Fibre Channel	CCW	`/sys/bus/ccw/drivers/zfcp`
Storage class memory supporting Flash Express	SCM	`/sys/bus/scm/`
Channel-attached tape	CCW	`/sys/bus/ccw/drivers/tape_34xx` `/sys/bus/ccw/drivers/tape_3590`
Cryptographic	AP	`/sys/bus/ap/drivers/cex4a` `/sys/bus/ap/drivers/cex4c` `/sys/bus/ap/drivers/cex4p` `/sys/bus/ap/drivers/cex3a` `/sys/bus/ap/drivers/cex3c` `/sys/bus/ap/drivers/cex2a` `/sys/bus/ap/drivers/pcica` `/sys/bus/ap/drivers/pcicc` `/sys/bus/ap/drivers/pcixcc`
DCSS	n/a	`/sys/devices/dcssblk`
XPRAM	n/a	`/sys/devices/system/xpram`
z/VM recording	IUCV	`/sys/bus/iucv/drivers/vmlogrdr`
qeth (OSA-Express features and HiperSockets)	CCW group	`/sys/bus/ccwgroup/drivers/qeth`
LCS	CCW group	`/sys/bus/ccwgroup/drivers/lcs`
CTCM	CCW group	`/sys/bus/ccwgroup/drivers/ctcm`
NETIUCV	IUCV	`/sys/bus/iucv/drivers/netiucv`
CLAW	CCW group	`/sys/bus/ccwgroup/drivers/claw`
10GbE RoCE Express devices (mlx4_en)	PCI	`sys/bus/pci/drivers/mlx4_core`

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 IUCV device driver and the IUCV-dependent z/VM recording device driver have their own category, IUCV.

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

Device directories

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

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

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

“Device views in sysfs” on page 13 tells you where you can find the device directories with their attributes in sysfs.

Device attributes

The device directories contain attributes. You control a device by setting 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 504 for details.

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

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

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

  `boxed`
  : The device is locked by another operating system instance and cannot be used until the lock is surrendered or the DASD is accessed by force (see “Accessing DASD by force” on page 164).

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

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

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

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

or

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

Setting attributes

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

Procedure

- You can set a writable attribute by writing the designated value to the corresponding attribute file.
- For CCW devices, you can also use the **chccwdev** command (see "chccwdev - Set CCW device attributes" on page 543) to set attributes.

  With a single **chccwdev** command you can:
  - Set an attribute for multiple devices
  - Set multiple attributes for a device, including setting the device online
  - Set multiple attributes for multiple devices

Working with newly available devices

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

About this task

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

Procedure

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

1. Attach the device, for example, with a z/VM CP ATTACH command.
2. Assure that the sysfs structures for the new device are complete.

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

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

Tip: For CCW devices, you can omit this step if you then use **`chccwdev`** (see "chccwdev - Set CCW device attributes" on page 543) to work with the devices. **`chccwdev`** triggers cio_settle for you and waits for cio_settle to complete.

Results

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

Device views in sysfs

sysfs provides multiple views of device specific data.

The most important views are:
- "Device driver view"
- "Device category view" on page 14
- "Device view" on page 14
- "Channel subsystem view" on page 14

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

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

where *<n>* is the subchannel set-ID and *<devno>* is the device number. Multiple subchannel sets are available on System z9® or later machines.

Device driver view

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

The device driver view is of the form:

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

where:

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

<driver>
: is a name that specifies an individual device driver or the device driver component that controls the device (see Table 3 on page 10).

<device_bus_id>
: identifies an individual device (see "Device directories" on page 11).

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

Examples

- This example shows the path for an ECKD™ type DASD device:
  `/sys/bus/ccw/drivers/dasd-eckd/0.0.b100`
- This example shows the path for a qeth device:
  `/sys/bus/ccwgroup/drivers/qeth/0.0.a100`
- This example shows the path for a cryptographic device (a CEX2A card):
  `/sys/bus/ap/drivers/cex2a/card3b`

Device category view

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

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

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

where:

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

<device_bus_id>
identifies an individual device (see “Device directories” on page 11).

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

Examples

- This example shows the path for a CCW device.
  `/sys/bus/ccw/devices/0.0.b100`
- This example shows the path for a CCW group device.
  `/sys/bus/ccwgroup/devices/0.0.a100`
- This example shows the path for a cryptographic device:
  `/sys/bus/ap/devices/card3b`

Device view

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

The device view is of the form:
```
/sys/devices/<driver>/<device>
```

where:

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

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

Examples

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

Channel subsystem view

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

The channel subsystem view is of the form:

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

where:

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

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

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

where:

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

<device_bus_id>
: is a device number with a leading "0.<*n*>.", where <*n*> is the subchannel set ID (see "Device directories" on page 11).

Examples

- This example shows a CCW device with device number 0xb100 that is associated with a subchannel 0x0001.
  ```
  /sys/devices/css0/0.0.0001/0.0.b100
  ```
- This example shows a CCW device with device number 0xb200 that is associated with a subchannel 0x0001 in subchannel set 1.
  ```
  /sys/devices/css0/0.1.0001/0.1.b200
  ```
- The entries for a group device show as separate subchannels. If a CCW group device uses three subchannels 0x0002, 0x0003, and 0x0004 the subchannel information could be:
  ```
  /sys/devices/css0/0.0.0002/0.0.a100
  /sys/devices/css0/0.0.0003/0.0.a101
  /sys/devices/css0/0.0.0004/0.0.a102
  ```

  Each subchannel is associated with a device number. Only the primary device number is used for the bus ID of the device in the device driver view and the device view.
- This example lists the information available for a non-I/O subchannel with which no device is associated:
  ```
  ls /sys/devices/css0/0.0.ff00/
  bus  driver  modalias  subsystem  type  uevent
  ```

Subchannel attributes

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

Subchannels have two common attributes:

`type`
: The subchannel type, which is a numerical value, for example:
  - 0 for an I/O subchannel
  - 1 for a CHSC subchannel
  - 3 for an EADM subchannel

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

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

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

chpids
is a list of the channel-path identifiers (CHPIDs) through with the device is connected. See also "Channel path ID information" on page 17.

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

Channel path measurement

A sysfs attribute controls the channel path measurement facility of the channel subsystem.

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

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

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

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

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

Two sysfs attributes are added for each channel path object:

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

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

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

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

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

Examples

- To turn measurements on issue:

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

- To turn measurements off issue:

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

Channel path ID information

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

The directories that represent the CHPIDs have the form:
`/sys/devices/css0/chp0.<chpid>`

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

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

Setting a CHPID logically online or offline

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

About this task

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

Procedure

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 is logically offline issue:

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

- To set the same CHPID logically online issue:

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

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

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

Configuring a CHPID on LPAR

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

About this task

The following configuration changes are supported:

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

Procedure

To configure a CHPID, set its configure attribute by writing the value `1` to it. To deconfigure a CHPID, set its configure attribute by writing `0` to it.

Issue a command of this form:

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

where:

<CHPID>
: is a two digit hexadecimal CHPID.

<value>
: is either `1` or `0`.

To query and set the configure value using commands, see “chchp - Change channel path status” on page 545 and “lschp - List channel paths” on page 612.

Examples

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

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

This operation is equivalent to performing a Configure Channel Path Off operation on the Hardware Management Console.

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

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

- To set the same CHPID to configured issue:

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

This operation is equivalent to performing a Configure Channel Path On operation on the Hardware Management Console.

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

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

Finding the physical channel associated with a CHPID

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

About this task

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

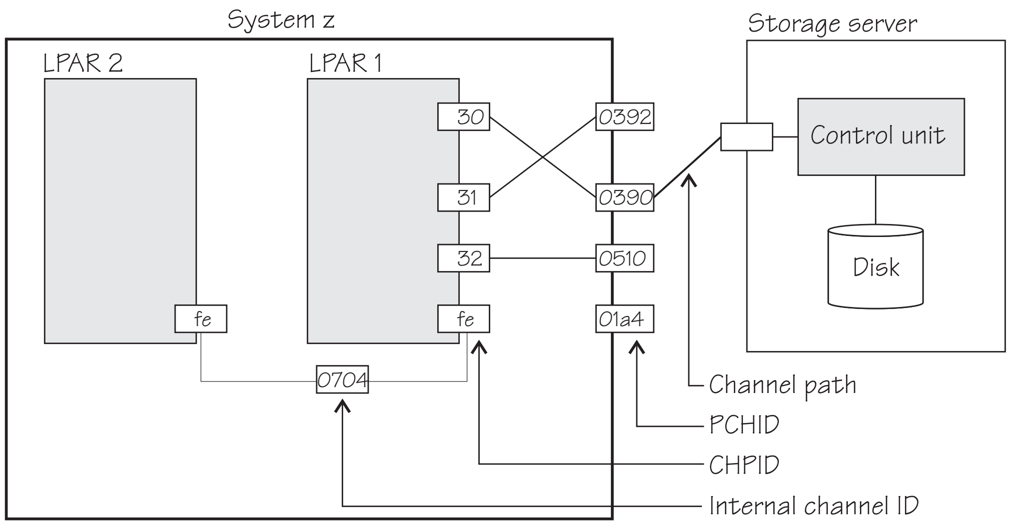


Figure 5. Relationships between CHPIDs, PCHIDs, and internal channel ID numbers.

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

- When Linux indicates that a CHPID is in an error state, you can use the PCHID number to identify the associated hardware.

- When a hardware interface requires service action, the PCHID mapping can be used to determine which CHPIDs and I/O devices will be affected.

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

Procedure

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

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

```
# lschp
```

- Find the channel-ID related files for the CHPID. These sysfs files are located under `/sys/devices/css0/chp0.<num>`, where *<num>* is the two-digit, lowercase, hexadecimal CHPID number. There are two attribute files:

  chid The channel ID number.

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

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

Example

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

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

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

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

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

CCW hotplug events

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

The hotplug events provide the following variables:

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

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

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

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

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

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

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

PCI Express support

The Peripheral Component Interconnect Express (PCIe) device driver provides support of RDMA over Converged Ethernet (RoCE).

| For more information about RoCE, see Chapter 22, “RDMA over Converged
| Ethernet,” on page 371.

PCIe functions are seen by Linux as devices, hence devices is used here synonymously. You can assign PCIe devices to LPARs in the IOCDS.

Building a kernel with the PCIe support

Control the build options for the PCIe support through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Figure 6 on page 22 summarizes the kernel configuration menu options that are relevant to the PCIe support:

```
I/O subsystem --->
    ...
    PCI support --->                                       (CONFIG_PCI)*
      Maximum number of PCI functions (1-4096)             (CONFIG_PCI_NR_FUNCTIONS)
      Maximum number of MSI interrupts (64-32768)          (CONFIG_PCI_NR_MSI)
       ...
```

Figure 6. PCIe kernel configuration menu options

CONFIG_PCI
: Enable PCI support.

  Depends on 64BIT.

CONFIG_PCI_NR_FUNCTIONS
: This option specifies the maximum number of PCI devices that this kernel supports.

CONFIG_PCI_NR_MSI
: This option defines the number of virtual interrupts the kernel provides for MSI interrupts. If you configure your system to have too few interrupts, drivers will fail to allocate MSI interrupts for all PCI devices.

The PCIe support is always compiled into the kernel.

Hotplug support

For PCI hotplug support, enable these common code configuration options:
- CONFIG_HOTPLUG
- CONFIG_PCI_HOTPLUG
- CONFIG_HOTPLUG_PCI_S390

Hotplug support enables simple online and offline configuration through the `power` sysfs attribute.

Setting up the PCIe support

If the PCI support has been compiled into the kernel, PCIe device scanning is enabled by default. You can disable it by specifying the kernel parameter `pci=off`.

Attention: Other PCI kernel parameters do not apply to System z and might have adverse effects on your system.

PCIe devices are automatically configured during the system boot process. In contrast to most System z devices, all PCIe devices that are in a configured state are automatically set online. PCIe devices that are in stand-by state are not automatically enabled.

Using PCIe hotplug

Use PCIe hotplug to change the availability of a shared PCIe device.

About this task

Only one LPAR can access a PCIe device. Other LPARs can be candidates for access. Use the HMC or SE to define which LPAR is connected and which LPARs are on the candidate list. A PCIe device that is defined, but not yet used, is shown as a PCIe slot in Linux.

On Linux, you use the `power` sysfs attribute of a PCIe slot to connect the device to the LPAR where Linux runs. While a PCIe device is connected to one LPAR, it is in the reserved state for all other LPARs that are in the candidates list. A reserved PCIe device is invisible to the operating system. The slot is removed from sysfs.

Procedure

The `power` attribute of a slot contains 0 if a PCIe device is in stand-by state, or 1 if the device is configured and usable.

1. Locate the slot for the card you want to work with. To locate the slot, read the `function_id` attribute of the PCIe device from sysfs. For example, to read the `/sys/bus/pci/devices/0000:00:00.0/function_id` issue:

   ```
   # cat /sys/bus/pci/devices/0000:00:00.0/function_id
   0x00000011
   ```

   where `00000011` is the slot. Alternatively, you can use the **`lscpi -v`** command to find the slot.
2. Write the value that you want to the `power` attribute:
   - Write 1 to `power` to connect the PCIe device to the LPAR in which your Linux instance is running. Linux automatically scans the device, registers it, and brings it online. For example:

     ```
     echo 1 > /sys/bus/pci/slots/00000011/power
     ```
   - Write 0 to `power` to stop using the PCIe device. The device state changes to stand-by. The PCIe device is set offline automatically. For example:

     ```
     echo 0 > /sys/bus/pci/slots/00000011/power
     ```

     A PCIe device in standby is also in the standby state to all other LPARs in the candidates list. A standby PCIe device appears as a slot, but without a PCIe device.

Recovering a PCIe device

Use the `recover` sysfs attribute to recover a PCIe device.

About this task

A message is displayed when a PCIe device enters the error state. It is not possible to automatically relieve the PCIe device from this state.

Procedure

1. Find the PCIe device directory in sysfs. PCIe device directories are of the form `/sys/devices/pci`*`<dev>`* where *<dev>* is the device ID. For example: `/sys/devices/pci0000:00/0000:00:00.0/`.
2. Write 1 to the `recover` attribute of the PCIe device. For example:

   ```
   # echo 1 > /sys/devices/pci0000:00/0000:00:00.0/recover
   ```

   After a successful recovery, the PCI device is de-registered and reprobed.

Displaying PCIe information

For each online PCIe device, there is a number of read-only attributes in sysfs that provide information about the device.

About this task

The sysfs representation of a PCIe device or slot is a directory of the form `/sys/devices/pci<device_bus_id>/<device_bus_id>`, where <*device_bus_id*> is the bus ID of the PCIe device. This sysfs directory contains a number of attributes with information about the PCIe device.

Table 4. Read-only attributes with PCIe device information

Attribute	Explanation
function_handle	Eight-character, hexadecimal PCI-function (device) handle.
function_id	Eight-character, hexadecimal PCI-function (device) ID. The ID identifies the PCIe device within the processor configuration.
pchid	Four-character, hexadecimal, physical channel ID. Specifies the slot of the PCIe adapter in the I/O drawer. Thus identifies the adapter that provides the device.
pfgid	Two-character, hexadecimal, physical function group ID.
pfip/segment0 /segment1 /segment2 /segment3	Two-character, hexadecimal, PCI-function internal path. Provides an abstract indication of the path that is used to access the PCI function. This can be used to compare the paths used by two or more PCI functions, to give an indication of the degree of isolation between them.
uid	Up to eight-character, hexadecimal, user-defined identifier.
vfn	Four-character, hexadecimal, virtual function number. If an adapter, identified by its PCHID, supports more than one PCI function, the VFN uniquely identifies the instance of that function within the adapter.

Procedure

Issue a command of this form to read an attribute:

```
# cat /sys/devices/pci<device_bus_id>/<device_bus_id>/<attribute>
```

where <*attribute*> is one of the attributes of Table 4.

Chapter 3. Kernel and module parameters

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

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

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

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

Separate kernel modules must be loaded before they can be used. This document describes module parameters as part of the syntax for loading the device driver or feature module to which they apply.

Where possible, this information describes kernel parameters with the device driver or feature to which they apply. Kernel parameters that apply to the base kernel or cannot be attributed to a particular device driver or feature are described in Chapter 53, "Selected kernel parameters," on page 703. You can also find descriptions for most of the kernel parameters in `Documentation/kernel-parameters.txt` in the Linux source tree.

Which device drivers or features are compiled into the kernel or provided as separate modules can vary between distributions. This document describes both kernel and module parameters for device drivers or features that can be either separate modules or part of the kernel image.

When you want to configure a device driver or feature, check how your distribution includes this device driver or feature. Use kernel parameters or module parameters, accordingly. To find the separate kernel modules for your distribution, list the contents of the subdirectories of `/lib/modules/<kernel-release>` in the Linux file system. In the path, <*kernel-release*> denotes the kernel level. You can query the value for <*kernel-release*> with `uname -r`.

Specifying kernel parameters

There are different methods for passing kernel parameters to Linux.

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

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

Note: Your distribution might set required kernel parameters for you. Parameters that you specify might interfere with these settings. Read `/proc/cmdline` to find out which parameters were used to start a running Linux instance.

Including kernel parameters in a boot configuration

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

Which sources of kernel parameters you can use depends on the mode in which you run **zipl**. See "zipl modes and syntax overview" on page 66 for details.

Running zipl in configuration-file mode

In configuration-file mode, you issue the **zipl** command with command arguments that identify a section in a **zipl** configuration-file.

You specify details about the boot configuration in the configuration file.

As shown in Figure 7, there are three sources of kernel parameters for **zipl** in configuration-file mode.

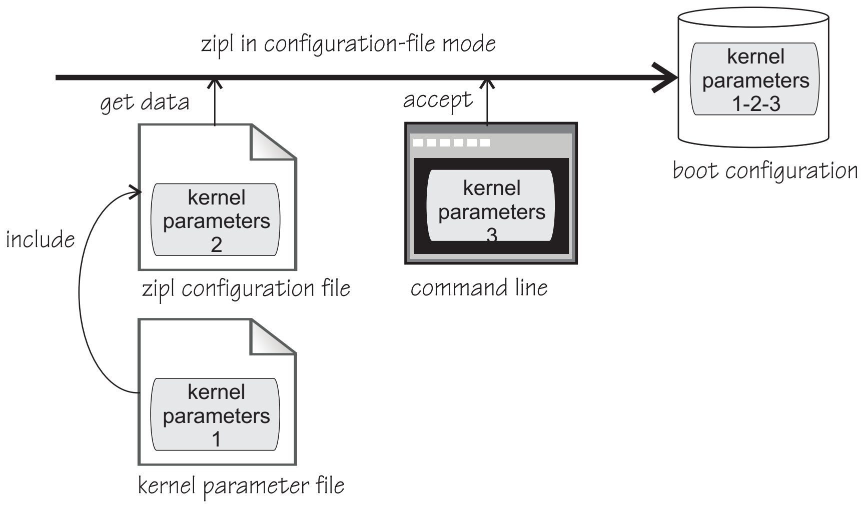


Figure 7. Sources of kernel parameters for zipl in configuration-file mode

In configuration-file mode, **zipl** concatenates the kernel parameters in the order:
1. Parameters that are specified in the kernel parameter file
2. Parameters that are specified in the **zipl** configuration-file
3. Parameters that are specified on the command line

See "zipl modes and syntax overview" on page 66 for details about the **zipl** command modes.

Running zipl in command-line mode

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

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

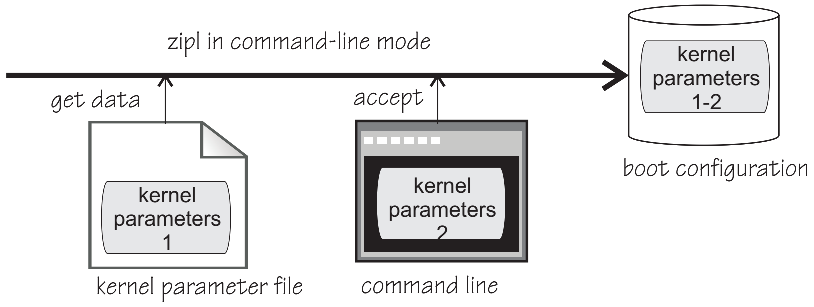


Figure 8. Sources of kernel parameters for zipl in command-line mode

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

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

See "zipl modes and syntax overview" on page 66 for details about the **zipl** command modes.

Conflicting settings and limitations

There is an override order for conflicting kernel parameter settings and there are multiple length limitations.

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

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

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

This limitation applies to the parameter string in the boot configuration. You can provide additional parameters when booting Linux. Linux accepts up to 4096 characters of kernel parameters in total. See "Adding kernel parameters to a boot configuration" on page 28.

Using a kernel parameter file

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

See "Booting from the z/VM reader" on page 97 for more details.

Specifying kernel parameters when booting Linux

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

zipl interactive boot menu on DASD

When booting Linux with a zipl interactive boot menu on a DASD boot device, you can display the menu and specify kernel parameters as you select a boot configuration. See "Example for a DASD menu configuration on z/VM" on page 94 and "Example for a DASD menu configuration (LPAR)" on page 101 for details.

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

For details, see the subsection of "Booting Linux in a z/VM guest virtual machine" on page 92 that applies to your boot device.

z/VM guest virtual machine with a SCSI boot device
When booting Linux in a z/VM guest virtual machine from a SCSI boot device, you can use the SET LOADDEV command with the SCPDATA option to specify kernel parameters. See "Booting from a SCSI device" on page 95 for details.

LPAR mode with a SCSI boot device
When booting Linux in LPAR mode from a SCSI boot device, you can specify kernel parameters in the **Operating system specific load parameters** field on the HMC Load panel. See Figure 26 on page 101.

LPAR mode with the snipl command
When using the **`snipl`** command to boot Linux in LPAR mode, you can specify additional kernel parameters with the command. See "Perform an IPL operation from a CCW device" on page 126 and "Perform an IPL or dump operation from a SCSI device" on page 127.

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

Adding kernel parameters to a boot configuration

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

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

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

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

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

Examples

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

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

Replacing all kernel parameters in a boot configuration

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

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

Note: This feature is intended for expert users who want to test a set of parameters. By replacing all parameters, you might inadvertently omit parameters that the boot configuration requires. Furthermore, you might omit parameters other than kernel parameters that distribution-specific tools include in the parameter string for use by the init process.

Read `/proc/cmdline` to find out with which parameters a running Linux instance was started (see also "Displaying the current kernel parameter line" on page 30).

Examples for kernel parameters

Typical parameters that are used for booting Linux on System z configure the console and the root file system.

`conmode=`*`<mode>`*`, condev=`*`<cuu>`*`, and console=`*`<name>`*
to set up the Linux console. See "Console kernel parameter syntax" on page 43 for details.

`noinitrd`
to suppress an initial RAM disk. Specify this parameter if your boot configuration includes an initial RAM disk but you do not want to use it.

`ramdisk_size=`*`<size>`*
to specify the size of the initial RAM disk.

`ro` to mount the root file system read-only.

`root=`*`<rootdevice>`*
to specify the device to be mounted as the root file system.

`resume=`*`<partition>`*`, noresume, no_console_suspend`
to configure suspend-and-resume support (see Chapter 7, "Suspending and resuming Linux," on page 111).

`dasd=`*`<devices>`*
to make specific DASDs available to the boot process.

This kernel parameter applies only if the DASD device driver is compiled into the kernel image. If your DASD device driver is a separate module, use the `dasd=` module parameter instead.

Exception: You can specify `dasd=` with the kernel parameters if your distribution makes special provisions to interpret it on the kernel parameter line.

Alternative: You can use `dasd_mod.dasd=` on the parameter line. With the notation *<module>.<module_parameter>*, you can pass module parameters through the kernel parameter line (see "Module parameters on the kernel parameter line" on page 30).

Displaying the current kernel parameter line

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

About this task

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

- Parameters that are evaluated by distribution-specific programs
- Parameters that are evaluated by modprobe

See also "Displaying current IPL parameters" on page 105 about displaying the parameters that were used to IPL and boot the running Linux instance.

Kernel parameters for rebooting

When rebooting, you can use the current kernel parameters or an alternative set of kernel parameters.

By default, Linux uses the current kernel parameters for rebooting. See "Rebooting from an alternative source" on page 106 about setting up Linux to use different kernel parameters for re-IPL and the associated reboot.

Specifying module parameters

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

In most cases, distribution-specific configuration tools handle module parameters, especially for modules that are also loaded automatically by the distribution.

If you load a module explicitly with a **`modprobe`** command, you can specify the module parameters as command arguments. Command-line arguments are not an option for modules that are loaded automatically by your distribution or that are included in an initial RAM disk.

Module parameters on the kernel parameter line

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

One of these user space programs is modprobe. modprobe interprets module parameters that are specified on the kernel parameter line if they are qualified with a leading module prefix and a dot.

For example, if the DASD device driver is compiled as a separate module, you can include a specification with `dasd_mod.dasd=` on the kernel parameter line. modprobe evaluates this specification as the `dasd=` module parameter when the dasd_mod module is loaded.

For some device drivers and features, the module parameters and their corresponding kernel parameters follow a naming convention that makes them effective regardless of whether the device driver or feature is compiled into the kernel or as a separate module. An example is the `zfcp.datarouter=` kernel parameter with its corresponding `datarouter=` module parameter.

If the SCSI-over-Fibre Channel device driver (zfcp device driver) is compiled into the kernel, `zfcp.datarouter=` is recognized as a kernel parameter. If the zfcp device driver is compiled as a separate module, modprobe interprets `zfcp.datarouter=` as the `datarouter=` parameter to be used when the zfcp module is loaded.

Note: Your distribution might set required module parameters for you. Parameters that you specify on the kernel parameter line might interfere with these settings.

Including module parameters in a boot configuration

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

About this task

Your distribution might use an initial RAM disk when booting. Follow these steps to provide module parameters for modules that are included in an initial RAM disk:

1. Make your configuration changes with the tools that your distribution provides.
2. Run **`mkinitrd`** to create an initial RAM disk that includes the module parameters.
3. Run **`zipl`** to include the new RAM disk in your boot configuration.

The distribution tools might run **`mkinitrd`** and **`zipl`** for you when saving changes you have made.

Displaying information about the modules

Information about module parameters is grouped in sysfs for easy access. You can display the parameters and their values for loaded modules. Not all parameters are visible.

The parameters for modules are available in sysfs as files that have names of the form:

/sys/module/*<module name>*/parameters/*<parameter name>*

Before you begin

You can display information about modules that fulfill these prerequisites:

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

Procedure

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

1. To see if the module you are interested in supports displaying parameters, try listing the module parameters for that module. Issue a command of the form:

   ```
   # ls /sys/module/<module name>/parameters
   ```

2. If the previous command listed parameters, you can display the value for the parameter you are interested in. Issue a command of the form:

   ```
   # cat /sys/module/<module name>/parameters/<parameter name>
   ```

Example

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

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

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

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

Part 2. Booting and shutdown

These device drivers and features are useful in the context of booting and shutting down Linux on System z.

Newest version

You can find the newest version of this publication at
www.ibm.com/developerworks/linux/linux390/documentation_dev.html

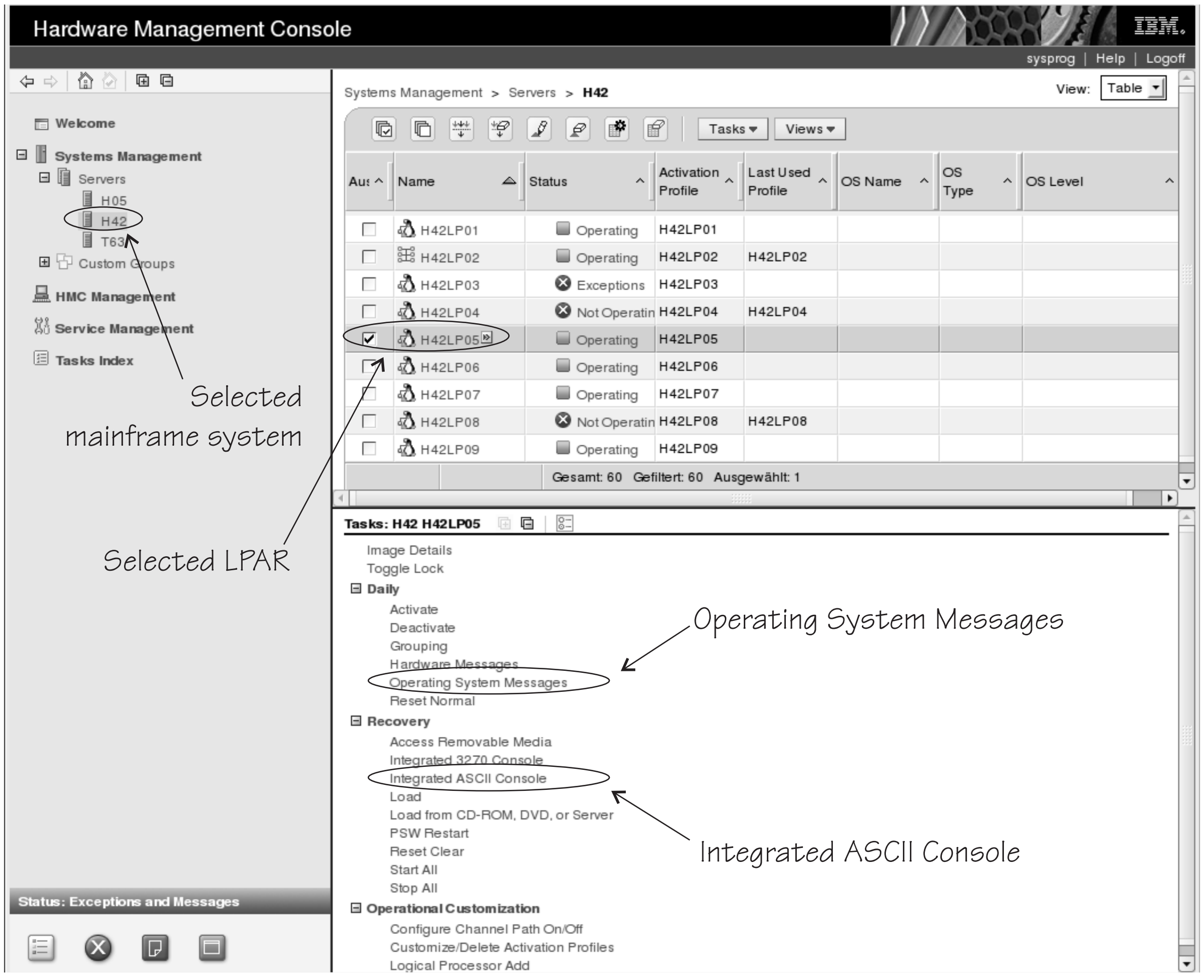
Restrictions

For prerequisites and restrictions see
www.ibm.com/developerworks/linux/linux390/development_technical.html
www.ibm.com/developerworks/linux/linux390/development_restrictions.html

Chapter 4. Console device drivers

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

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



Aus	Name	Status	Activation Profile	Last Used Profile	OS Name	OS Type	OS Level
☐	H42LP01	Operating	H42LP01				
☐	H42LP02	Operating	H42LP02	H42LP02			
☐	H42LP03	Exceptions	H42LP03				
☐	H42LP04	Not Operatin	H42LP04	H42LP04			
✓	H42LP05	Operating	H42LP05				
☐	H42LP06	Operating	H42LP06				
☐	H42LP07	Operating	H42LP07				
☐	H42LP08	Not Operatin	H42LP08	H42LP08			
☐	H42LP09	Operating	H42LP09				

Gesamt: 60 Gefiltert: 60 Ausgewählt: 1

Figure 9. Hardware Management Console

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

Console features

The console device drivers support several types of terminal devices.

HMC applets
: You can use two applets.

  Operating System Messages
  : This applet provides a line-mode terminal. See Figure 10 for an example.

  Integrated ASCII Console
  : This applet provides a full-screen mode terminal.

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

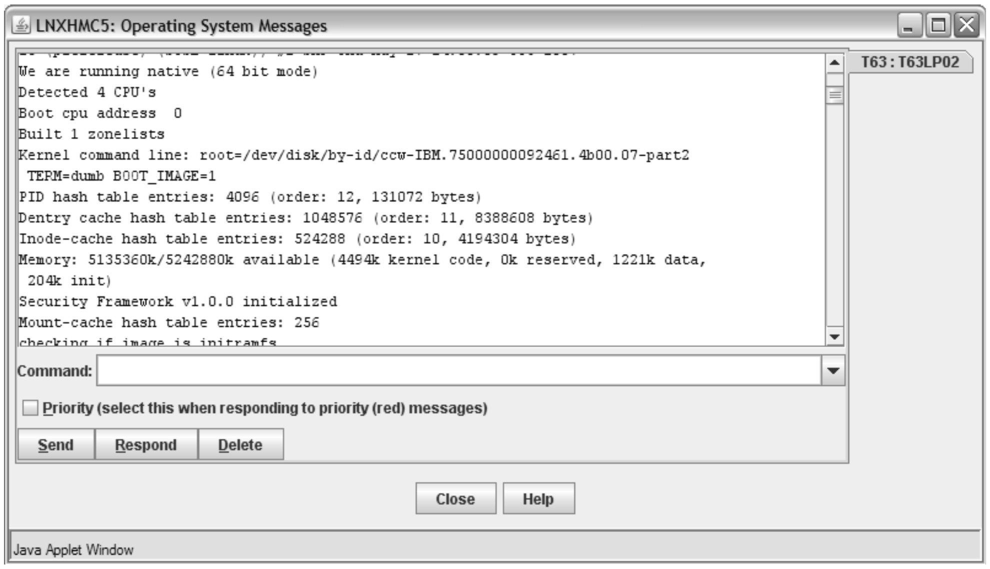
3270 terminal
: This terminal can be based on physical 3270 terminal hardware or a 3270 terminal emulation.

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

The iucvconn program
: You can use the iucvconn program from Linux on z/VM to access terminal devices on other Linux instances that run as guests of the same z/VM system.

  See *How to Set up a Terminal Server Environment on z/VM*, SC34-2596 for information about the iucvconn program.

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


LNXHMC5: Operating System Messages
T63:T63LP02
Command:
Priority (select this when responding to priority (red) messages)
Send
Respond
Delete
Close
Help
Java Applet Window

```
We are running native (64 bit mode)
Detected 4 CPU's
Boot cpu address  0
Built 1 zonelists
Kernel command line: root=/dev/disk/by-id/ccw-IBM.75000000092461.4b00.07-part2
 TERM=dumb BOOT_IMAGE=1
PID hash table entries: 4096 (order: 12, 131072 bytes)
Dentry cache hash table entries: 1048576 (order: 11, 8388608 bytes)
Inode-cache hash table entries: 524288 (order: 10, 4194304 bytes)
Memory: 5135360k/5242880k available (4494k kernel code, 0k reserved, 1221k data,
 204k init)
Security Framework v1.0.0 initialized
Mount-cache hash table entries: 256
checking if image is initramfs
```

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

What you should know about the console device drivers

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

Console terminology

Terminal and *console* have special meanings in Linux.

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

The Linux console
: is an output device that displays Linux kernel messages.

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

The HMC
: is a device that gives a system programmer control over the hardware resources, for example the LPARs. The HMC is a web application on a web server that is connected to the support element (SE). The HMC can be accessed from the SE but more commonly is accessed from a workstation within a secure network.

Console device
: in the context of the console device drivers, a device, as seen by Linux, to which Linux kernel messages can be directed.

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

Before you have a Linux terminal - the zipl boot menu

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

Depending on your setup, a zipl boot menu might be displayed when you perform an IPL. The zipl boot menu is part of the boot loader that loads the Linux kernel and is displayed before a Linux terminal is set up. The zipl boot menu is very limited in its functions. For example, there is no way to specify uppercase letters because all input is converted to lowercase characters. For more details about booting Linux, see Chapter 6, “Booting Linux,” on page 89. For more information about the zipl boot menu, see Chapter 5, “Initial program loader for System z - zipl,” on page 65.

Device and console names

Each terminal device driver can provide a single console device.

Table 5 on page 38 lists the terminal device drivers and the corresponding device names and console names.

Table 5. Device and console names

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

As shown in Table 5, the console with name ttyS0 can be provided either by the SCLP console device driver or by the 3215 line-mode terminal device driver. The system environment and settings determine which device driver provides ttyS0. For details, see the information about the conmode kernel parameter in "Console kernel parameter syntax" on page 43.

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

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

Device nodes

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

Table 6 lists the standard device nodes the terminal devices. Which terminal devices are present on your system depends on your system environment, LPAR or z/VM, on your kernel parameters, and on the presence of online 3270 devices.

Table 6. Terminal device nodes

Device driver	On LPAR	On z/VM	Major	Minor
SCLP line-mode terminal device driver	`/dev/sclp_line0`	`/dev/sclp_line0`	4	64
SCLP VT220 terminal device driver	`/dev/ttysclp0`	`/dev/ttysclp0`	4	65
3215 line-mode terminal device driver	n/a	`/dev/ttyS0`	4	64
3270 terminal device driver	`/dev/3270/tty1` to `/dev/3270/tty<N>`	`/dev/3270/tty1` to `/dev/3270/tty<N>`	227	1 - <*N*>
z/VM IUCV HVC device driver	n/a	`/dev/hvc0` to `/dev/hvc7`	229	0 - 7

Your distribution must create these device nodes early in the boot process, for example, with udev. Otherwise, Linux might not boot and leave you without a command prompt for creating the nodes yourself. In this case, you can create the nodes from a support system that has access to the root file system of the failed Linux instance.

Terminal modes

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

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

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

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

Table 7 summarizes when to expect which terminal mode.

Table 7. Terminal modes

Accessed through	Environment	Device driver	Mode
Operating System Messages applet on the HMC	LPAR	SCLP line-mode terminal device driver	Line mode
z/VM emulation of the HMC **Operating System Messages** applet	z/VM	SCLP line-mode terminal device driver	Line mode
Integrated ASCII Console applet on the HMC	z/VM or LPAR	SCLP VT220 terminal device driver	Full-screen mode
3270 terminal hardware or emulation	z/VM with CONMODE=3215	3215 line-mode terminal device driver	Line mode
3270 terminal hardware or emulation	z/VM with CONMODE=3270	3270 terminal device driver	Block mode
iucvconn program	z/VM	z/VM IUCV HVC device driver	Full-screen mode

The 3270 terminal device driver provides three different views. See “Switching the views of the 3270 terminal device driver” on page 56 for details.

How console devices are accessed

How you can access console devices depends on your environment.

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

Using the HMC for Linux in an LPAR

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

Figure 11 on page 40 shows the possible terminal devices for Linux instances that run directly in an LPAR.

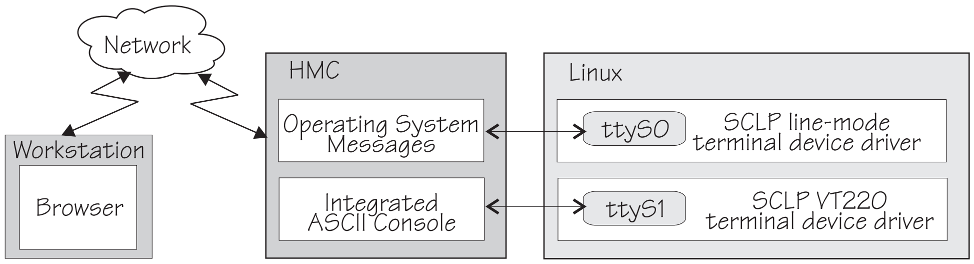

Network
Workstation
Browser
HMC
Operating System Messages
Integrated ASCII Console
Linux
ttyS0
SCLP line-mode terminal device driver
ttyS1
SCLP VT220 terminal device driver

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

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

Using the HMC for Linux on z/VM

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

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

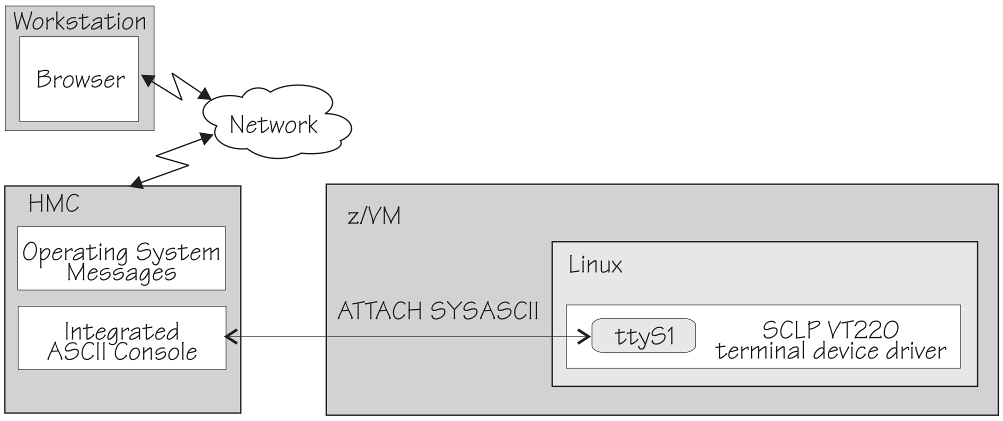


Figure 12. Accessing terminal devices from the HMC for Linux on z/VM

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

Using 3270 terminal hardware or a 3270 terminal emulation

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

Figure 13 on page 41 illustrates how z/VM can handle the 3270 communication.

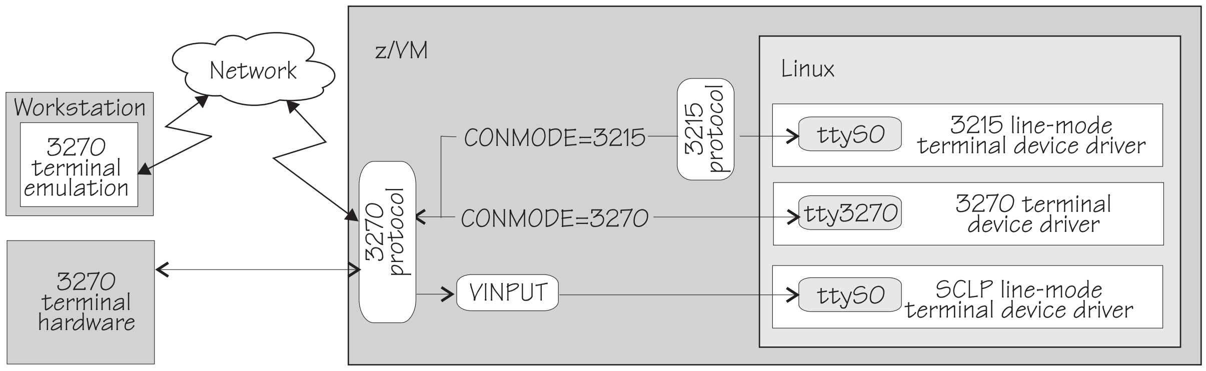


Figure 13. Accessing terminal devices from a 3270 device

Note: Figure 13 shows two console devices with the name ttyS0. Only one of these devices can be present at any one time.

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

`CONMODE=3270`
: connects the 3270 terminal hardware or emulation to the 3270 terminal device driver in the Linux kernel.

`VINPUT`
: is a z/VM CP command that directs input to the ttyS0 device provided by the SCLP line-mode terminal device driver. In a default z/VM environment, ttyS0 is provided by the 3215 line-mode terminal device driver. You can use the conmode kernel parameter to make the SCLP line-mode terminal device driver provide ttyS0 (see "Console kernel parameter syntax" on page 43).

Using iucvconn on Linux on z/VM

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

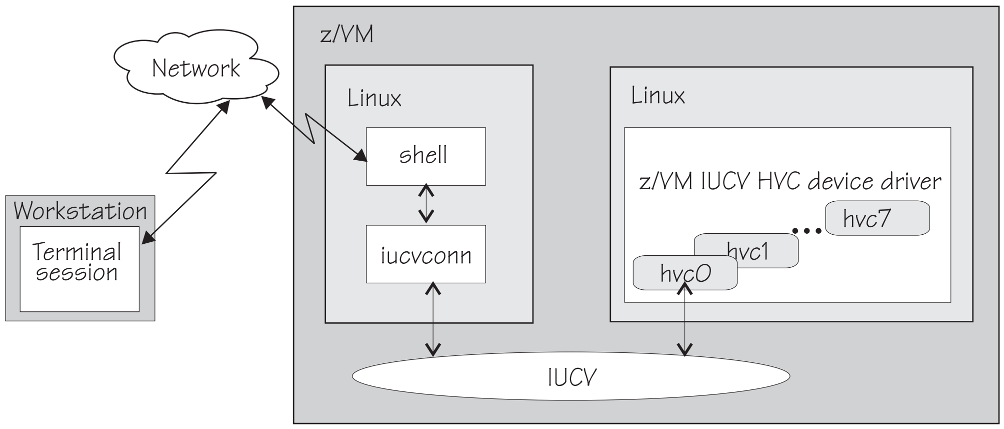


Figure 14. Accessing terminal devices from a peer Linux instance

As illustrated in Figure 14, you access the devices with the iucvconn program from another Linux instance. Both Linux instances are guests of the same z/VM system.

IUCV provides the communication between the two Linux instances. With this setup, you can access terminal devices on Linux instances with no external network connection.

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

Building a kernel with the console device drivers

Control the build options for the console device drivers through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Figure 15 summarizes the kernel configuration menu options that are relevant to the console device drivers. You must select at least one option that adds support for kernel messages to an output device. Select all options to compile a Linux kernel that supports all available console devices with all their features.

```
Device Drivers --->
   ...
   Character devices --->
      ...
      z/VM IUCV Hypervisor console support (VM only)                  (CONFIG_HVC_IUCV)
      ...
      --- S/390 character device drivers (depends on S390) ---
      Support for locally attached 3270 terminals                     (CONFIG_TN3270)
      ├─ Support for tty input/output on 3270 terminals               (CONFIG_TN3270_TTY)
      ├─ Support for fullscreen applications on 3270 terminals        (CONFIG_TN3270_FS)
      └─ Support for console on 3270 terminal                         (CONFIG_TN3270_CONSOLE)*
      Support for 3215 line mode terminal                             (CONFIG_TN3215)
      └─ Support for console on 3215 line mode terminal               (CONFIG_TN3215_CONSOLE)
      Support for SCLP line mode terminal                             (CONFIG_SCLP_TTY)
      └─ Support for console on SCLP line mode terminal               (CONFIG_SCLP_CONSOLE)
      Support for SCLP VT220-compatible terminal                      (CONFIG_SCLP_VT220_TTY)
      └─ Support for console on SCLP VT220-compatible terminal        (CONFIG_SCLP_VT220_CONSOLE)
```

Figure 15. Console kernel configuration menu options

CONFIG_HVC_IUCV
: Adds the z/VM IUCV HVC device driver and IUCV support. This device driver supports terminal access through the iucvconn program to instances of Linux on z/VM.

CONFIG_TN3270
: Adds the 3270 terminal device driver. This device driver supports IBM 3270 terminal hardware and 3270 terminal emulations.

CONFIG_TN3270_TTY
: Adds the terminal input and output support to the 3270 terminal device driver.

CONFIG_TN3270_FS
: Adds limited support for full-screen applications the 3270 terminal device driver.

CONFIG_TN3270_CONSOLE
Adds the operating system messages view to the 3270 terminal device driver.

CONFIG_TN3215
Adds the 3215 line-mode terminal device driver. Through a translation between the 3215 protocol and the 3270 protocol within z/VM, this device driver supports IBM 3270 terminal hardware and 3270 terminal emulations.

CONFIG_TN3215_CONSOLE
Adds support for kernel messages to the 3215 line-mode terminal device driver.

CONFIG_SCLP_TTY
Adds the SCLP line-mode terminal device driver. This device driver supports the **Operating System Messages** applet on the HMC.

CONFIG_SCLP_CONSOLE
Adds support for kernel messages to the SCLP line-mode terminal device driver.

CONFIG_SCLP_VT220_TTY
Adds the VT220-compatible SCLP VT220 terminal device driver. This device driver supports the **Integrated ASCII console** applet on the HMC.

CONFIG_SCLP_VT220_CONSOLE
Adds support for kernel messages to the SCLP VT220 terminal device driver.

Linux requires a device for writing kernel messages early in the boot process. Always compile the console device drivers and their components into the kernel.

Setting up the console device drivers

You configure the console device drivers through kernel parameters. You also might have to enable user logins on terminals, and ensure suitable terminal settings.

Console kernel parameter syntax

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

The `condev=` kernel parameter applies only to P/390 and is described in "Defining a 3215 terminal to Linux on P/390" on page 53.

Console kernel parameter syntax

```
                                 ┌─────────────────────────┐
                                 ▼                         │ (1)
►►─┬─────────────────────────┬─── console=<console_name>─┴──────────────────────────►
   └─ conmode=──┬─hwc──┬─────┘
                ├─sclp─┤
                ├─3215─┤
                └─3270─┘

   ┌─ sclp_con_drop=0─┐  ┌─ sclp_con_pages=6───┐ (2)
►──┼──────────────────┼──┼─────────────────────┼───────────────────────────────────►
   └─ sclp_con_drop=1─┘  └─ sclp_con_pages=<n>─┘

   ┌─ hvc_iucv=1─────────────────────┐                                   (3)
►──┼─────────────────────────────────┼──┬──────────────────────────────────┬──────►◄
   └─ hvc_iucv=<number_of_devices>───┘  │                  ┌─,───────────┐ │
                                        │                  ▼             │ │
                                        └─ hvc_iucv_allow=───<z/VM user ID>─┴─┘
```

Notes:

1 If you specify both the `conmode=` and the `console=` parameter, specify them in the sequence that is shown, `conmode=` first.

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

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

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 8 shows how the device driver that is used by default depends on the environment.

Table 8. Default device driver for the line-mode terminal device

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

You can use the conmode= parameter to override the default.

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

You need this specification if you want to use the z/VM CP VINPUT command ("Using a z/VM emulation of the HMC Operating System Messages applet" on page 59).

`3270`
specifies the 3270 device driver.

`3215`
specifies the 3215 device driver.

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

The preferred console is used as an initial terminal device, beginning at the stage of the boot process when the 'init'-program is called. Messages from programs that run at this stage are displayed on the preferred console only. Multiple terminal devices can be activated to receive Linux kernel messages, but only one of the activated terminal devices can be the preferred console.

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

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

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

<console_name> is the console name that is associated with the terminal device to be activated to receive Linux kernel messages. Of the terminal devices that are provided by the z/VM IUCV HVC device driver, only `hvc0` can be activated. Specify the console names as shown in Table 5 on page 38.

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

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

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

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

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

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

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

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

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

Examples

- To activate `ttyS1` in addition to `ttyS0`, and to use `ttyS1` as the preferred console, specify:

  ```
  console=ttyS1
  ```
- To activate `ttyS1` in addition to `ttyS0`, and to keep `ttyS0` as the preferred console, specify:

  ```
  console=ttyS1 console=ttyS0
  ```
- To use an emulated HMC Operating System Messages applet in a z/VM environment, specify:

  ```
  conmode=sclp
  ```
- To activate `hvc0` in addition to `ttyS0`, use `hvc0` as the preferred console, configure the z/VM IUCV HVC device driver to provide four devices, and limit the z/VM guest virtual machines that can connect to HVC terminal devices to `lxtserv1` and `lxtserv2`, specify:

  ```
  console=hvc0 hvc_iucv=4 hvc_iucv_allow=lxtserv1,lxtserv2
  ```
- The following specification selects the SCLP line-mode terminal and configures 32 4-KB pages (128 KB) for the output buffer. If buffer pages run out, the SCLP line-mode terminal device driver does not wait for pages to be written to an output device. Instead of pausing, it reuses output buffer pages at the expense of losing content.

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

Setting up a z/VM guest virtual machine for iucvconn

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

See “Setting up your z/VM guest virtual machine for IUCV” on page 360 for details about setting up the z/VM guest virtual machine.

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

Setting up a line-mode terminal

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

The preferred user access to a running Linux on System z is through a user login that runs, for example, in an ssh session. See “Terminal modes” on page 38 for information about the available line-mode terminals.

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

```
# export TERM=dumb
```

Setting up a full-screen mode terminal

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

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

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

```
# export TERM=linux
```

Setting up a terminal provided by the 3270 terminal device driver

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

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

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

```
# export TERM=linux
```

Enabling a terminal for user logins

Inadequate settings for user logins can render terminals inaccessible.

Important

When enabling a terminal for user logins be aware that:

- Multiple logins on the same terminal render the terminal inaccessible.

  Do not inadvertently configure two logins on the preferred console, one explicitly through the terminal-specific device node and a second one through the generic `/dev/console`.
- At least one operational terminal device must be set up for logins.

The terminal that is associated with the preferred console and the availability of terminals in general differ, depending on your environment and on your kernel parameters (see “Console kernel parameter syntax” on page 43).

Enabling a terminal for user logins using inittab

If your distribution uses inittab, you can use an inittab entry to allow user logins on 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 <dev>
```

where:

<id>
: is a unique identifier for the entry in the inittab file.

<dev>
: specifies the device node of the terminal, omitting the leading `/dev/` (see "Device nodes" on page 38). For example, instead of specifying `/dev/sclp_line0`, specify `sclp_line0`.

With mingetty, you must explicitly export the TERM environment variable with the terminal name as explained in "Setting up a full-screen mode terminal" on page 47. The terminal name indicates the capabilities of the terminal device. Examples for terminal names are `linux`, `dumb`, `xterm`, or `vt220`.

Instead of mingetty, you can use agetty, which can set the TERM environment variable at startup.

To set the TERM environment variable to `linux` and enable user logins with the agetty program add a line of this form to the `/etc/inittab` file:

```
<id>:2345:respawn:/sbin/agetty -L 9600 <dev> linux
```

where *<id>* and *<dev>* have the same meanings as in the mingetty example.

The `/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 "Device nodes" on page 38 for the device node names. If an existing entry uses a name that you cannot map to the names of "Device nodes" on page 38, you can comment it out and replace it.

For more information, see the man page for the inittab file.

Example

To enable a device `ttyS0` for user logins with mingetty specify, for example:

```
1:2345:respawn:/sbin/mingetty --noclear ttyS0
```

Enabling a terminal for user logins using Upstart

If your distribution uses Upstart, you can use an Upstart job file to allow user logins on a terminal.

To enable user logins with Upstart, create an Upstart job file with the following content:

```
start on runlevel [2345]
stop on runlevel [01]

respawn
exec /sbin/mingetty --noclear <dev>
```

You can use a file name of your choice. The directory where you must locate the file depends on your distribution.

In the sample file, <*dev*> specifies the device node of the terminal, omitting the leading `/dev/` (see "Device nodes" on page 38). For example, instead of specifying `/dev/sclp_line0`, specify `sclp_line0`.

With mingetty, you must explicitly export the TERM environment variable with the terminal name as explained in "Setting up a full-screen mode terminal" on page 47. The terminal name indicates the capabilities of the terminal device. Examples for terminal names are `linux`, `dumb`, `xterm`, or `vt220`.

Instead of mingetty, you can use agetty, which can set the TERM environment variable at startup.

To set the TERM environment variable to `linux` and enable user logins with Upstart create an Upstart job file with the following content:

```
start on runlevel [2345]
stop on runlevel [01]

respawn
exec /sbin/agetty -L 9600 <dev> linux
```

Example

The following Upstart job file enables a device `hvc0` for user logins with mingetty.

```
start on runlevel [2345]
stop on runlevel [01]

respawn
exec /sbin/mingetty --noclear hvc0
```

Enabling a terminal for user logins using systemd

If your distribution uses systemd, you can enable getty services to allow user logins on a terminal.

Before you begin

You must explicitly enable user logins for the HVC terminals `hvc1` to `hvc7` and for any dynamically attached virtual or real 3270 terminals. On all other terminals that are available in your environment, including `hvc0` and `3270/tty1`, systemd automatically enables user logins for you.

Tip: Instead of the getty service, use the ttyrun service to enable user logins on HVC terminals (see "Using systemd" on page 52). With the ttyrun service, you are free to change the conditions that affect the availability of HVC terminals.

Procedure

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

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

   ```
   # systemctl enable serial-getty@<terminal>.service
   ```

   where <*terminal*> specifies one of the terminals `hvc1` to `hvc7` or `3270-tty<N>` and <*N*> is an integer greater than 1.

Note: You specify terminal `3270/tty<N>` as `3270-tty<N>`.

2. Optional: Start the new getty service by issuing a command of this form:

```
# systemctl start serial-getty@<terminal>.service
```

Results

At the next system start, systemd automatically starts the getty service for you.

Example

For `3270/tty2`, issue:

```
# systemctl enable serial-getty@3270-tty2.service
# systemctl start serial-getty@3270-tty2.service
```

Preventing respawns for non-operational terminals

An inittab entry, Upstart job file, or systemd service unit for user logins on an unavailable or non-operational terminal can cause recurring respawns of the getty program.

Failing respawns increase system and logging activities.

The availability of some terminals depends on the environment where the Linux instance runs, LPAR or z/VM, and on terminal-related kernel parameters. For more information, see the explanations for the `conmode=` and `hvc_iucv=` kernel parameters in "Console kernel parameter syntax" on page 43.

You can use the ttyrun program to prevent respawns if a terminal is not available or not operational. With suitable inittab entries, Upstart job files, or systemd ttyrun service units in place, you can freely change conditions that affect the presence of terminals.

By default, ttyrun runs quietly. Use the verbose option, **`-V`**, to obtain syslog messages, for example, if a terminal device is not available.

Using inittab

Typical inittab entries differ, depending on the type of terminal and whether the availability of the terminal is assured.

To use ttyrun with an entry for mingetty, change the entry to this form:

```
<id>:2345:respawn:/sbin/ttyrun <dev> /sbin/mingetty --noclear %t
```

To use ttyrun with an entry for agetty, change the entry to this form:

```
<id>:2345:respawn:/sbin/ttyrun <dev> /sbin/agetty -L 9600 %t <term>
```

where:

<id>
: is a unique identifier for the entry in the inittab file.

<dev>
: specifies the device node of the terminal, omitting the leading `/dev/` (see "Device nodes" on page 38). For example, instead of specifying `/dev/sclp_line0`, specify `sclp_line0`.

<term>
: specifies the terminal name. The terminal name indicates the capabilities of the terminal device. Examples for terminal names are `linux`, `dumb`, `xterm`, or `vt220`.

  With mingetty, you must explicitly export the TERM environment variable with the terminal name as explained in “Setting up a full-screen mode terminal” on page 47.

%t is a variable that the ttyrun program resolves to the device node that is specified for *<dev>*.

Example

To enable terminal devices `hvc0` through `hvc3` for user logins with agetty and to take into account that the terminals might not be operational, specify, for example:

```
h0:2345:respawn:/sbin/ttyrun -V hvc0 /sbin/agetty -L 9600 %t linux
h1:2345:respawn:/sbin/ttyrun hvc1 /sbin/agetty -L 9600 %t linux
h2:2345:respawn:/sbin/ttyrun hvc2 /sbin/agetty -L 9600 %t linux
h3:2345:respawn:/sbin/ttyrun hvc3 /sbin/agetty -L 9600 %t linux
```

These terminal devices are operational only in a z/VM environment. In addition, they depend on the `hvc_iucv=` kernel parameter (see “Console kernel parameter syntax” on page 43). The verbose option, **-V**, in the first line causes a syslog entry if the `hvc0` terminal device is not available. The other lines omit this option, so no syslog entries are created if any of the other terminal devices are unavailable.

Using Upstart

Typical Upstart job files differ, depending on the type of terminal and whether the availability of the terminal is assured.

To use ttyrun with an Upstart job file for mingetty, use a file of this form:

```
start on runlevel [2345]
stop on runlevel [01]

respawn
normal exit <value>
exec /sbin/ttyrun -e <value> <dev> /sbin/mingetty --noclear %t
```

To use ttyrun with an Upstart job file for agetty, use a file of this form:

```
start on runlevel [2345]
stop on runlevel [01]

respawn
normal exit <value>
exec /sbin/ttyrun -e <value> <dev> /sbin/agetty -L 9600 <term> %t
```

where:

<value>
: specifies an integer in the range 1 to 255. See the ttyrun man page for details.

<dev>
: specifies the device node of the terminal, omitting the leading `/dev/` (see “Device nodes” on page 38). For example, instead of specifying `/dev/sclp_line0`, specify `sclp_line0`.

<term>
: specifies the terminal name. The terminal name indicates the capabilities of the terminal device. Examples for terminal names are `linux`, `dumb`, `xterm`, or `vt220`.

With mingetty, you must explicitly export the TERM environment variable with the terminal name as explained in “Setting up a full-screen mode terminal” on page 47.

%t is a variable that the ttyrun program resolves to the device node that is specified for <*dev*>.

See the init(5) man page for details about the individual lines in the Upstart job file.

Example

To enable terminal device `hvc0` for user logins with mingetty and to take into account that the terminal might not be operational, the complete Upstart job file could look like this:

```
start on runlevel [2345]
stop on runlevel [01]

respawn
normal exit 123
exec /sbin/ttyrun -V -e 123 hvc0 /sbin/mingetty --noclear %t
```

Enabling `hvc1` through `hvc7`, requires a similar Upstart job file for each terminal device, with the respective device node specified on the exec line. These terminal devices are operational only in a z/VM environment. In addition, they depend on the `hvc_iucv=` kernel parameter (see “Console kernel parameter syntax” on page 43). The verbose option, **-V**, causes a syslog entry if the `hvc0` terminal device is not available.

Using systemd

If your distribution uses systemd, use the ttyrun service to prevent systemd from respawning the getty program for non-operational HVC terminals (`hvc1` to `hvc7`).

Before you begin

You require a service unit that is included in the s390-tools package. When installing the s390-tools package, you must set the installation directory for the unit with the SYSTEMDSYSTEMUNITDIR make variable. This directory depends on where your distribution maintains the systemd system unit directory, usually `/lib/systemd/system` or `/etc/systemd/system`. If you do not set the SYSTEMDSYSTEMUNITDIR parameter, the template is not installed with the s390-tools package.

About this task

If user logins are enabled on unavailable HVC terminals `hvc1` to `hvc7`, systemd might keep respawning the getty program. To be free to change the conditions that affect the availability of these terminals, use the ttyrun service to enable user logins for them. HVC terminals are operational only in a z/VM environment, and they depend on the `hvc_iucv=` kernel parameter (see “Console kernel parameter syntax” on page 43).

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

Procedure

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

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

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

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

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

Results

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

Example

For `hvc1`, issue:

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

Setting up the code page for an x3270 emulation on Linux

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

Add these settings:

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

Defining a 3215 terminal to Linux on P/390

The default console device driver for Linux on IBM PC Server System/390® (P/390) is the 3215 console device driver.

You can change this default with the conmode kernel parameter (see "Console kernel parameter syntax" on page 43). If you use the 3215 console device driver on a P/390 system, you must provide the device number of the 3215 terminal to Linux. Use the condev kernel parameter to specify the device number.

condev kernel parameter syntax

```
►►──condev=<cuu>──────────────────────────────────────────────►◄
```

<*cuu*> is the device Control Unit and Unit number, and can be expressed in hexadecimal form (preceded by `0x`) or in decimal form.

Example

To instruct the device driver to use device number 0x001F for the 3215 terminal, specify:

```
condev=0x001f
```

or:

```
condev=31
```

Working with Linux terminals

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

- "Using the terminal applets on the HMC"
- "Accessing terminal devices over z/VM IUCV" on page 55
- "Switching the views of the 3270 terminal device driver" on page 56
- "Setting a CCW terminal device online or offline" on page 56
- "Entering control and special characters on line-mode terminals" on page 57
- "Using the magic sysrequest feature" on page 58
- "Using a z/VM emulation of the HMC Operating System Messages applet" on page 59
- "Using a 3270 terminal in 3215 mode" on page 62

Using the terminal applets on the HMC

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

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

- On an HMC, you can open each applet only once.
- Within an LPAR, there can be only one active terminal session for each applet, even if multiple HMCs are used.
- A particular Linux instance supports only one active terminal session for each applet.
- Security hint: Always end a terminal session by explicitly logging off (for example, type "exit" and press Enter). Simply closing the applet leaves the session active and the next user to open the applet resumes the existing session without a logon.
- Slow performance of the HMC is often due to a busy console or increased network traffic.

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

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

Accessing terminal devices over z/VM IUCV

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

About this task

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

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

Procedure

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

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

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

   where:

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

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

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

   ```
   # iucvconn LXGUEST1 lnxhvc0
   ```

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

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

Security hint

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

Switching the views of the 3270 terminal device driver

The 3270 terminal device driver provides three different views.

Use function key 3 (PF3) to switch between the views (see Figure 16).

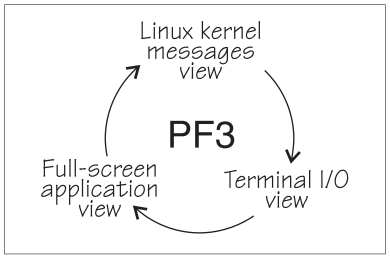


Figure 16. Switching views of the 3270 terminal device driver

The availability of the individual views depends on the configuration options that were selected when the kernel was compiled (see “Building a kernel with the console device drivers” on page 42). In addition, the Linux kernel messages view is available only if the terminal device is activated for Linux kernel messages.

The full-screen application view is available only if there is an application that uses this view, for example, the ned editor. Be aware that the 3270 terminal provides only limited full-screen support. The full-screen application view of the 3270 terminal is not intended for applications that require vt220 capabilities. The application itself must create the 3270 data stream.

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

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

Setting a CCW terminal device online or offline

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

About this task

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

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

  If this device exists, it comes online early during the Linux boot process. In a default z/VM environment, the device number for this device is 0009. In sysfs, it is represented as /sys/bus/ccw/drivers/3270/0.0.0009. You need not set this device online and you must not set it offline.

- CCW terminal devices through which users can log in to Linux with the CP DIAL command.

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

See *z/VM CP Commands and Utilities Reference*, SC24-6175 for more information about the DEF GRAF and DIAL commands.

Procedure

You can use the **chccwdev** command (see "chccwdev - Set CCW device attributes" on page 543) to set a CCW terminal device online or offline. Alternatively, you can write 1 to the device's online attribute to set it online, or 0 to set it offline.

Examples

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

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

  Alternatively issue:

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

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

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

  Alternatively issue:

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

Entering control and special characters on line-mode terminals

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

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

Table 9 summarizes how you can use the caret character (^) to enter some control characters and to enter strings without appended newline characters.

Table 9. Control and special characters on line-mode terminals

For the key combination	Enter	Usage
Ctrl+C	^c	Cancel the process that is running in the foreground of the terminal.
Ctrl+D	^d	Generate an end of file (EOF) indication.

Table 9. Control and special characters on line-mode terminals (continued)

For the key combination	Enter	Usage
`Ctrl+Z`	`^z`	Stop a process.
n/a	`^n`	Suppresses the automatic generation of a new line. Thus, you can enter single characters; for example, the characters that are needed for yes/no answers in some utilities.

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

Using the magic sysrequest feature

You can call the magic sysrequest functions from a line-mode terminal and, depending on your setup, from the hvc0 terminal device.

Before you begin: The magic sysrequest functions are available only on Linux instances that were compiled with the common code kernel configuration option CONFIG_MAGIC_SYSRQ.

To call the magic sysrequest functions on a line-mode terminal, enter the 2 characters "^-" (caret and hyphen) followed by a third character that specifies the particular function.

You can also call the magic sysrequest functions from the hvc0 terminal device if it is present and is activated to receive Linux kernel messages. To call the magic sysrequest functions from hvc0, enter the single character Ctrl+o followed by the character for the particular function.

Table 10 provides an overview of the commands for the magic sysrequest functions:

Table 10. Magic sysrequest functions

On line-mode terminals, enter	On hvc0, enter	To
`^-b`	Ctrl+o `b`	Re-IPL immediately (see "lsreipl - List IPL and re-IPL settings" on page 629).
`^-c`	Ctrl+o `c`	Crash through a forced kernel panic.
`^-s`	Ctrl+o `s`	Emergency sync all file systems.
`^-u`	Ctrl+o `u`	Emergency remount all mounted file systems read-only.
`^-t`	Ctrl+o `t`	Show task info.
`^-m`	Ctrl+o `m`	Show memory.
`^-` followed by a digit (0 - 9)	Ctrl+o followed by a digit (0 - 9)	Set the console log level.
`^-e`	Ctrl+o `e`	Send the TERM signal to end all tasks except init.
`^-i`	Ctrl+o `i`	Send the KILL signal to end all tasks except init.

Table 10. Magic sysrequest functions (continued)

On line-mode terminals, enter	On hvc0, enter	To
`^-p`	Ctrl+o p	See "Obtaining details about the CPU-measurement facilities" on page 518.

Note: In Table 10 on page 58 Ctrl+o means pressing O while holding down the control key.

Table 10 on page 58 lists the main magic sysrequest functions that are known to work on Linux on System z. For a more comprehensive list of functions, see `Documentation/sysrq.txt` in the Linux source tree. Some of the listed functions might not work on your system.

Activating and deactivating the magic sysrequest feature

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

Procedure

From a Linux terminal or a command prompt, enter the following command to activate the magic sysrequest feature:

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

Enter the following command to deactivate the magic sysrequest feature:

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

Triggering magic sysrequest functions from procfs

If you are working from a terminal that does not support a key sequence or combination to call magic sysrequest functions, you can trigger the functions through procfs.

Procedure

Write the character for the particular function to `/proc/sysrq-trigger`.
You can use this interface even if the magic sysrequest feature is not activated as described in "Activating and deactivating the magic sysrequest feature."

Example

To set the console log level to 9, enter:

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

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

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

About this task

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

The emulation requires a terminal device that is provided by the SCLP line-mode terminal device driver. To use the emulation, you must override the default device driver for z/VM environments (see “Console kernel parameter syntax” on page 43).

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

The examples in the sections that follow show the input line of a 3270 terminal or terminal emulator (for example, x3270). Omit the leading `#CP` if you are in CP read mode. For more information about VINPUT, see *z/VM CP Commands and Utilities Reference*, SC24-6175.

Priority and non-priority commands

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

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

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

Linux does not distinguish priority and non-priority commands.

Example

The specifications:

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

and

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

are equivalent.

Case conversion

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

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

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

This behavior and the delimiter are adjustable at build-time by editing the driver sources.

Examples

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

-

  ```
  #cp vinput vmsg ls -l
  CP VINPUT VMSG LS -L
  ls -l
  ...
  ```

- The following input would result in a bash command that contains a variable $path, which is not defined in lowercase:

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

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

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

Using the escape character

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

Example

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

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

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

Using the end-of-line character

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

If you are using the standard settings according to "Using a 3270 terminal in 3215 mode" on page 62, you must specify "# to pass # to Linux.

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

Example

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

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

The escape character prevents the number sign from being interpreted as an end-of-line character:

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

Simulating the Enter and Spacebar keys

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

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

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

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

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

Using a 3270 terminal in 3215 mode

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

The default line-editing symbols depend on your terminal emulator. You can reassign the symbols by changing the settings of LINEND, TABCHAR, CHARDEL, LINEDEL, or ESCAPE with the **CP TERMINAL** command. Table 11 shows the most commonly used settings:

Table 11. Line edit characters

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

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

Examples

The following examples assume the settings of Table 11 on page 62 with the opening bracket character ([) as the "delete line" character.

- To specify a tab character specify:
  ```
  "|
  ```
- To specify a double quotation mark character, specify:
  ```
  ""
  ```
- If you type the character string:
  ```
  #CP HALT#CP ZIPL 190[#CP IPL 1@290 PARM vmpoff=""MSG OP REBOOT"#IPL 290""
  ```
  the actual commands that are received by CP are:
  ```
  CP HALT
  CP IPL 290 PARM vmpoff="MSG OP REBOOT#IPL 290"
  ```

Chapter 5. Initial program loader for System z - zipl

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

Instead of preparing a dump device with the zipl tool you can also use the kdump infrastructure. To use kdump, no preparation with **zipl** is necessary. For more information about the kdump infrastructure and the dump tools that **zipl** installs, see *Using the Dump Tools*, SC33-8412.

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

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

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

zipl base functions

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

Table 12. zipl base functions

Base function	Command line short option	Command line long option	Configuration file option
Install a boot loader See "Preparing a boot device" on page 68 for details.	`-i`	`--image`	`image=`
Prepare a DASD, SCSI, or tape dump device See "Preparing a DASD, SCSI, or tape dump device" on page 75 for details.	`-d`	`--dumpto`	`dumpto=`
Prepare a list of ECKD volumes for a multi-volume dump See "Preparing a multi-volume dump on ECKD DASD" on page 77 for details.	`-M`	`--mvdump`	`mvdump=`

Table 12. zipl base functions (continued)

Base function	**Command line short option**	**Command line long option**	**Configuration file option**
Install a menu configuration See "Installing a menu configuration" on page 78 for details.	`-m`	`--menu`	(None)

zipl modes and syntax overview

When running **zipl**, you can either directly specify a base function with its parameters or a configuration file with specifications, or you can use the default **zipl** configuration file.

zipl operates in one of two modes:

Command-line mode

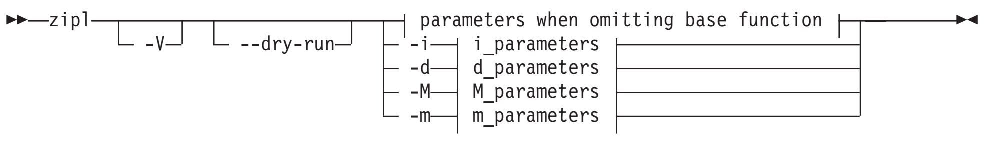
If a **zipl** command is issued with a base function other than installing a menu configuration (see "Installing a menu configuration" on page 78), the entire configuration must be defined using command-line parameters. See the following base functions for how to specify command-line parameters:

- "Preparing a boot device" on page 68
- "Preparing a DASD, SCSI, or tape dump device" on page 75
- "Preparing a multi-volume dump on ECKD DASD" on page 77

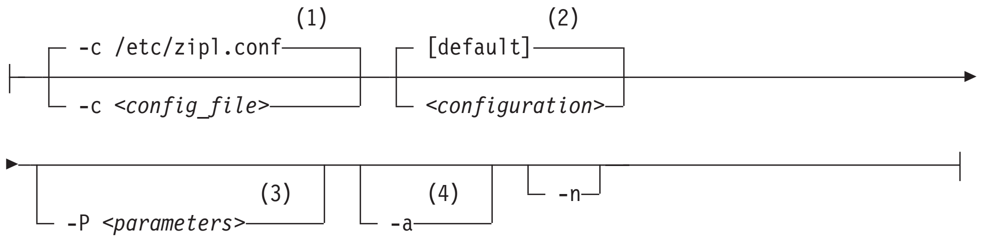
Configuration-file mode

If a **zipl** command is issued either without a base function or to install a menu configuration, a configuration file is accessed. For more information, see "Configuration file structure" on page 82.

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 in the [defaultboot] section of the configuration file (see "Configuration file structure" on page 82).

3 In a boot configuration only.

4 In 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 a boot configuration section. See "How kernel parameters from different sources are combined" on page 71 for information about how kernel parameters specified with the **-P** option are combined with any kernel parameters specified in the configuration file.

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

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

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

-V provides verbose command output.

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

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

-v displays version information.

-h displays help information.

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

See "Parameter overview" on page 79 for a summary of the short and long command line options and their configuration file equivalents.

Examples

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

```
# zipl
```

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

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

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

```
# zipl myconf
```

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

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

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

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

Preparing a boot device

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

zipl command line syntax for preparing a boot device

```
                         ┌─,0x10000──────┐
►►──zipl── -i <image>────┼───────────────┼──────────────────────────────────────────────►
                         └─,<image_addr>─┘

►──┬─ -t <directory>──┬──────────────────────────────┬─┬────────────────────────────────►
   │                  │                          (1) │ │
   │                  └─┤ Base device parameters ├───┘ │
   └─ -T <tape_node>───────────────────────────────────┘

►──┬──────────────────────────────────┬─┬──────────────────────────────────────┬────────►
   └─ -r <ramdisk>─┬─────────────────┬┘ │                ┌─,0x1000──────┐      │
                   └─,<initrd_addr>──┘  └─ -p <parmfile>─┼──────────────┼──────┘
                                                         └─,<parm_addr>─┘

►──┬───────────────────┬─┬──────────┬─┬──────┬──────────────────────────────────────────►◄
   └─ -P <parameters>──┘ └─ -k auto─┘ └─ -a──┘
```

Notes:

1 Additional parameters that are used only if **-t** specifies a logical device as a target. See "Using base device parameters" on page 73.

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

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

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

Optionally, you can also specify:

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

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

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

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

You can specify parameters *<parameters>* directly on the command line. Instead or in addition, you can specify a location *<parmfile>* of a kernel parameter file on the file system. See "How kernel parameters from different sources are combined" on page 71 for a discussion of how **zipl** combines multiple kernel parameter specifications.

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

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

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

See "Parameter overview" on page 79 for a summary of the parameters. This summary includes the long options that you can use on the command line.

Figure 17 summarizes how to specify a boot configuration within a configuration file section. Required specifications are shown in bold. See "Configuration file structure" on page 82 for more details about the configuration file.

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

Figure 17. zipl syntax for preparing a boot device - configuration file mode

Example

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

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

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

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

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

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

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

How kernel parameters from different sources are combined

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

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

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

In configuration file mode, there are three possible sources of kernel parameters. The parameters are processed in the following order:

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

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

For more information about the different sources of kernel parameters, see "Including kernel parameters in a boot configuration" on page 26.

Preparing a logical device as a boot device

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

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

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

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

You always boot from a real device. **zipl** must be able to write to that device, starting at block 0. In a logical device setup, starting at the top of the mapping hierarchy, the first block device that grants access to block 0 (and subsequent blocks) is the *base device*, see Figure 18.

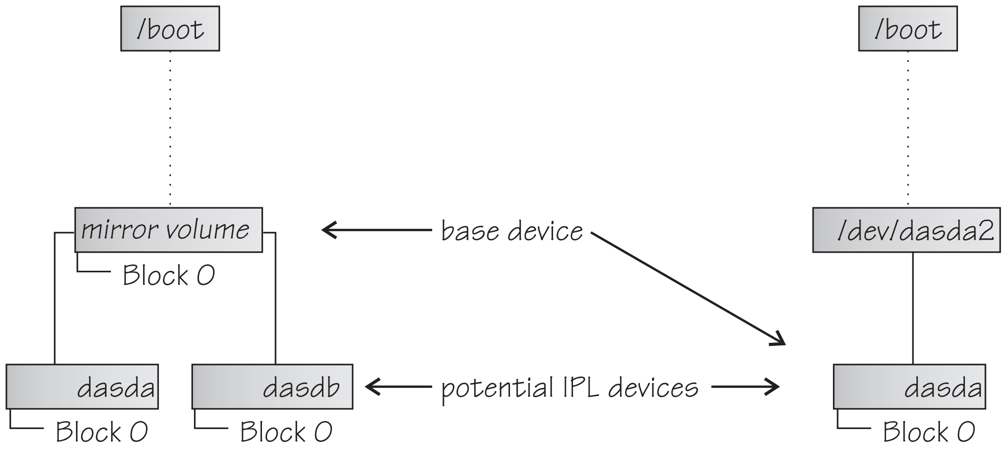


Figure 18. Definition of base device

A base device can have the following mappings:

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

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

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

If you decide not to use the supplied helper script, or want to write your own helper script, you can use parameters to supply the base device information to **zipl**, see "Using base device parameters" on page 73 and "Writing your own helper script" on page 74.

Using a helper script

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

The helper script is used automatically when you run **zipl** to prepare a boot device. Specify the parameters for the kernel image, parameter file, initial RAM disk, and target as usual. See "Preparing a boot device" on page 68 for details about the parameters.

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

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

The corresponding configuration file section becomes:

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

Using base device parameters

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

The following command syntax for the base device parameters is used for logical boot devices. It extends the **zipl** syntax as shown in “Preparing a boot device” on page 68.

Base device parameters:

```
├── --targetbase <targetbase_node>──────────────────────────────────────────►

►─ --targettype──┬──┬─ LDL─┬── --targetgeometry <cylinders>,<heads>,<sectors>─┬────►
                 │  └─ CDL─┘                                                 │
                 └──┬─ FBA─────────────────────────────────────────────────────┘
                    └─ SCSI─┘

►─ --targetblocksize <targetblocksize>── --targetoffset <targetoffset>────────┤
```

You must specify the following device information:

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

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

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

LDL	for ECKD type DASD with the Linux disk layout.
CDL	for ECKD type DASD with the compatible disk layout.
FBA	for FBA type DASD.
SCSI	for FCP-attached SCSI disks.

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

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

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

Figure 19 shows how to specify this information in a configuration file.

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

Figure 19. zipl syntax for preparing a logical device as a boot device- configuration file mode

Example

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

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

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

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

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

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

Writing your own helper script

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

- The script must accept the name of the target directory as an argument. From this specification, it must determine a suitable base device. See “Using base device parameters” on page 73.
- The script must write the following base device parameter=<*value*> pairs to stdout as ASCII text. Each pair must be written on a separate line.
  - **targetbase**=<*targetbase_node*>
  - **targettype**=<*type*> where type can be LDL, CDL, FBA, or SCSI.
  - **targetgeometry**=<*cylinders*>,<*heads*>,<*sectors*> (For LDL and CDL only)
  - **targetblocksize**=<*blocksize*>
  - **targetoffset**=<*offset*>

  See “Using base device parameters” on page 73 for the meaning of the base device parameters.
- The script must be named `zipl_helper.`<*device*> where <*device*> is the device name as specified in `/proc/devices`.
- The script must be in `/lib/s390-tools`.

Preparing a DASD, SCSI, or tape dump device

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

zipl command line syntax for preparing a DASD, SCSI, or tape dump device

```
►►──zipl── -d  <dump_device>──┬─────────┬──┬────┬──────────────────►◄
                              └─,<size>─┘  └ -n┘
```

To prepare a DASD, SCSI , or tape dump device, you must specify:

The device node <*dump_device*>
: of the DASD or SCSI 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.

  Note:
  - If the dump device is an ECKD disk with fixed-block layout (ldl), a dump overwrites the dump utility. You must reinstall the dump utility before you can use the device for another dump.
  - If the dump device is a tape, SCSI disk, 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 that is used by the system to be dumped, the resulting dump is incomplete.

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

DASD, SCSI, or tape dump devices are not formatted with a file system so no target directory can be specified. See *Using the Dump Tools*, SC33-8412 for details about processing these dumps.

See "Parameter overview" on page 79 for a summary of the parameters. The summary includes the long options that you can use on the command line.

Figure 20 summarizes how to specify a DASD, SCSI, or tape dump configuration in a configuration file. See "Configuration file structure" on page 82 for a more comprehensive discussion of the configuration file.

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

Figure 20. zipl syntax for preparing a DASD, SCSI, or tape dump device - configuration file mode

DASD example

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

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

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

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

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

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

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

SCSI example

The following command prepares a SCSI partition
| /dev/mapper/36005076303ffd40100000000000020c0-part1 as a dump device:

```
| # zipl -d /dev/mapper/36005076303ffd40100000000000020c0-part1
```

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

```
[dumpscsi]
| dumpto=/dev/mapper/36005076303ffd40100000000000020c0-part1
```

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

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

Preparing a multi-volume dump on ECKD DASD

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

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

```
►►──zipl──┬──────┬── -M <dump_device_list>──┬────────┬──┬────┬──────────────►◄
          └─ -f ─┘                          └─,<size>┘  └ -n ┘
```

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

A file -M *<dump_device_list>*
: containing the device nodes of the dump partitions, separated by one or more line feed characters (0x0a). **zipl** writes a dump signature to each involved partition and installs the stand-alone multi-volume dump tool on each involved volume. Duplicate partitions are not allowed. A maximum of 32 partitions can be listed. The volumes must be formatted with cdl. You can use any block size, even mixed block sizes. However, to speed up the dump process and to reduce wasted disk space, use block size 4096.

Optionally, you can also specify:

An option -f or --force
: to force that no signature checking takes place when dumping. Any data on all involved partitions is 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 that is used by the system to be dumped, the resulting dump is incomplete.

DASD or tape dump devices are not formatted with a file system so no target directory can be specified. See *Using the Dump Tools*, SC33-8412 for details about processing these dumps.

See “Parameter overview” on page 79 for a summary of the parameters. This summary includes the long options that you can use on the command line.

Figure 21 on page 78 summarizes how to specify a multi-volume DASD dump configuration in a configuration file. See “Configuration file structure” on page 82 for a more comprehensive discussion of the configuration file.

```
[<section_name>]
mvdump=<dump_device_list>,<size>
```

Figure 21. zipl syntax for preparing DASD devices for a multi-volume dump - configuration file mode

Example

The following command prepares two DASD partitions /dev/dasdc1, /dev/dasdd1 for a multi-volume dump and suppresses confirmation prompts that require an operator response:

```
# zipl -M sample_dump_conf -n
```

where the sample_dump_conf file contains the two partitions, separated by line breaks:

```
/dev/dasdc1
/dev/dasdd1
```

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

```
[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 must be specified with the **zipl** command that processes the configuration.

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

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

Installing a menu configuration

Use **zipl** with the **-m** (**--menu**) command-line option to install a menu configuration.

To prepare a menu configuration, you need a configuration file that includes at least one menu section (see "Menu configurations" on page 84) or with a default section that supports an automatic menu (see "Default section" on page 83).

zipl syntax for installing a menu configuration

```
                                               (1)
                                   ┌─ -c /etc/zipl.conf──┐
►►──zipl── -m <menu_name>──────────┤                     ├───┬──────┬──────────────►◄
                                   └─ -c <config_file>───┘   └─ -a──┘
```

Notes:

1 You can change the default configuration file with the ZIPLCONF environment variable.

Where:

`-m or --menu` *<menu_name>*
specifies the menu that defines the menu configuration in the configuration file.

`-c or --config` *<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`
adds the kernel image file, parmfile, and initial RAM disk image to the bootmap files in the respective target directories instead of referencing them. Use this option if the files are spread across disks to ensure that the files are available at IPL time. Specifying this option significantly increases the size of the bootmap file that is created in the target directory.

Example

Using the sample configuration file of Figure 22 on page 86, you could install a menu configuration with:

```
# zipl -m menu1
```

Parameter overview

You might need to know all **`zipl`** options and how to specify them on the command line or in the configuration file.

Command line short option **Command line long option** **Configuration file option**	**Explanation**
-a **--add-files** n/a	Causes kernel image, kernel parameter file, and initial RAM disk to be added to the bootmap file in the target directory rather than being referenced from this file. Use this option when these files are spread across multiple disks to ensure that they are available at IPL time. Specifying this option significantly increases the size of the bootmap file that is created in the target directory.

Command line short option Command line long option Configuration file option	Explanation
-c *<config_file>* **--config**=*<config_file>* n/a	Specifies the configuration file. You can change the default configuration file `/etc/zipl.conf` with the environment variable ZIPLCONF.
<configuration> n/a n/a	Specifies a configuration section to be read and processed from the configuration file.
-d *<dump_device>*[,*<size>*] **--dumpto**=*<dump_device>*[,*<size>*] **dumpto**=*<dump_device>*[,*<size>*]	Specifies the DASD partition, SCSI partition, or tape device to which a dump is to be written after IPL. The optional size specification limits the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary. If you limit the dump size below the amount of memory that is used by the system to be dumped, the resulting dump is incomplete. If no limit is provided, all of the available physical memory is dumped. For details, see “Preparing a DASD, SCSI, or tape dump device” on page 75 and *Using the Dump Tools*, SC33-8412.
-h **--help** n/a	Displays help information.
-i *<image>*[,*<image_addr>*] **--image=***<image>*[,*<image_addr>*] **image=***<image>*[,*<image_addr>*]	Specifies the location of the Linux kernel image on the file system and, optionally, in memory after IPL. The default memory address is 0x10000. See “Preparing a boot device” on page 68 for details.
-k auto **--kdump=auto** **kdump=auto**	Installs a kdump kernel that can be used as a stand-alone dump tool. You can IPL this kernel in an LPAR or guest virtual machine to create a dump of a previously running operating system instance that was configured with a reserved memory area for kdump. For Linux, this memory area is reserved with the `crashkernel=` kernel parameter. See “Preparing a boot device” on page 68 for details.
-m *<menu_name>* **--menu**=*<menu_name>* n/a	Specifies the name of the menu that defines a menu configuration in the configuration file (see “Menu configurations” on page 84).

Command line short option Command line long option Configuration file option	Explanation
-M <*dump_device_list*>[,<*size*>] **--mvdump**=<*dump_device_list*>[,<*size*>] **mvdump**=<*dump_device_list*>[,<*size*>]	Specifies a file with a list of DASD partitions to which a dump is to be written after IPL. The optional size specification limits the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary. If you limit the dump size below the amount of memory that is used by the system to be dumped, the resulting dump is incomplete. If no limit is provided, all of the available physical memory is dumped. See "Preparing a multi-volume dump on ECKD DASD" on page 77 and *Using the Dump Tools*, SC33-8412 for details.
-n **--noninteractive** n/a	Suppresses all confirmation prompts (for example, when preparing a DASD or tape dump device).
-p <*parmfile*>[,<*parm_addr*>] **--parmfile**=<*parmfile*>[,<*parm_addr*>] **parmfile**=<*parmfile*>[,<*parm_addr*>]	In a boot configuration, specifies the location of a kernel parameter file. You can specify multiple sources of kernel parameters. For more information, see "How kernel parameters from different sources are combined" on page 71. The optional <*parm_addr*> specifies the memory address where the combined kernel parameter list is to be loaded at IPL time.
-P <*parameters*> **--parameters**=<*parameters*> **parameters**=<*parameters*>	In a boot configuration, specifies kernel parameters. Individual parameters are single keywords or have the form `key=value`, without spaces. If you provide multiple parameters, separate them with a blank and enclose them within single quotation marks (') or double quotation marks ("). You can specify multiple sources of kernel parameters. For more information, see "How kernel parameters from different sources are combined" on page 71.
-r <*ramdisk*>[,<*initrd_addr*>] **--ramdisk**=<*ramdisk*>[,<*initrd_addr*> **ramdisk**=<*ramdisk*>[,<*initrd_addr*>	Specifies the location of the initial RAM disk (initrd) on the file system and, optionally, in memory after IPL. If you do not specify a memory address, **`zipl`** investigates the location of other components and calculates a suitable address for you.
-t <*directory*> **--target**=<*directory*> **target**=<*directory*>	Specifies the target directory where **`zipl`** creates boot-relevant files. The boot loader is installed on the disk that contains the target directory.
none **--targetbase**=<*targetbase_node*> **targetbase**=<*targetbase_node*>	For logical boot devices, specifies the device node of the base device, either by using the standard device name or in form of the major and minor number, separated by a colon (:). See "Using base device parameters" on page 73 for details.

Command line short option Command line long option Configuration file option	Explanation
none **--targetblocksize**=<*targetblocksize*> **targetblocksize**=<*targetblocksize*>	For logical boot devices, specifies the bytes per block of the base device. See "Using base device parameters" on page 73 for details.
none **--targetgeometry**=<*cylinders*>,<*heads*>,<*sectors*> **targetgeometry**=<*cylinders*>,<*heads*>,<*sectors*>	For logical boot devices that map to ECKD type base devices, specifies the disk geometry of the base device in cylinders, heads, and sectors. See "Using base device parameters" on page 73 for details.
none **--targetoffset**=<*targetoffset*> **targetoffset**=<*targetoffset*>	For logical boot devices, specifies the offset in blocks between the start of the physical device and the start of the logical device. See "Using base device parameters" on page 73 for details.
none **--targettype**=<*type*> **targettype**=<*type*>	For logical boot devices, specifies the device type of the base device. See "Using base device parameters" on page 73 for details.
-T <*tape_node*> **--tape**=<*tape_node*> **tape**=<*tape_node*>	Specifies the tape device where **zipl** installs the boot loader code.
-v **--version** n/a	Prints version information.
-V **--verbose** n/a	Provides more detailed command output.

If you call **zipl** in configuration file mode without specifying a configuration file, the default `/etc/zipl.conf` is used. You can change the default configuration file with the environment variable ZIPLCONF.

Configuration file structure

A configuration file comprises a default section and one or more sections with IPL configurations. In addition, there can be sections that define menu configurations.

`[defaultboot]`
a default section that defines what is to be done if the configuration file is called without a section specification.

[*<configuration>***]**
one or more sections that describe IPL configurations.

:*<menu_name>*
optionally, one or more menu sections that describe menu configurations.

A configuration file section consists of a section identifier and one or more option lines. Option lines are valid only as part of a section. Blank lines are permitted, and lines that begin with the number sign (#) are treated as comments and ignored. Option specifications consist of keyword=value pairs. There can but need not be blanks before and after the equal sign (=) of an option specification.

Default section

The default section 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.

If you specify a `target` parameter with this option, *<section_name>* is ignored and a menu with all DASD and SCSI IPL sections is build as for the `defaultauto` option.

defaultmenu=*<menu_name>*
where *<menu_name>* is the name of a menu configuration that is described in the configuration file. If the configuration file is called without a section specification, IPL devices are prepared according to this menu configuration. The `defaultmenu` option tolerates but does not require `target` parameters for the individual IPL sections.

defaultauto
If the configuration file is called without a section specification, a menu configuration is built. This configuration contains all DASD and SCSI IPL configurations in the configuration file. In the menu, these configurations appear in the order in which they appear in the configuration file.

The `defaultauto` option requires an additional option line with the `target` parameter. You can add further option lines with the `default`, `prompt`, and `timeout` parameters. These parameters have the same meaning as in "Menu configurations" on page 84.

The `defaultauto` option tolerates but does not require `target` parameters for the individual IPL sections. The resulting menu configuration is always written to the directory specified with the `target` parameter line within the default section.

As for configuration sections, extra parameters might be required for logical boot devices (see "Preparing a logical device as a boot device" on page 71).

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
```

- This default specification creates a menu with all IPL sections in the configuration file. The first IPL configuration in the automatically created menu is the default.

```
[defaultboot]
defaultauto
target=/boot
default=1
```

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 68 for details.

dumpto=<*dump_device*>
: Defines a DASD, SCSI, or tape dump configuration. For details, see "Preparing a DASD, SCSI, or tape dump device" on page 75.

mvdump=<*dump_device_list*>
: Defines a multi-volume DASD dump configuration. See "Preparing a multi-volume dump on ECKD DASD" on page 77 for details.

Additional parameters might be required for logical boot devices (see "Preparing a logical device as a boot device" on page 71).

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`*`>`
: for a DASD target device, determines whether the menu is displayed when an IPL is performed. Menus cannot be displayed for SCSI target devices.

  For prompt=1 the menu is displayed, for prompt=0 it is suppressed. If this option is omitted, the menu is not displayed. Independent of this parameter, the operator can force a menu to be displayed by specifying “prompt” in place of a configuration number for an IPL configuration to be used.

  If the menu of a menu configuration is not displayed, the operator can either specify the configuration number of an IPL configuration or the default configuration is used.

`timeout=``<`*`seconds`*`>`
: for a DASD target device and a displayed menu, specifies the time in seconds, after which the default configuration is IPLed, if no configuration has been specified by the operator. If this option is omitted or if `0` is specified as the timeout, the menu stays displayed indefinitely on the operator console and no IPL is performed until the operator specifies an IPL configuration.

As for any configuration section, additional parameters might be required for logical boot devices (see “Preparing a logical device as a boot device” on page 71).

Example

Figure 22 on page 86 shows a sample configuration file that defines multiple configuration sections and two menu configurations.

```
[defaultboot]
defaultmenu=menu1

# First boot configuration (DASD)
[boot1]
ramdisk=/boot/initrd
parameters='root=/dev/ram0 ro'
image=/boot/image-1
target=/boot

# Second boot configuration (SCSI)
[boot2]
image=/boot/mnt/image-2
ramdisk=/boot/mnt/initrd,0x900000
parmfile=/boot/mnt/parmf-2
target=/boot

# Third boot configuration (DASD)
[boot3]
image=/boot/mnt/image-3
ramdisk=/boot/mnt/initrd
parmfile=/boot/mnt/parmf-3
target=/boot

# Configuration for dumping to tape
[dumptape]
dumpto=/dev/rtibm0

# Configuration for dumping to DASD
[dumpdasd]
dumpto=/dev/dasdc1

# Configuration for multi-volume dumping to DASD
[multi_volume_dump]
mvdump=sample_dump_conf

# Configuration for dumping to SCSI disk
[dumpscsi]
dumpto=/dev/mapper/36005076303ffd40100000000000020c0-part1

# Menu containing the SCSI boot and SCSI dump configurations
:menu1
1=dumpscsi
2=boot2
target=/boot
default=2

# Menu containing two DASD boot configurations
:menu2
1=boot1
2=boot3
target=/boot
default=1
prompt=1
timeout=30
```

Figure 22. Sample /etc/zipl.conf file

The following commands assume that the configuration file of the sample is the default configuration file.

- Call **zipl** to use the default configuration file settings:

```
# zipl
```

Result: **zipl** reads the default option from the [defaultboot] section and selects the :menu1 section. It then installs a menu configuration with a boot configuration and a SCSI dump configuration.

- Call **zipl** to install a menu configuration (see also "Installing a menu configuration" on page 78):

  ```
  # zipl -m menu2
  ```

  Result: **zipl** selects the :menu2 section. It then installs a menu configuration with two DASD boot configurations. "Example for a DASD menu configuration on z/VM" on page 94 and "Example for a DASD menu configuration (LPAR)" on page 101 illustrate what this menu looks like when it is displayed.
- Call **zipl** to install a boot loader for boot configuration [boot2]:

  ```
  # zipl boot2
  ```

  Result: **zipl** selects the [boot2] section. It then installs a boot loader that loads copies of /boot/mnt/image-2, /boot/mnt/initrd, and /boot/mnt/parmf-2.
- Call **zipl** to prepare a tape that can be IPLed for a tape dump:

  ```
  # zipl dumptape
  ```

  Result: **zipl** selects the [dumptape] section and prepares a dump tape on /dev/rtibm0.
- Call **zipl** to prepare a DASD dump device:

  ```
  # zipl dumpdasd -n
  ```

  Result: **zipl** selects the [dumpdasd] section and prepares the dump device /dev/dasdc1. Confirmation prompts that require an operator response are suppressed.
- Call **zipl** to prepare a SCSI dump device:

  ```
  # zipl dumpscsi
  ```

  Result: **zipl** selects the [dumpscsi] section and prepares the dump device. The associated dump is created in the dump partition
| /dev/mapper/36005076303ffd40100000000000020c0-part1.

Chapter 6. Booting Linux

The options and requirements you have for booting Linux depend on your platform, LPAR or z/VM, and on your boot medium.

Find a general overview of how to boot Linux in an LPAR or in a z/VM guest virtual machine. For details about setting up a z/VM guest virtual machine for Linux, see *z/VM Getting Started with Linux on System z*, SC24-6194, the chapter about creating your first z/VM guest virtual machine for Linux and installing Linux.

IPL and booting

On System z, you usually start booting Linux by performing an Initial Program Load (IPL).

Figure 23 summarizes the main steps of the boot process.

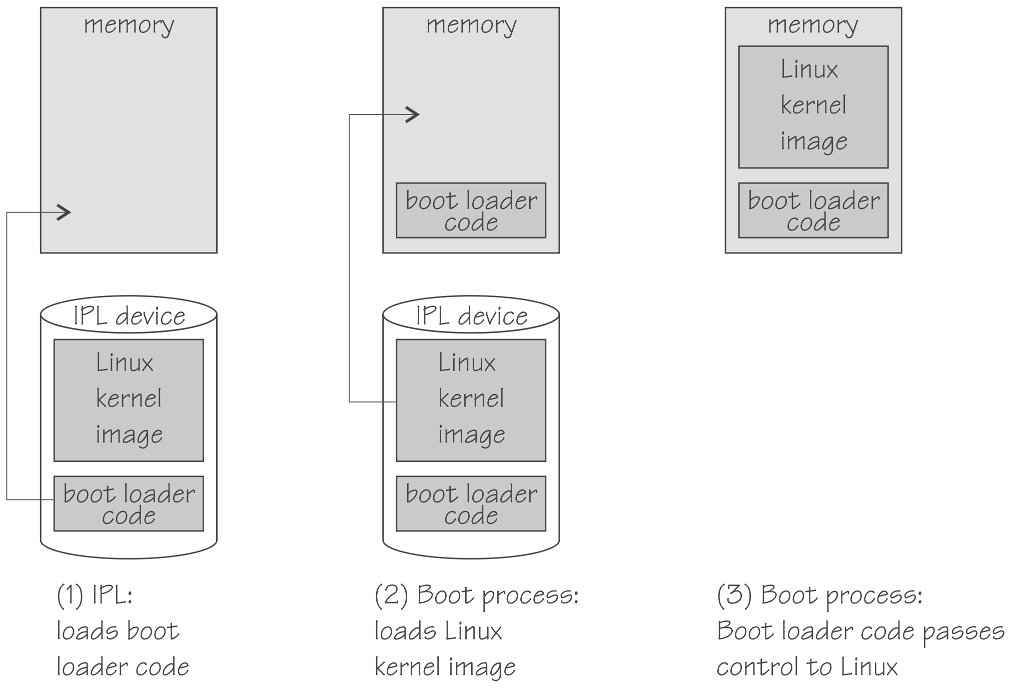


Figure 23. IPL and boot process

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 removable media or from an FTP server" on page 102).

An IPL can also start a dump process. See *Using the Dump Tools*, SC33-8412 for more information about dumps. You can find the latest version of this document on developerWorks at:

www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Use the **zipl** tool to prepare DASD, SCSI, and tape devices as IPL devices for booting Linux or for dumping.For more information about **zipl**, see Chapter 5, “Initial program loader for System z - zipl,” on page 65.

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). For Linux on z/VM, the control point is the control program (CP) of the hosting z/VM system.

The media that can be used as boot devices also depend on where Linux is to run. Table 13 provides an overview of the possibilities:

Table 13. Boot media

	DASD	tape	SCSI	NSS	z/VM reader	CD-ROM/DVD/ FTP
z/VM guest	✓	✓	✓	✓	✓	
LPAR	✓	✓	✓			✓

DASDs, tapes on channel-attached tape devices, and SCSI disks that are attached through an FCP channel can be used for both LPAR and z/VM guest virtual machines. A SCSI device can be a disk or an FC-attached CD-ROM or DVD drive. Named saved systems (NSS) and the z/VM reader are available only in a z/VM environment.

If your Linux runs in LPAR mode, you can also boot from a CD-ROM drive on the SE or HMC, or you can obtain the boot data from a remote FTP server.

Menu configurations

If you use **zipl** to prepare a DASD or SCSI boot device, you can define a menu configuration.

A boot device with a menu configuration can hold the code for multiple boot configurations. For SCSI devices, the menu can also include one or more SCSI system dumpers.

Each boot and dump configuration in a menu is associated with a configuration number. At IPL time, you can specify a configuration number to select the configuration to be used.

For menu configurations on DASD, you can display a menu with the configuration numbers (see “Example for a DASD menu configuration on z/VM” on page 94 and “Example for a DASD menu configuration (LPAR)” on page 101). For menu configurations on SCSI devices, you need to know the configuration numbers without being able to display the menus.

See “Menu configurations” on page 84 for information about defining menu configurations.

Boot data

To boot Linux, you generally need a kernel image, boot loader code, kernel parameters, and an initial RAM disk image.

For sequential I/O boot devices, z/VM reader and tape, the order in which this data is provided is significant. For random access devices, there is no required order.

Kernel image

You can use an existing kernel image or compile your own kernel.

You can obtain the kernel source from www.kernel.org. Check for additional System z specific patches on developerWorks at www.ibm.com/developerworks/linux/linux390/development_recommended.html.

If the size of the kernel image is an issue, you can compress the image when building it. You select the compression program in the kernel configuration menu at **General setup > Kernel compression mode**. The kernel configuration menu provides a help text for each available compression program. The help texts include information about the speed and efficiency of the programs. The compression program that you choose must be available on your build system.

To create a compressed kernel image run **`make bzImage`** instead of **`make image`**. No special user action is needed when booting a compressed kernel image. The image is decompressed automatically.

Important: Both compiling your own kernel or recompiling an existing distribution usually means that you must maintain your kernel yourself.

Boot loader code

A kernel image is usually compiled to contain boot loader code for a particular boot device.

For example, there are Linux configuration menu options to compile boot loader code for tape or for the z/VM reader into the kernel image.

If your kernel image does not include any boot loader code or if you want to boot a kernel image from a device that does not correspond to the included boot loader code, you can provide alternate boot loader code separate from the kernel image.

You can use **`zipl`** to prepare boot devices with separate DASD, SCSI, or tape boot loader code. You can then boot from DASD, SCSI, or tape regardless of the boot loader code in the kernel image.

Kernel parameters

The kernel parameters are in form of an ASCII text string. If the boot device is tape or the z/VM reader, the string can also be encoded in EBCDIC.

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

If you use the **zipl** command to prepare your boot device, you can provide kernel parameters on the command line, in a parameter file, and in a **zipl** configuration file.

See Chapter 3, "Kernel and module parameters," on page 25, Chapter 5, "Initial program loader for System z - zipl," on page 65, or the **zipl** and `zipl.conf` man pages for more details.

Initial RAM disk image

An initial RAM disk holds files, programs, or modules that are not included in the kernel image but are required for booting.

For example, booting from DASD requires the DASD device driver. If you want to boot from DASD but the DASD device driver has not been compiled into your kernel, you must provide the DASD device driver module on an initial RAM disk. If your image contains all files, programs, and modules that are needed for booting, you do not need an initial RAM disk.

Distributions often provide specific RAM disk images to go with their kernel images.

Booting Linux in a z/VM guest virtual machine

Boot Linux in a z/VM guest virtual machine by issuing CP commands from a CMS or CP session.

This information summarizes booting Linux in a z/VM guest virtual machine. For more information about z/VM guest environments for Linux, see *z/VM Getting Started with Linux on System z*, SC24-6194.

Booting from a tape device

Boot Linux by issuing the IPL command with a tape boot device. The boot data on the tape must be arranged in a specific order.

Before you begin

You need a tape that is prepared as a boot device. A tape boot device must contain the following items in the specified order: 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.

Procedure

Perform these steps to start the boot process:

1. Establish a CMS or CP session with the z/VM guest virtual machine where you want to boot Linux.
2. Ensure that the boot device is accessible to your z/VM guest virtual machine.
3. Ensure that the correct tape is inserted and rewound.
4. Issue a command of this form:

   ```
   #cp i <devno> parm <kernel_parameters>
   ```

   where

   <devno>
      is the device number of the boot device as seen by the guest virtual machine.

   parm *<kernel_parameters>*
      is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters that are used by your boot configuration (see “Preparing a boot device” on page 68 for information about the boot configuration).

      See also “Specifying kernel parameters when booting Linux” on page 27.

Booting from a DASD

Boot Linux by issuing the IPL command with a DASD boot device. You can specify additional parameters with the IPL command.

Before you begin

You need a DASD boot device prepared with **zipl** (see “Preparing a boot device” on page 68).

Procedure

Perform these steps to start the boot process:

1. Establish a CMS or CP session with the z/VM guest virtual machine where you want to boot Linux.
2. Ensure that the boot device is accessible to your z/VM guest virtual machine.
3. Issue a command of this form:

   ```
   #cp i <devno> 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.

When the menu is displayed, you can specify additional kernel parameters (see "Example for a DASD menu configuration on z/VM"). These additional kernel parameters are appended to the parameters you might have provided in a parameter file. The combined parameter string must not exceed 895 bytes.

See "Menu configurations" on page 84 for more details about 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 68 for information about the boot configuration).

See also "Specifying kernel parameters when booting Linux" on page 27.

Example for a DASD menu configuration on z/VM

Use the VI VMSG z/VM CP command to choose a boot configuration from a menu configuration.

This example illustrates how menu2 in the sample configuration file in Figure 22 on page 86 is displayed on the z/VM guest virtual machine console:

```
00: zIPL interactive boot menu
00:
00:  0. default (boot1)
00:
00:  1. boot1
00:  2. boot3
00:
00: Note: VM users please use '#cp vi vmsg <input>'
00:
00: Please choose (default will boot in 30 seconds): #cp vi vmsg 2
```

You choose a configuration by specifying the configuration number. For example, to boot configuration boot3 specify

```
#cp vi vmsg 2
```

You can also specify additional kernel parameters by appending them to the configuration number. For example, you can specify:

```
#cp vi vmsg 2 maxcpus=1 mem=64m
```

These parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration when booting Linux.

Booting from a SCSI device

Boot Linux by issuing the IPL command with an FCP channel as the IPL device. You must specify the target port and LUN for the boot device in advance by setting the z/VM CP LOADDEV parameter.

Before you begin

You need a SCSI boot device that is prepared with **zipl** (see "Preparing a boot device" on page 68). A SCSI device can be a disk or an FC-attached CD-ROM or DVD drive.

Procedure

Perform these steps to start the boot process:

1. Establish a CMS or CP session with the z/VM guest virtual machine where you want to boot Linux.
2. Ensure that the FCP channel that provides access to the SCSI boot disk is accessible to your z/VM guest virtual machine.
3. Specify the target port and LUN of the SCSI boot disk. Enter a command of this form:

   ```
   #cp set loaddev portname <wwpn> lun <lun>
   ```

   where:

   <wwpn>
   : specifies the world wide port name (WWPN) of the target port in hexadecimal format. A blank separates the first eight digits from the final eight digits.

   <lun>
   : specifies the LUN of the SCSI boot disk in hexadecimal format. A blank separating the first eight digits from the final eight digits.

   Example: To specify a WWPN 0x5005076300c20b8e and a LUN 0x5241000000000000:

   ```
   #cp set loaddev portname 50050763 00c20b8e lun 52410000 00000000
   ```

4. Optional for menu configurations: Specify the boot configuration (boot program in z/VM terminology) to be used. Enter a command of this form:

   ```
   #cp set loaddev bootprog <n>
   ```

   where *<n>* specifies the configuration number of the boot configuration. Omitting the bootprog parameter or specifying the value `0` selects the default configuration. For more information about menu configurations, see "Menu configurations" on page 84.

   Example: To select a configuration with configuration number `2` from a menu configuration:

   ```
   #cp set loaddev bootprog 2
   ```

5. Optional: Specify kernel parameters.

```
#cp set loaddev scpdata <APPEND|NEW> '<kernel_parameters>'
```

where:

<kernel_parameters>
: specifies a set of kernel parameters to be stored as system control program data (SCPDATA). When booting Linux, these kernel parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration.

  <kernel_parameters> must contain ASCII characters only. If characters other than ASCII characters are present, the boot process ignores the SCPDATA.

  <kernel_parameters> as entered from a CMS or CP session is interpreted as lowercase on Linux. If you require uppercase characters in the kernel parameters, run the SET LOADDEV command from a REXX script instead. In the REXX script, use the "address command" statement. See *REXX/VM Reference*, SC24-6221 and *REXX/VM User's Guide*, SC24-6222 for details.

Optional: APPEND
: appends kernel parameters to existing SCPDATA. This is the default.

Optional: NEW
: replaces existing SCPDATA.

Examples:

- To append kernel parameter `noresume` to the current SCPDATA:

  ```
  #cp set loaddev scpdata 'noresume'
  ```

- To replace the current SCPDATA with the kernel parameters `resume=/dev/sda2` and `no_console_suspend`:

  ```
  #cp set loaddev scpdata NEW 'resume=/dev/sda2 no_console_suspend'
  ```

  For a subsequent IPL command, these kernel parameters are concatenated to the end of the existing kernel parameters in your boot configuration.

6. Start the IPL and boot process by entering a command of this form:

   ```
   #cp i <devno>
   ```

   where *<devno>* is the device number of the FCP channel that provides access to the SCSI boot disk.

Tip

You can specify the target port and LUN of the SCSI boot disk, a boot configuration, and SCPDATA all with a single SET LOADDEV command. See *z/VM CP Commands and Utilities Reference*, SC24-6175 for more information about the SET LOADDEV command.

Booting from a named saved system

Boot Linux by issuing the IPL command with a named saved system (NSS). With the IPL command, you can specify kernel parameters.

Before you begin

The NSS that you use as an IPL device must contain a Linux kernel with kernel sharing support (see "Building a kernel with NSS support" on page 463 for details).

Procedure

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: NSSCSITE, NSS1234, and 73248734 are all valid NSS names.

parm <*kernel_parameters*>
: is an optional 56-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters that are used by your boot configuration (see "Preparing a boot device" on page 68 for information about the boot configuration).

  See also "Specifying kernel parameters when booting Linux" on page 27.

Booting from the z/VM reader

Boot Linux by issuing the IPL command with the z/VM reader as the IPL device. You first must transfer the boot data to the reader.

Before you begin

You need the following files, all in record format `fixed 80`:
- Linux kernel image
- Kernel parameters (optional)
- Initial RAM disk image (optional)

About this task

This information is a summary of how to boot Linux from a z/VM reader. For more details, see the Redpaper *Building Linux Systems under IBM VM*, REDP-0120.

Procedure

Proceed like this to boot Linux from a z/VM reader:
1. Establish a CMS session with the guest where you want to boot Linux.
2. Transfer the kernel image, kernel parameters, and the initial RAM disk image to your guest. You can obtain the files from a shared minidisk or use:
   - The z/VM sendfile facility.
   - An FTP file transfer in binary mode.

Files that are sent to your reader contain a file header that you must remove before you can use them for booting. Receive files that you obtain through your z/VM reader to a minidisk.

3. Set up the reader as a boot device.
   a. Ensure that your reader is empty.
   b. Direct the output of the punch device to the reader. Issue:

      ```
      spool pun * rdr
      ```

   c. Use the CMS PUNCH command to transfer each of the required files to the reader. Be sure to use the "no header" option to omit the file headers.
      First transfer the kernel image.
      Second transfer the kernel parameters.
      Third transfer the initial RAM disk image, if present.

      For each file, issue a command of this form:

      ```
      pun <file_name> <file_type> <file_mode> (noh
      ```

   d. Optional: Ensure that the contents of the reader remain fixed.

      ```
      change rdr all keep nohold
      ```

      If you omit this step, all files are deleted from the reader during the IPL that follows.
4. Issue the IPL command:

   ```
   ipl 000c clear parm <kernel_parameters>
   ```

   where:

   0x000c
   is the device number of the reader.

   parm *<kernel_parameters>*
   is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters that are used by your boot configuration (see "Preparing a boot device" on page 68 for information about the boot configuration).

   See also "Specifying kernel parameters when booting Linux" on page 27.

Booting Linux in LPAR mode

You can boot Linux in LPAR mode from a Hardware Management Console (HMC) or Support Element (SE).

About this task

The following description refers to an HMC, but the same steps also apply to an SE.

Booting from DASD, tape, or SCSI

The initial steps for Linux in LPAR mode are the same for DASD, tape, or SCSI.

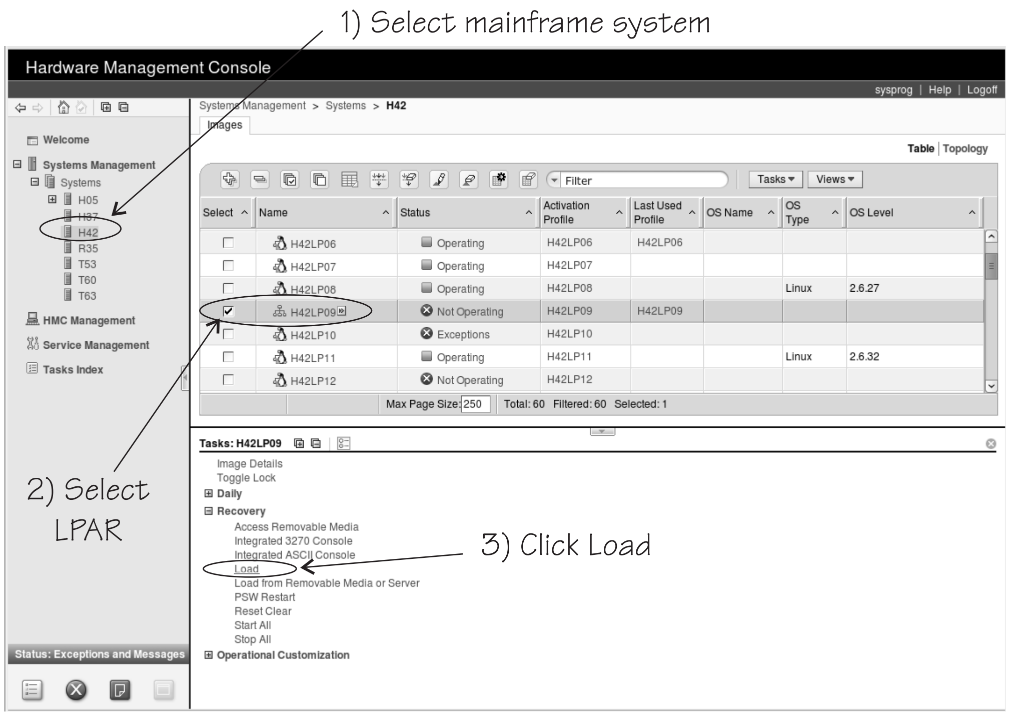
Before you begin

- You need a boot device that is prepared with **`zipl`** (see “Preparing a boot device” on page 68).
- For booting from a SCSI boot device, you must have the SCSI IPL feature (FC9904) installed.

Procedure

Perform these steps to boot from a DASD, tape, or SCSI boot device:

1. In the navigation pane of the HMC, expand **Systems Management** and **Servers** and select the mainframe system that you want to work with. A table of LPARs is displayed on the **Images** tab in the content area.
2. Select the LPAR where you want to boot Linux.
3. In the **Tasks** area, expand **Recovery** and click **Load** (see Figure 24).

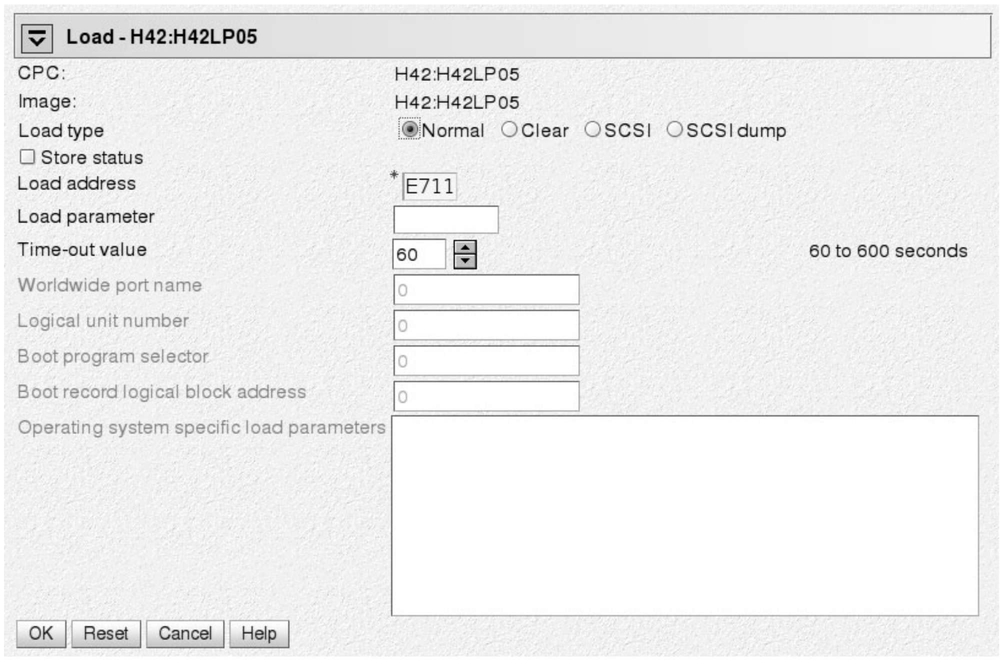


Select	Name	Status	Activation Profile	Last Used Profile	OS Name	OS Type	OS Level
☐	H42LP06	Operating	H42LP06	H42LP06			
☐	H42LP07	Operating	H42LP07				
☐	H42LP08	Operating	H42LP08			Linux	2.6.27
✓	H42LP09	Not Operating	H42LP09	H42LP09			
☐	H42LP10	Exceptions	H42LP10				
☐	H42LP11	Operating	H42LP11			Linux	2.6.32
☐	H42LP12	Not Operating	H42LP12				

Max Page Size: 250 Total: 60 Filtered: 60 Selected: 1

Figure 24. Load task on the HMC

4. Proceed according to your boot device.
   - For booting from tape:
     a. Select the load type **Normal** (see Figure 25 on page 100).


Load - H42:H42LP05

CPC: H42:H42LP05

Image: H42:H42LP05

Load type [✓ Normal] Clear SCSI SCSI dump

☐ Store status

Load address * E711

Load parameter

Time-out value 60 — 60 to 600 seconds

Worldwide port name 0

Logical unit number 0

Boot program selector 0

Boot record logical block address 0

Operating system specific load parameters

Figure 25. Load panel for booting from tape or DASD

b. Enter the device number of the tape boot device in the **Load address** field.

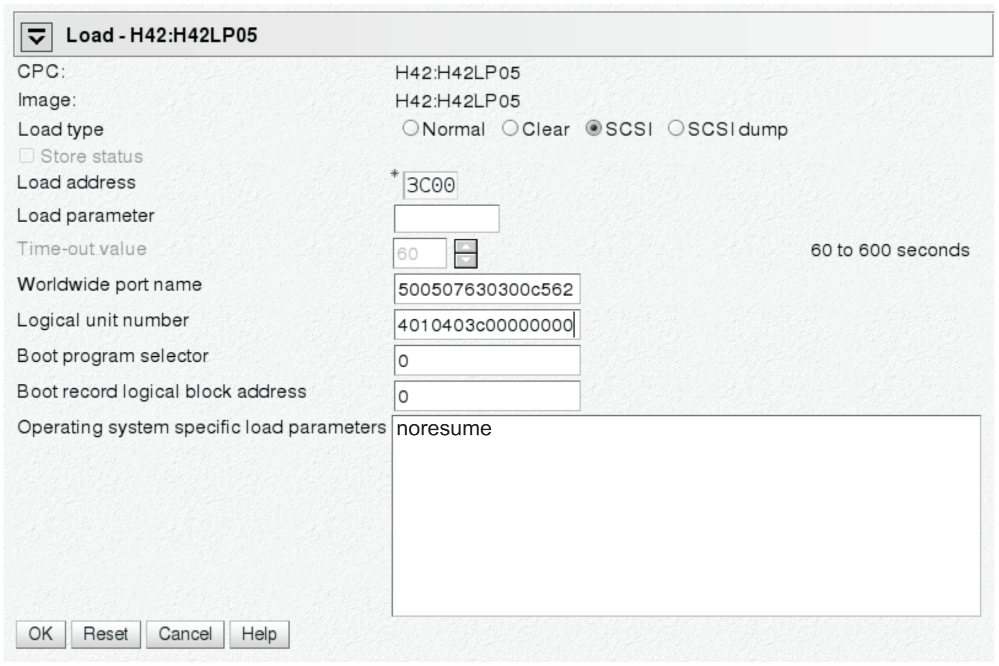
- For booting from DASD
  a. Select load type **Normal** (see Figure 25).
  b. Enter the device number of the DASD boot device in the **Load address** field.
  c. If the boot configuration is part of a **`zipl`** created menu configuration, type the configuration number that identifies your DASD boot configuration within the menu in the **Load parameter** field.

     Configuration number `0` specifies the default configuration. Depending on the menu configuration, omitting this option might display the menu or select the default configuration. Specifying “prompt” instead of a configuration number forces the menu to be displayed.

     When the menu is displayed, you can specify additional kernel parameters (see “Example for a DASD menu configuration (LPAR)” on page 101). 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 84 for more details about 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 26 on page 101).



CPC:	H42:H42LP05
Image:	H42:H42LP05
Load type	○ Normal ○ Clear ● SCSI ○ SCSI dump
☐ Store status	
Load address	* 3C00
Load parameter	
Time-out value	60 (60 to 600 seconds)
Worldwide port name	500507630300c562
Logical unit number	4010403c00000000
Boot program selector	0
Boot record logical block address	0
Operating system specific load parameters	noresume

Figure 26. Load panel with SCSI feature enabled — for booting from a SCSI disk

b. Enter the device number of the FCP channel through which the SCSI disk is accessed in the **Load address** field.

c. Enter the WWPN of the SCSI disk in the **World wide port name** field.

d. Enter the LUN of the SCSI disk in the **Logical unit number** field.

e. If the boot configuration is part of a **`zipl`** created menu configuration, type the configuration number that identifies your SCSI boot configuration within the menu in the **Boot program selector** field. Configuration number `0` specifies the default configuration.

   See “Menu configurations” on page 84 for more details about menu configurations.

f. Optional: Type kernel parameters in the **Operating system specific load parameters** field. These parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration when booting Linux.

   Use ASCII characters only. If you enter characters other than ASCII characters, the boot process ignores the data in the **Operating system specific load parameters** field.

g. Accept the defaults for the remaining fields.

5. Click **OK** to start the boot process.

What to do next

Check the output on the preferred console (see “Console kernel parameter syntax” on page 43) to monitor the boot progress.

Example for a DASD menu configuration (LPAR)

Use the Operating System Messages applet on the HMC or SE to choose a boot configuration from a menu configuration.

This example illustrates how menu2 in the sample configuration file in Figure 22 on page 86 is displayed on the HMC or SE:

```
zIPL interactive boot menu

0. default (boot1)

1. boot1
2. boot3

Please choose (default will boot in 30 seconds): 2
```

You choose a configuration by specifying the configuration number. For example, to boot configuration boot3 specify 2.

You can also specify additional kernel parameters by appending them to the configuration number. For example, you can specify:

```
2 maxcpus=1 mem=64m
```

These parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration when booting Linux.

Loading Linux from removable media or from an FTP server

You can use the SE to copy the Linux kernel image directly to your LPAR memory. This process bypasses IPL and does not require a boot loader.

Before you begin

You need installation data that includes a special file with installation information (with extension "ins"). This file can be located:

- On a disk that is inserted in the CD-ROM or DVD drive of the SE or of the system where the HMC runs
- In the file system of an FTP server to which you have access

The "ins-file" contains a mapping of the location of installation data on the disk or FTP server and the memory locations where the data is to be copied.

About this task

The SE performs the tasks that are normally done by the boot loader code. After the Linux kernel is loaded, Linux is started using restart PSW.

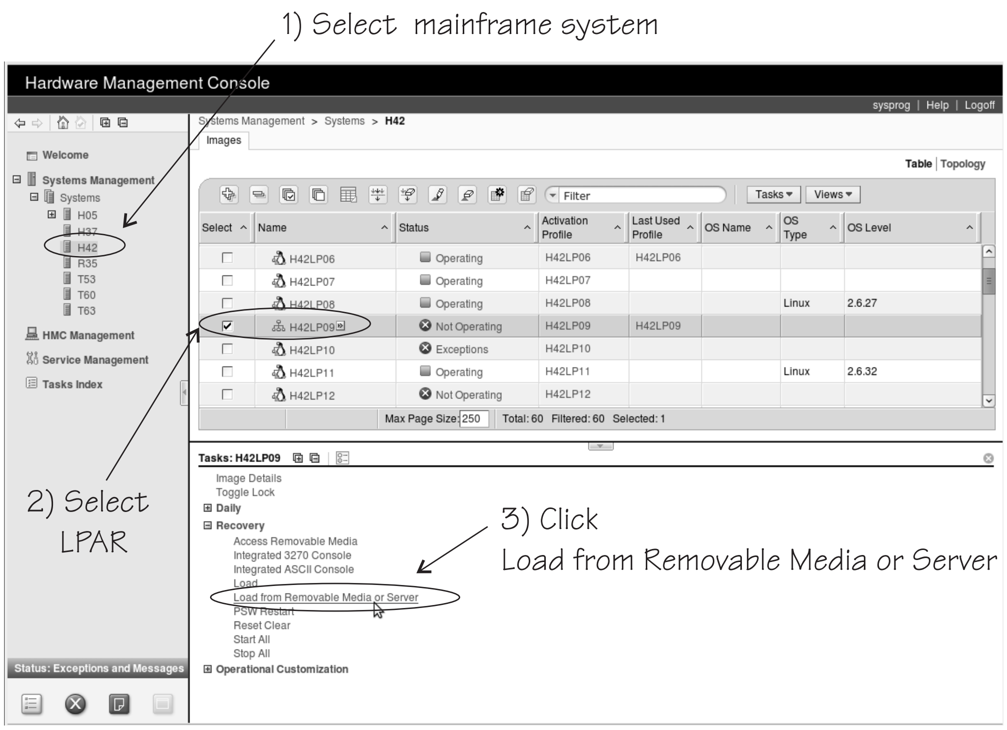
As a source, you can use the CD-ROM or DVD drive of the SE or any device on a remote system that you can access through FTP from your SE. If you access the SE remotely from an HMC, you can also use the CD-ROM or DVD drive of the system where your HMC runs.

Procedure

Perform these steps:

1. In the navigation pane of the HMC, expand **Systems Management** and **Servers** and select the mainframe system that you want to work with. A table of LPARs is displayed on the **Images** tab in the content area.
2. Select the LPAR where you want to boot Linux.

3. In the **Tasks** area, expand **Recovery** and click **Load from Removable Media or Server** (see Figure 27).

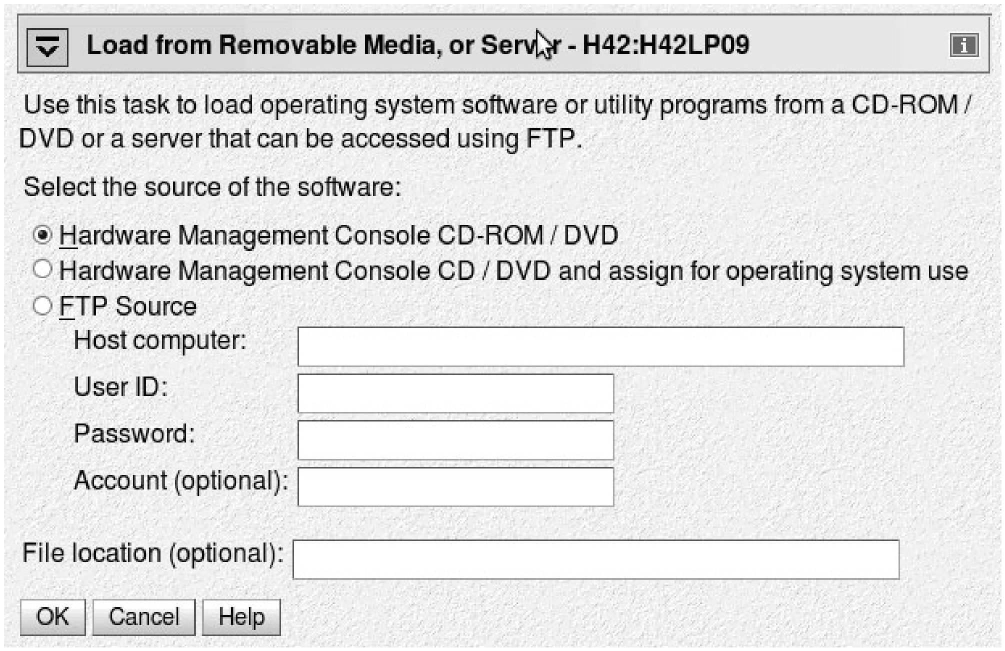


Select	Name	Status	Activation Profile	Last Used Profile	OS Name	OS Type	OS Level
☐	H42LP06	Operating	H42LP06	H42LP06			
☐	H42LP07	Operating	H42LP07				
☐	H42LP08	Operating	H42LP08			Linux	2.6.27
☑	H42LP09	Not Operating	H42LP09	H42LP09			
☐	H42LP10	Exceptions	H42LP10				
☐	H42LP11	Operating	H42LP11			Linux	2.6.32
☐	H42LP12	Not Operating	H42LP12				

Max Page Size: 250 Total: 60 Filtered: 60 Selected: 1

Figure 27. Load from Removable Media or Server task on the HMC

4. Specify the source of the code to be loaded.
   - For loading from a CD-ROM or DVD drive
     a. Select **Hardware Management Console CD-ROM/DVD** (see Figure 28 on page 104).


Load from Removable Media, or Server - H42:H42LP09

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]
- Hardware Management Console CD / DVD and assign for operating system use
- FTP Source

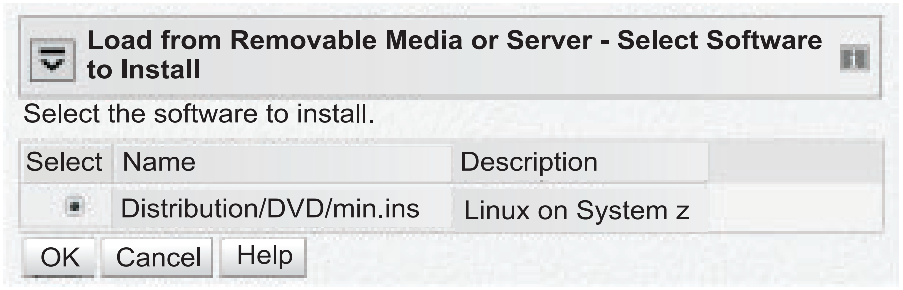
Host computer:	
User ID:	
Password:	
Account (optional):	
File location (optional):	

Figure 28. Load from Removable Media or Server panel

b. Enter the path for the directory where the “ins-file” is in the **File location** field. You can leave this field blank if the “ins-file” is in the root directory of the file system on the CD-ROM or DVD.

- For loading from an FTP server
  a. Select **FTP Source**.
  b. Enter the IP address or host name of the FTP server with the installation code 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, type your account information in the **Account** entry field.
  f. Enter the path for the directory where the “ins-file” resides in the file location entry field. You can leave this field blank if the file is in the FTP server's root directory.

5. Click **Continue** to display the Select Software to Install panel (Figure 29).


Load from Removable Media or Server - Select Software to Install

Select the software to install.

Select	Name	Description
●	Distribution/DVD/min.ins	Linux on System z

Figure 29. Select Software to Install panel

6. Select the “ins-file” to be used.
7. Click **OK** to start loading Linux.

Results

Distribution-specific configuration scripts take over, if present.

Displaying current IPL parameters

To display the IPL parameters, use the **lsreipl** command with the **-i** option. Alternatively, a sysfs interface is available.

For more information about the **lsreipl** command, see "lsreipl - List IPL and re-IPL settings" on page 629. In sysfs, information about IPL parameters is available in subdirectories of /sys/firmware/ipl.

/sys/firmware/ipl/ipl_type

The /sys/firmware/ipl/ipl_type file contains the device type from which the kernel was booted. The following values are possible:

ccw
The IPL device is a CCW device, for example, a DASD or the z/VM reader.

fcp
The IPL device is an FCP device.

nss
The IPL device is a z/VM named saved system.

unknown
The IPL device is not known.

Depending on the IPL type, there might be more files in /sys/firmware/ipl/.

If the device is a CCW device, the additional files device and loadparm are present.

device
Contains the bus ID of the CCW device that is used for IPL, for example:

```
# cat /sys/firmware/ipl/device
0.0.1234
```

loadparm
Contains up to 8 characters for the loadparm that is used for IPL, for example:

```
# cat /sys/firmware/ipl/loadparm
1
```

parm

Contains additional kernel parameters that are specified with the PARM parameter when booting with the z/VM CP IPL command, for example:

```
# cat /sys/firmware/ipl/parm
noresume
```

See also "Specifying kernel parameters when booting Linux" on page 27.

A leading equal sign (=) indicates that the existing kernel parameters used by the boot configuration were ignored and the kernel parameters of the parm attribute where the only kernel parameters used for booting Linux. See "Replacing all kernel parameters in a boot configuration" on page 29.

If the device is FCP, a number of additional files are present (also see Chapter 11, "SCSI-over-Fibre Channel device driver," on page 185 for details):

device Contains the bus ID of the FCP device that is used for IPL, for example:

```
# cat /sys/firmware/ipl/device
0.0.50dc
```

wwpn Contains the WWPN used for IPL, for example:

```
# cat /sys/firmware/ipl/wwpn
0x5005076300c20b8e
```

lun Contains the LUN used for IPL, for example:

```
# cat /sys/firmware/ipl/lun
0x5010000000000000
```

br_lba Contains the logical block address of the boot record on the boot device (usually 0).

bootprog
Contains the boot program number.

scp_data
Contains additional kernel parameters that are used when booting from a SCSI device, for example:

```
# cat /sys/firmware/ipl/scp_data
noresume
```

See "Booting from a SCSI device" on page 95 and "Booting from DASD, tape, or SCSI" on page 98).

A leading equal sign (=) indicates that the existing kernel parameters used by the boot configuration were ignored and the kernel parameters of the `scp_data` attribute where the only kernel parameters used for booting Linux. See "Replacing all kernel parameters in a boot configuration" on page 29.

binary_parameter
Contains the information of the preceding files in binary format.

Rebooting from an alternative source

When you reboot Linux, the system conventionally boots from the last used location. However, you can configure an alternative device to be used for re-IPL instead of the last used IPL device.

Before you start:

- Linux must be compiled to support rebooting from an alternative source. This feature is built into the kernel by default.
- The System z hardware must have zfcp IPL support for re-IPL from SCSI devices.

Use the **chreipl** tool to configure the re-IPL device. For more information about the **chreipl** tool, see "chreipl - Modify the re-IPL configuration" on page 550.

Alternatively, you can use a sysfs interface. In sysfs, the virtual configuration files are located under `/sys/firmware/reipl`. To configure, write strings into the configuration files. The following re-IPL types can be set with the `/sys/firmware/reipl/reipl_type` attribute:

`ccw` For ccw devices such as DASDs that are attached through ESCON or FICON.

`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

You can find the attributes for re-IPL type ccw in the `/sys/firmware/reipl/ccw` sysfs directory.

`device` Device number of the re-IPL device. For example, 0.0.7412.

Note: IPL is possible only from subchannel set 0.

`loadparm`
Up to eight characters for the loadparm used to select the boot configuration in the zipl menu (if available).

`parm` A 64-byte string of kernel parameters that is concatenated to the boot command-line. The PARM parameter can be set only for Linux on z/VM. See also “Specifying kernel parameters when booting Linux” on page 27.

A leading equal sign (=) means that the existing kernel parameter line in the boot configuration is ignored and the boot process uses the kernel parameters in the `parm` attribute only. See also “Replacing all kernel parameters in a boot configuration” on page 29.

Attributes for fcp

You can find the attributes for re-IPL type fcp in the `/sys/firmware/reipl/fcp` sysfs directory.

`device` Device number of the FCP device that is used for re-IPL. For example, 0.0.7412.

Note: IPL is possible only from subchannel set 0.

`wwpn` World wide port number of the FCP re-IPL device.

`lun` Logical unit number of the FCP re-IPL device.

`bootprog`
Boot program selector. Used to select the boot configuration in the zipl menu (if available).

`br_lba` Boot record logical block address. Master boot record. Is always 0 for Linux.

scp_data
Kernel parameters to be used for the next FCP re-IPL.

A leading equal sign (=) means that the existing kernel parameter line in the boot configuration is ignored and the boot process uses the kernel parameters in the `scp_data` attribute only. See also "Replacing all kernel parameters in a boot configuration" on page 29.

Attributes for nss

You can find the attributes for re-IPL type nss in the `/sys/firmware/reipl/nss` sysfs directory.

name Name of the NSS. The NSS name can be one to eight characters long and must consist of alphabetic or numeric characters. The following examples are all valid NSS names: 73248734, NSSCSITE, or NSS1234.

parm If the NSS contains a Linux instance, a 56-byte string of kernel parameters that is concatenated to the kernel parameters in the boot configuration. (Note the difference in length compared to ccw.) See also "Specifying kernel parameters when booting Linux" on page 27.

A leading equal sign (=) means that the existing kernel parameter line in the boot configuration is ignored and the boot process uses the kernel parameters in the `parm` attribute only. See also "Replacing all kernel parameters in a boot configuration" on page 29.

Kernel panic settings

Set the attribute `/sys/firmware/shutdown_actions/on_panic` to `reipl` to make the system re-IPL with the current re-IPL settings if a kernel panic occurs.

See also the description of the **dumpconf** tool in *Using the Dump Tools*, SC33-8412 on the developerWorks website at
www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Examples for configuring re-IPL

Typical examples include configuring re-IPL from an FCP device and specifying parameters for re-IPL.

- To configure an FCP re-IPL device 0.0.5711 with a LUN 0x1711000000000000 and a WWPN 0x5005076303004715 with an additional kernel parameter `noresume`:

```
# echo 0.0.5711 > /sys/firmware/reipl/fcp/device
# echo 0x5005076303004715 > /sys/firmware/reipl/fcp/wwpn
# echo 0x1711000000000000 > /sys/firmware/reipl/fcp/lun
# echo 0 > /sys/firmware/reipl/fcp/bootprog
# echo 0 > /sys/firmware/reipl/fcp/br_lba
# echo "noresume" > /sys/firmware/reipl/fcp/scp_data
# echo fcp > /sys/firmware/reipl/reipl_type
```

  Note: IPL is possible only from subchannel set 0.

- To set up re-IPL from a Linux NSS with different parameters:
  1. Change to the reipl sysfs directory:

```
# cd /sys/firmware/reipl/
```

  2. Set the reipl_type to nss:

```
# echo nss > reipl_type
```

3. Set up the attributes in the nss directory:

   ```
   # echo LNXNSS > name
   # echo "dasd=0150 root=/dev/dasda1" > parm
   ```

   Assuming that dasd= and root= are already included in your nss boot configuration and that no other kernel parameters are required, you can change to a root file system on a different device by replacing the existing kernel parameters.

   ```
   # echo LNXNSS > name
   # echo "=dasd=0150 root=/dev/dasda1" > parm
   ```

- To specify additional kernel parameters for Linux re-IPL, follow these steps:
  1. Change to the sysfs directory appropriate for the next re-IPL:

     ```
     # cd /sys/firmware/reipl/$(cat /sys/firmware/reipl/reipl_type)
     /sys/firmware/reipl/ccw
     ```

  2. Use the echo command to output the parameter string into the `parm` attribute:

     ```
     # echo "noresume" > parm
     ```

Chapter 7. Suspending and resuming Linux

With suspend and resume support, you can stop a running Linux on System z instance and later continue operations.

When Linux is suspended, data is written to a swap partition. The resume process uses this data to make Linux continue from where it left off when it was suspended. A suspended Linux instance does not require memory or processor cycles.

Linux on System z suspend and resume support applies to both Linux on z/VM and Linux instances that run directly in an LPAR.

While a Linux instance is suspended, you can run another Linux instance in the z/VM guest virtual machine or in the LPAR where the suspended Linux instance was running.

What you should know about suspend and resume

Before suspending a Linux instance, you must be aware of the prerequisites and of activities that can cause resume to fail.

Prerequisites for suspending a Linux instance

Suspend and resume support checks for conditions that might prevent resuming a suspended Linux instance. You cannot suspend a Linux instance unless all prerequisites that are fulfilled.

The following prerequisites must be fulfilled regardless of whether a Linux instance runs directly in an LPAR or as a z/VM guest:

- All tape device nodes must be closed and online tape drives must be unloaded.
- There must be no configured Common Link Access to Workstation (CLAW) devices.

  The CLAW device driver does not support suspend and resume. You must ungroup all CLAW devices before you can suspend a Linux instance.
- The Linux instance must not have used any hotplug memory since it was last booted.
- No program must be in a prolonged uninterruptible sleep state.

  Programs can assume this state while they are waiting for an outstanding I/O request to complete. Most I/O requests complete in a very short time and do not compromise suspend processing. An example of an I/O request that can take too long to complete is rewinding a tape.

For Linux on z/VM, the following additional prerequisites must be fulfilled:

- No discontiguous saved segment (DCSS) device must be accessed in exclusive-writable mode.

  You must remove all DCSSs of segment types EW, SW, and EN by writing the DCSS name to the sysfs `remove` attribute.

  You must remove all DCSSs of segment types SR and ER that are accessed in exclusive-writable mode or change their access mode to `shared`.

For more information, see “Removing a DCSS device” on page 459 and “Setting the access mode” on page 456.

- All device nodes of the z/VM recording device driver must be closed.
- All device nodes of the z/VM unit record device driver must be closed.
- No watchdog timer must run and the watchdog device node must be closed.

Precautions while a Linux instance is suspended

There are conditions outside the control of the suspended Linux instance that can cause resume to fail.

- The CPU configuration must remain unchanged between suspend and resume.
- The data that is written to the swap partition when the Linux instance is suspended must not be compromised.

  In particular, be sure that the swap partition is not used if another operating system instance runs in the LPAR or z/VM guest virtual machine while the initial Linux instance is suspended.
- If the Linux instance uses expanded storage (XPRAM), this expanded storage must remain unchanged until the Linux instance is resumed.

  If the size or content of the expanded memory is changed before the Linux instance is resumed or if the expanded memory is unavailable when the Linux instance is resumed, resuming fails with a kernel panic.
- If an instance of Linux on z/VM uses one or more DCSSs, these DCSSs must remain unchanged until the Linux instance is resumed.

  If the size, location, or content of a DCSS is changed before the Linux instance is resumed, resuming fails with a kernel panic.
- For an instance of Linux on z/VM with a Linux kernel that is a named saved system (NSS), the NSS must remain unchanged until the Linux instance is resumed.

  If the size, location, or content of the NSS is changed before the Linux instance is resumed, resuming fails.
- Take special care when replacing a DASD and, thus, making a different device available at a particular device bus-ID.

  You might intentionally replace a device with a backup device. Changing the device also changes its UID-based device nodes. Expect problems if you run an application that depends on UID-based device nodes and you exchange one of the DASD the application uses. In particular, you cannot use multipath tools when the UID changes.
- The SCSI configuration must remain unchanged until the Linux instance is resumed.
- Generally, avoid changes to the real or virtual hardware configuration between suspending and resuming a Linux instance.
- Disks that hold swap partitions or the root file system must be present when resuming the Linux instance.

Handling of devices that are unavailable when resuming

Devices that were available when the Linux instance was suspended might be unavailable when resuming.

If such unavailable devices were offline when the Linux instance was suspended, they are de-registered and the device name can be assigned to other devices.

If unavailable devices where online when the Linux instance was suspended, handling depends on the respective device driver. DASD and FCP devices remain registered as disconnected devices. The device name and the device configuration are preserved. Devices that are controlled by other device drivers are de-registered.

Handling of devices that become available at a different subchannel

The mapping between subchannels and device bus-IDs can change if the real or virtual hardware is restarted between suspending and resuming Linux.

If the subchannel changes for a DASD or FCP device, the device configuration is changed to reflect the new subchannel. This change is accomplished without de-registration. Thus, device name and device configuration are preserved.

If the subchannel changes for any other device, the device is de-registered and registered again as a new device.

Building a kernel with suspend and resume support

To obtain a kernel with suspend and resume support, you require specific common code options in the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

You need a kernel with the following common code options compiled into the kernel:
- CONFIG_PM
- CONFIG_HIBERNATION

Selecting these items automatically selects all other options that are required.

Setting up Linux for suspend and resume

Configure suspend and resume support through kernel parameters and set up a suitable swap partition for suspending and resuming a Linux instance.

Kernel parameters

You configure the suspend and resume support by adding parameters to the kernel parameter line.

suspend and resume kernel parameter syntax

```
►►──resume=<device_node>──┬────────────────────┬──┬──────────┬──────────►◄
                          └─no_console_suspend─┘  └─noresume─┘
```

where:

resume=*<device_node>*
specifies the standard device node of the swap partition with the data that is required for resuming the Linux instance.

no_console_suspend
prevents Linux consoles from being suspended early in the suspend process. Without this parameter, you cannot see the kernel messages that are issued by the suspend process.

noresume
boots the kernel without resuming a previously suspended Linux instance. Add this parameter to circumvent the resume process, for example, if the data written by the previous suspend process is damaged.

Example

To use a partition /dev/dasda2 as the swap partition and prevent Linux consoles from being suspended early in the suspend process specify:

```
resume=/dev/dasda2 no_console_suspend
```

Setting up a swap partition

During the suspend process, Linux writes data to a swap partition. This data is required later to resume Linux.

Set up a swap partition that is at least the size of the available LPAR memory or the memory of the z/VM guest virtual machine.

Do not use this swap partition for any other operating system that might run in the LPAR or z/VM guest virtual machine while the Linux instance is suspended.

You cannot suspend a Linux instance while most of the memory and most of the swap space is in use. If there is not sufficient remaining swap space to hold the data for resuming the Linux instance, suspending the Linux instance fails. To assure sufficient swap space you might have to configure two swap partitions, one partition for regular swapping and another for suspending the Linux instance. Configure the swap partition for suspending the Linux instance with a lower priority than the regular swap partition.

Use the pri= parameter to specify the swap partitions in /etc/fstab with different priorities. See the swapon man page for details.

The following example shows two swap partitions with different priorities:

```
# cat /etc/fstab
...
/dev/dasdb1 swap swap pri=-1 0 0
/dev/dasdc1 swap swap pri=-2 0 0
```

In the example, the partition to be used for the resume data is /dev/dasdc1.

You can check your current swap configuration by reading /proc/swaps.

```
# cat /proc/swaps
Filename          Type        Size       Used     Priority
/dev/dasdb1       partition   7212136    71056    -1
/dev/dasdc1       partition   7212136    0        -2
```

Configuring for fast resume

The more devices are available to a Linux instance, the longer it takes to resume a suspended instance.

With a thousand or more available devices, the resume process can take longer than an IPL. If the duration of the resume process is critical for a Linux instance with many devices, include unused devices in the exclusion list (see “cio_ignore - List devices to be ignored” on page 704 and “cio_ignore - Manage the I/O exclusion list” on page 558).

Suspending a Linux instance

Suspend a Linux instance by writing to the `/sys/power/state` sysfs attribute.

Before you begin

Attention: Suspend only Linux instances for which you specified the `resume=` kernel parameter. Without this parameter, you cannot resume the suspended Linux instance.

Procedure

Enter the following command to suspend a Linux instance:

```
# echo disk > /sys/power/state
```

Results

On the Linux console you might see progress messages until the console itself is suspended. Most of these messages require log level 7 or higher to be printed. See “Using the magic sysrequest feature” on page 58 about setting the log level. You cannot see the progress messages if you suspend the Linux instance from an ssh session.

Resuming a suspended Linux instance

Boot Linux to resume a suspended Linux instance.

About this task

Use the same kernel, initial RAM disk, and kernel parameters that you used to first boot the suspended Linux instance.

You must reestablish any terminal session for HVC terminal devices and for terminals that are provided by the iucvtty program. You also must reestablish all ssh sessions that timed out while the Linux instance was suspended.

If resuming the Linux instance fails, boot Linux again with the `noresume` kernel parameter. The boot process then ignores the data that was written to the swap partition and starts Linux without resuming the suspended instance.

Configuring Linux to suspend on SIGNAL SHUTDOWN

You can configure Linux on z/VM to suspend when receiving the z/VM CP SIGNAL SHUTDOWN command.

About this task

A z/VM administrator can use a CP command, SIGNAL SHUTDOWN, to shut down (log off) a z/VM guest virtual machine. Typically, the z/VM configuration defines a shutdown interval between the command and the logoff.

By default, a Linux instance performs a regular shutdown during the shutdown interval. After the z/VM guest virtual machine is logged on again, Linux must be rebooted with an IPL.

You can configure Linux to suspend to disk instead of shutting down. A subsequent IPL then leads to Linux resuming rather than booting.

How to configure suspend on SIGNAL SHUTDOWN depends on whether your Linux distribution uses Upstart or inittab.

Using inittab

For distributions that use inittab, edit `/etc/inittab`.

Replace the line

```
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

with

```
ca::ctrlaltdel:/bin/sh -c "/bin/echo disk > /sys/power/state || /sbin/shutdown -t3 -h now"
```

If suspending Linux fails, a shutdown is performed as a backup action.

Using Upstart

For distributions that use Upstart, edit `/etc/event.d/control-alt-delete`.

Replace the line

```
exec /sbin/shutdown -t3 -r now "Control-Alt-Delete pressed"
```

with

```
exec /bin/sh -c "/bin/echo disk > /sys/power/state || /sbin/shutdown -t3 -h now"
```

for each Linux guest.

If suspending Linux fails, a shutdown is performed as a backup action.

Chapter 8. Shutdown actions

Several triggers can cause Linux to shut down. For each shutdown trigger, you can configure a specific shutdown action to be taken as a response.

Table 14. Shutdown triggers and default action overview

Trigger	Command or condition	Default shutdown action
halt	Linux **shutdown -H** command	stop
poff	Linux **poweroff** or **shutdown -P** command	stop
reboot	Linux **reboot** or **shutdown -r** command	reipl
restart	• **PSW restart** on the HMC for Linux in LPAR mode • z/VM CP **system restart** command for Linux on z/VM	stop
panic	Linux kernel panic	stop

The available shutdown actions are summarized in Table 15.

Table 15. Shutdown actions

Action	Explanation	See also
stop	For `panic` or `restart`, enters a disabled wait state. For all other shutdown triggers, stops all CPUs.	n/a
ipl	Performs an IPL according to the specifications in `/sys/firmware/ipl`.	“Displaying current IPL parameters” on page 105
reipl	Performs an IPL according to the specifications in `/sys/firmware/reipl`.	“Rebooting from an alternative source” on page 106
dump	Creates a dump according to the specifications in `/sys/firmware/dump`.	*Using the Dump Tools*, SC33-8412
dump_reipl	Performs the `dump` action followed by the `reipl` action.	*Using the Dump Tools*, SC33-8412
vmcmd	For Linux on z/VM, issues one or more z/VM CP commands according to the specifications in `/sys/firmware/vmcmd`.	“Configuring z/VM CP commands as a shutdown action” on page 119

Use **lsshut** to find out which shutdown action is configured for each shutdown trigger, see “lsshut - List the current system shutdown actions” on page 632.

Use the applicable command to configure the shutdown action for a shutdown trigger:

- For `halt`, `poff`, and `reboot` use **`chshut`**, see "chshut - Control the system shutdown actions" on page 554.
- For `restart` and `panic` use **`dumpconf`**, see *Using the Dump Tools*, SC33-8412.

Possible overrides

- For distributions that map `halt` to `poff`, the action that is specified for `halt` is ignored and the action that is specified for `poff` is triggered instead.
- If kdump is set up for a Linux instance, kdump is started to create a dump, regardless of the shutdown actions that are specified for `restart` and `panic`. With kdump, these settings act as a backup that is used only if kdump fails.

Note: kdump is not a shutdown action that you can set as a sysfs attribute value for a shutdown trigger. See *Using the Dump Tools*, SC33-8412 about how to set up kdump.

The shutdown configuration in sysfs

The configured shutdown action for each shutdown trigger is stored in a sysfs attribute `/sys/firmware/shutdown_actions/on_<`*`trigger`*`>`.

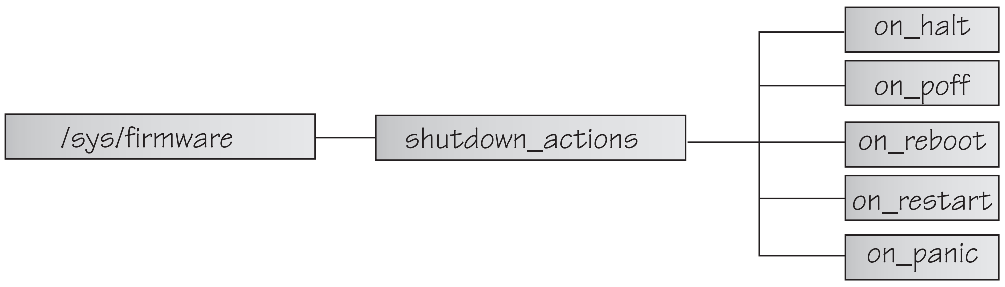


Figure 30. sysfs branch with shutdown action settings

The preferred way to read or change these settings is using the **`lsshut`**, **`chshut`**, and **`dumpconf`** commands. Alternatively, you can read and write to the `/sys/firmware/shutdown_actions/on_<`*`trigger`*`>` attributes.

Examples

- This command reads the shutdown setting for the `poff` shutdown trigger.

```
# cat /sys/firmware/shutdown_actions/on_poff
stop
```

- This command changes the setting for the `restart` shutdown trigger to `ipl`:

```
# echo ipl > /sys/firmware/shutdown_actions/on_restart
```

Details for the `ipl`, `reipl`, `dump`, and `vmcmd` shutdown actions are contained in the corresponding subdirectories in `/sys/firmware`. For example, `/sys/firmware/ipl` contains specifications for an IPL device and other IPL parameters.

Configuring z/VM CP commands as a shutdown action

Use **chshut** and **dumpconf** to configure a CP command as a shutdown action, or directly write to the relevant sysfs attributes.

Before you start: This information applies to Linux on z/VM only.

Two attributes are required to set a z/VM CP command as a shutdown action for a trigger *<trigger>*:

- /sys/firmware/shutdown_actions/on_*<trigger>* must be set to vmcmd.
- /sys/firmware/vmcmd/on_*<trigger>* specifies the z/VM CP command.

Example

The commands in this example configure the z/VM CP LOGOFF command as the shutdown action for the poff shutdown trigger:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_poff
# echo LOGOFF > /sys/firmware/vmcmd/on_poff
```

Figure 31 illustrates this example.

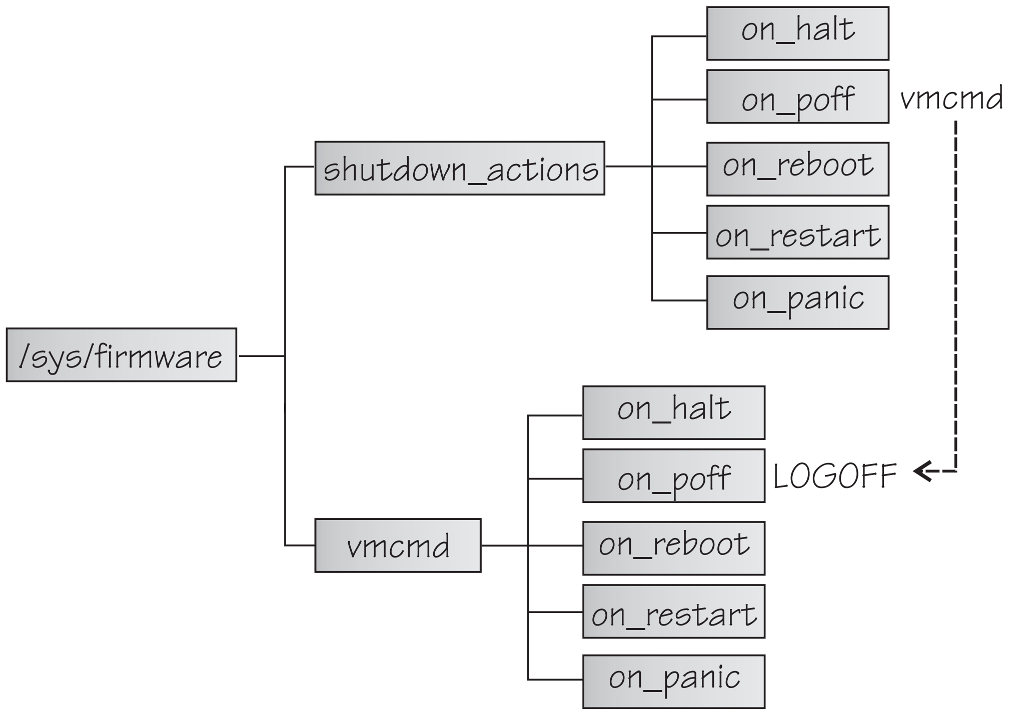


Figure 31. sysfs branch with shutdown action settings

Use the **chshut** and **dumpconf** commands to change your settings. As an alternative, you can write directly to the sysfs attributes. The values of the attributes in the /sys/firmware/vmcmd directory must conform to these rules:

- The value must be a valid z/VM CP command.
- The commands, including any z/VM user IDs or device numbers, must be specified with uppercase characters.

- Commands that include blanks must be delimited by double quotation marks (").
- The value must not exceed 127 characters.

You can specify multiple z/VM CP commands that are separated by the newline character "\n". Each newline is counted as one character. When writing values with multiple commands, use this syntax to ensure that the newline character is processed correctly:

```
# echo -e <cmd1>\n<cmd2>... | cat > /sys/firmware/vmcmd/on_<trigger>
```

where *<cmd1>*\n*<cmd2>*... are two or more z/VM CP commands and on_*<trigger>* is one of the attributes in the vmcmd directory.

Example

The commands in this example configure two z/VM CP commands as the shutdown action for the poff shutdown trigger. First a message is sent to user OPERATOR, and then the LOGOFF command is issued.

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_poff
# echo -e "MSG OPERATOR Going down\nLOGOFF" | cat > /sys/firmware/vmcmd/on_poff
```

The **-e echo** option and redirect through **cat** are required because of the newline character.

Chapter 9. Remotely controlling virtual hardware - snipl

snipl is a command-line tool for remotely controlling virtual System z hardware.

This information applies to simple network IPL (snipl) version 2.3.0. You can obtain the **snipl** package at www.ibm.com/developerworks/linux/linux390/snipl.html.

You can use **snipl** to activate and deactivate virtual System z hardware with Linux instances. You can set up a Linux instance on a mainframe system or on a different hardware platform for running **snipl**.

snipl helps you to automate tasks that are typically performed by human operators, for example, through the graphical interfaces of the HMC or SE. Automation is required, for example, for failover setups within Linux clusters.

snipl can run in LPAR mode or in z/VM mode.

Attention: **snipl** is intended for use by experienced system programmers and administrators. Incautious use of **snipl** can result in unplanned downtime and loss of data.

snipl security

snipl commands, including passwords, are sent in cleartext. Use a physically secure network to communicate. Likewise, **snipl** configuration files are cleartext, and might store passwords.

LPAR mode

In LPAR mode, **snipl** provides basic System z support element (SE) functions.

With **snipl** in LPAR mode you can perform the following tasks:
- Activate, reset, or deactivate an LPAR.
- Load (IPL) an LPAR from a disk device, for example, a DASD device or a SCSI device.
- Create a dump on a DASD or SCSI dump device.
- Send commands to the operating system and retrieve operating system messages.

Setting up snipl for LPAR mode

The Linux instance where **snipl** runs requires access to all SEs that control LPARs you want to work with.

snipl uses the "hwmcaapi" network management application programming interfaces (API) provided by the SE. The API establishes an SNMP network connection and uses the SNMP protocol to send and retrieve data. The libraries that implement the API are available from IBM Resource Link® at www.ibm.com/servers/resourcelink.

Customize the API settings on the HMC or SE you want to connect to:

- Configure SNMP support.
- Add the IP address of the Linux instance where **snipl** runs and set the SNMP community.
- In the firewall settings, ensure that UDP port 161 and TCP port 3161 are enabled.

If **snipl** in LPAR mode repeatedly reports a timeout, the specified SE is most likely inaccessible or not configured properly. For details about configuring the HMC or SE, see the following publications:
- The *Support Element Operations Guide* for your mainframe system.
- The applicable *Hardware Management Console Operations Guide*.
- *System z Application Programming Interfaces*, SB10-7030
- *S/390 Application Programming Interfaces*, SC28-8141

You can obtain these publications from IBM Resource Link at www.ibm.com/servers/resourcelink.

Command line syntax (LPAR mode)

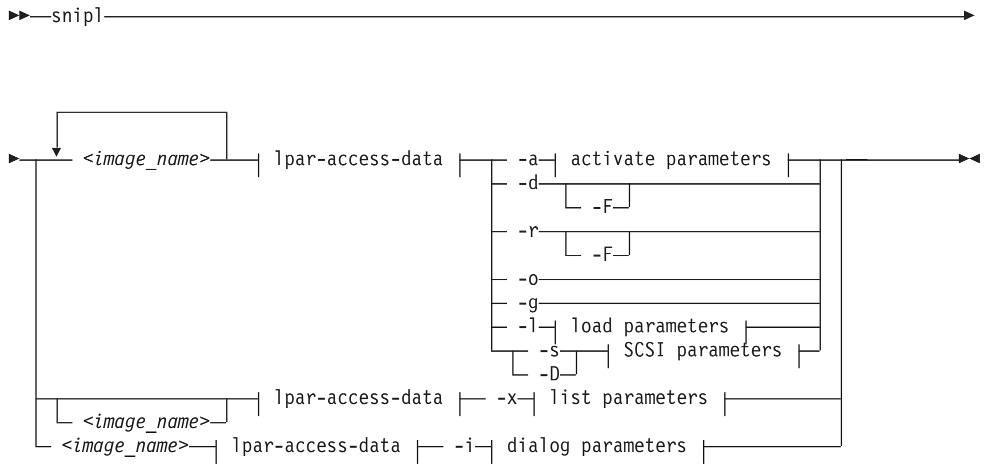
There is a generic syntax with main options. Each main option has a specific set of parameters.

"Overview for LPAR mode" summarizes **snipl** command in LPAR mode. Details for each option are provided in context in the sections that follow.

Overview for LPAR mode

On the command line, a **snipl** command in LPAR mode always requires a main option, access data, and, with one exception, specifications for one or more LPARs.

LPAR mode: overview



Where:

<image_name>
: specifies an LPAR. If **snipl** directly accesses the SE, this is the LPAR name as defined in the hardware setup.

If **snipl** accesses the SE through an HMC, the specification has the format *<mainframe_system>-<lpar_name>* where *<mainframe_system>* is the name that identifies the System z mainframe on the HMC. If you are using a **snipl** configuration file that defines an alias for an LPAR, you can specify the alias.

SE Example: `lpar204`

HMC Example: `z02-lpar204`

A **snipl** command applies to one or more LPARs that are controlled by the same HMC or SE. If multiple LPARs are specified, it is assumed that all LPARs are controlled by the same HMC or SE as the first LPAR. Other LPARs are ignored.

|lpar-access-data|
is described in "Specifying access data for LPAR mode."

-a, -d, -r, -o, -g
are described in "Activate, deactivate, reset, stop, or get status information" on page 125.

-l is described in "Perform an IPL operation from a CCW device" on page 126.

-s, -D
are described in "Perform an IPL or dump operation from a SCSI device" on page 127.

-x is described in "List LPARs" on page 129.

-i is described in "Emulate the Operating Systems Messages applet" on page 130.

-F or --force
unconditionally forces the operation.

-v or --version
displays the version of **snipl** and exits.

-h or --help
displays a short usage description and exits. To view the man page enter **man snipl**.

Specifying access data for LPAR mode

The **snipl** command requires access data for the HMC or SE that controls a particular LPAR.

lpar-access-data:

```
                      (1)  ┌─ -p public──────┐   ┌─ -f <defaultfile>─┐
├── -L <ip_address>────────┼─────────────────┼───┼───────────────────┼──────►
                           ├─ -p <community>─┤   └─ -f <filename>────┘
                           └─ -P─────────────┘

   ┌─ --timeout 60000─────┐
►──┼──────────────────────┼─────────────────────────────────────────────────┤
   └─ --timeout <timeout>─┘
```

Notes:

1 **-L** can be omitted if the required information is specified through a configuration file.

-L *<ip_address>* **or --lparserver** *<ip_address>*
specifies the IP address or host name of the HMC or SE that controls the LPAR or LPARs you want to work with. You can omit this parameter if the IP address or host name is specified through a configuration file.

-p *<community>* **or --password** *<community>*
specifies the password in the SNMP configuration settings on the SE that controls the LPAR or LPARs you want to work with. This parameter can also be specified through a configuration file. The default password is `public`.

Note: The default password feature is deprecated and will be removed in a subsequent release.

-P or --promptpassword
prompts for a password in protected entry mode.

-f *<filename>* **or --configfilename** *<filename>*
specifies the name of a configuration file that maps LPARs to the corresponding specifications for the HMC or SE address and password (community).

If no configuration file is specified, the user-specific default file `~/.snipl.conf` is used. If this file does not exist, the system default file `/etc/snipl.conf` is used.

Be sure that the command line parameters you provide uniquely identify the configuration-file section you want to work with. If you specify multiple LPARs on the command line, only the first specification is used to identify the section. If your specifications map to multiple sections, the first match is processed.

If conflicting specifications are provided through the command line and the configuration file, the command line specification is used.

If a configuration file is neither specified nor available at the default locations, all required parameters must be specified on the command line.

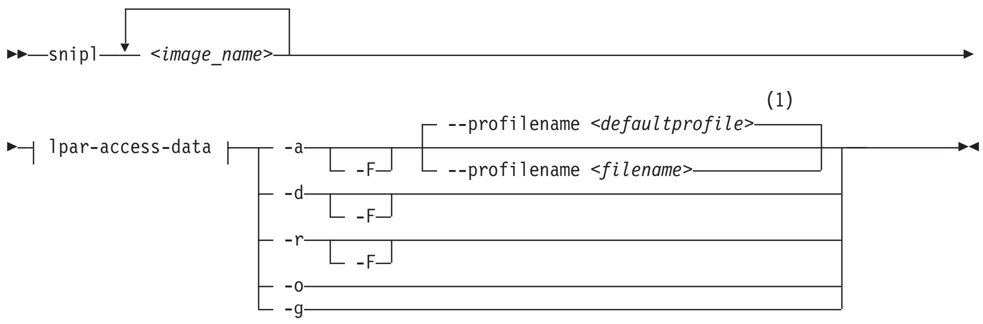
For more information about the configuration file, see "The snipl configuration file" on page 135.

--timeout *<timeout>*
specifies the timeout in milliseconds for general management API calls. The default is 60000 ms.

Activate, deactivate, reset, stop, or get status information

Several main options follow a simple command syntax that requires specifications for one or more LPARs and the corresponding access data.

LPAR mode: -a, -d, -r, -o, -g options



Notes:

1 If not specified, the HMC or SE default profile for the specified LPAR is used.

Where:

<image_name>
see "Overview for LPAR mode" on page 122.

|lpar-access-data|
see "Specifying access data for LPAR mode" on page 123.

-a or --activate
activates the specified LPARs.

--profilename *<filename>*
specifies an activation profile. If omitted, the SE or an HMC default profile for the specified LPAR is used.

-d or --deactivate
deactivates the specified LPARs.

-r or --reset
resets the specified LPARs.

-o or --stop
stops all CPUs for the specified LPARs.

-g or --getstatus
returns the status for the specified LPARs.

-F or --force
unconditionally forces the operation.

Examples

- The following command deactivates an LPAR SZ01LP02 with the force option:

```
# snipl SZ01LP02 -L 192.0.2.4 -P -d -F
Enter password:
Warning : No default configuration file could be found/opened.
processing......
SZ01LP02: acknowledged.
```

- The following command retrieves the status for an LPAR SZ01LP03:

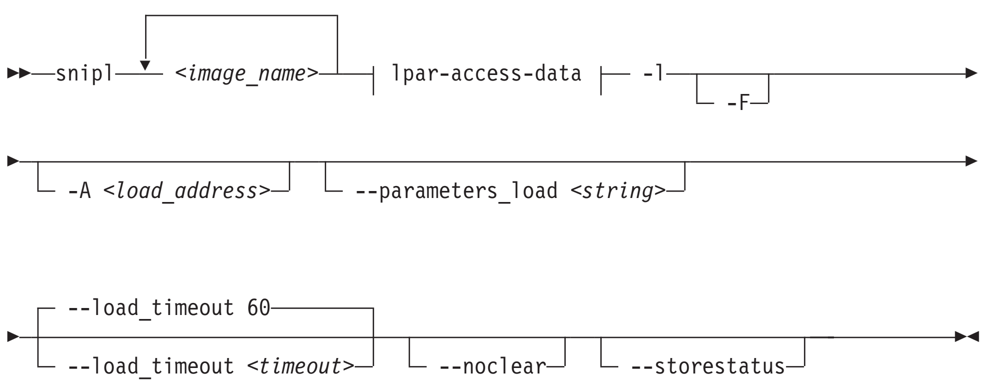
```
# snipl SZ01LP03 -L 192.0.2.4 -P -g
Enter password:
Warning : No default configuration file could be found/opened.
status of sz01lp03:  operating
```

Perform an IPL operation from a CCW device

To IPL an LPAR from a CCW device, **snipl** requires specifications for the LPAR, the corresponding access data, and the IPL device. There are also several optional parameters.

For IPL from a SCSI device, see "Perform an IPL or dump operation from a SCSI device" on page 127.

LPAR mode: IPL from CCW



Where:

<image_name>
: specifies the LPARs for which to perform the IPL. If multiple LPARs are specified, the same IPL device and IPL parameters are used for all of them. See also "Overview for LPAR mode" on page 122.

|lpar-access-data|
: see "Specifying access data for LPAR mode" on page 123.

-l or --load
: performs an IPL for the specified LPARs.

-F or --force
: unconditionally forces the IPL operation.

-A <*loadaddress*> **or --address_load** <*loadaddress*>
specifies the hexadecimal four-digit device number of the IPL device. If this parameter is omitted, the IPL device of the most recent IPL of the LPAR is used.

--parameters_load <*string*>
specifies a parameter string for IPL. If this parameter is omitted, the string of the most recent IPL of the LPAR is used.

--load_timeout <*timeout*>
specifies the maximum time for load completion in seconds. The timeout must be in the range of 60 - 600 seconds. The default timeout is 60 seconds.

If the timeout expires, control is returned without an indication about the success of the IPL operation.

--noclear
prevents the memory from being cleared before loading.

--storestatus
stores status before performing the IPL. This option implies **--noclear** and also prevents the main memory from being cleared before loading.

Example: The following command performs an IPL from a CCW device with bus ID 0.0.5119 for an LPAR SZ01LP02:

```
# snipl SZ01LP02 -L 192.0.2.4 -P -l -A 5119
Enter password:
Warning : No default configuration file could be found/opened.
processing......
SZ01LP02: acknowledged.
```

Perform an IPL or dump operation from a SCSI device

To IPL an LPAR from a SCSI device, **snipl** requires specifications for the LPAR, the corresponding access data, the IPL device, target WWPN, and LUN. There are also several optional parameters.

For IPL from a CCW device, see "Perform an IPL operation from a CCW device" on page 126.

LPAR mode: SCSI IPL or dump

```
              ┌──────────────────┐
              ▼                  │
►►──snipl──────  <image_name>────┴─┤ lpar-access-data ├──┬─ -s─┬──┬────┬──────────────►
                                                          └─ -D─┘  └ -F─┘

►──┬──────────────────────┬──┬──────────────────────────────┬──────────────────────────►
   └─ -A <load_address>───┘  └─ --parameters_load <string>──┘

►──┬──────────────────────────────┬──┬──────────────────────────────┬─────────────────►
   └─ --wwpn_scsiload <portname>──┘  └─ --lun_scsiload <unitnumber>─┘

►──┬──────────────────────────────┬──┬──────────────────────────────────┬─────────────►
   └─ --bps_scsiload <selector>───┘  └─ --ossparms_scsiload <string>────┘

►──┬─────────────────────────────────────────┬────────────────────────────────────────►◄
   └─ --bootrecord_scsiload <hexaddress>─────┘
```

Where:

<image_name>
specifies the LPARs for which to perform the IPL or dump operation. If multiple LPARs are specified, the same command parameters apply to all of them. See also "Overview for LPAR mode" on page 122.

|lpar-access-data|
see "Specifying access data for LPAR mode" on page 123.

-s or --scsiload
performs an IPL from a SCSI device for the specified LPARs.

-D or --scsidump
creates a dump for the specified LPAR to a SCSI device.

-F or --force
unconditionally forces the operation.

-A *<loadaddress>* **or --address_load** *<loadaddress>*
specifies the hexadecimal four-digit device number of the IPL device. If this parameter is omitted, the IPL device of the most recent SCSI IPL of the LPAR is used.

--parameters_load *<string>*
specifies a parameter string for IPL. If this parameter is omitted, the string of the most recent SCSI IPL of the LPAR is used.

--wwpn_scsiload *<portname>*
specifies the worldwide port name (WWPN) for the SCSI IPL device. If fewer

than 16 characters are specified, the WWPN is padded with zeroes at the end. If this parameter is omitted, the WWPN of the most recent SCSI IPL of the LPAR is used.

--lun_scsiload *<unitnumber>*
: specifies the logical unit number (LUN) for the SCSI IPL device. If fewer than 16 characters are specified, the LUN is padded with zeroes at the end. If this parameter is omitted, the LUN of the most recent SCSI IPL of the LPAR is used.

--bps_scsiload *<selector>*
: specifies the boot program that is required for the SCSI IPL device. Selector values are in the range 0 - 30. If this parameter is omitted, the boot program of the most recent SCSI IPL of the LPAR is used.

--ossparms_scsiload *<string>*
: specifies an operating system-specific parameter string for IPL from a SCSI device. If this parameter is omitted, the string of the most recent SCSI IPL of the LPAR is used. This parameter string is ignored by the boot program and passed to the operating system or dump program to be loaded. For example, you can specify additional kernel parameters for Linux (see "Specifying kernel parameters when booting Linux" on page 27).

--bootrecord_scsiload *<hexaddress>*
: specifies the boot record logical block address for the SCSI IPL device. If fewer than 16 characters are specified, the address is padded with zeroes at the end. If this parameter is omitted, the address of the most recent SCSI IPL of the LPAR is used.

Example: The following command performs a SCSI IPL for an LPAR SZ01LP00:

```
# snipl SZ01LP00 -L 192.0.2.4 -P -s -A 3d0f --wwpn_scsiload 500507630303c562 \
--lun_scsiload 4010404900000000
Enter password:
Warning : No default configuration file could be found/opened.
processing...
SZ01LP00: acknowledged.
```

Note: Instead of using the continuation sign (\) at the end of the first line, you can specify the complete command on a single line.

List LPARs

To list all LPARs that are controlled by an HMC or SE, **snipl** requires specifications for the HMC or SE and the corresponding access data.

Use the **-x** option to list all LPARs of a System z mainframe.

LPAR mode: list

```
►►──snipl──┬──────────────┬──┤ lpar-access-data ├── -x──────────────►◄
           └─ <image_name>┘
```

Where:

<image_name>
specifies an LPAR to identify a section in the **snipl** configuration file. Omit this parameter if an HMC or SE is specified with the **-L** option (see “Overview for LPAR mode” on page 122).

|lpar-access-data|
see “Specifying access data for LPAR mode” on page 123.

-x or --listimages
retrieves a list of all LPARs from the specified HMC or SE. If an HMC is specified, all LPARs for all managed mainframe systems are listed.

Example: The following command lists the LPARs for an SE with IP address 192.0.2.4:

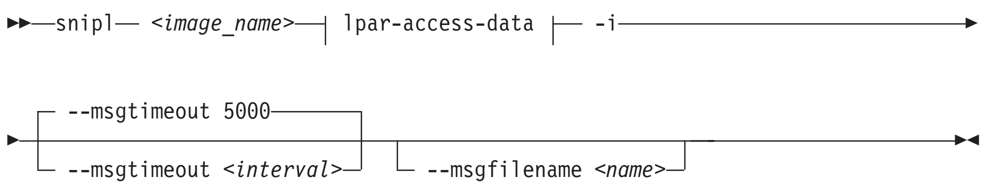
```
# snipl -L 192.0.2.4 -P -x
Enter password:
Warning : No default configuration file could be found/opened.

available images for server 192.0.2.4 :

    SZ01LP00      SZ01LP01      SZ01LP02      SZ01LP03
```

Emulate the Operating Systems Messages applet

To emulate the HMC or SE Operating Systems Messages applet, **snipl** requires specifications for the LPAR and the corresponding access data. There are also optional parameters.

Use the **-i** option to start an emulation of the HMC or SE Operating Systems Messages applet for a specified LPAR. End the emulation with CTRL+D.

LPAR mode: dialog



Where:

<image_name>
specifies the LPAR for which you want to emulate the HMC or SE Operating Systems Messages applet (see also “Overview for LPAR mode” on page 122).

|lpar-access-data|
see “Specifying access data for LPAR mode” on page 123.

-i or --dialog
starts an emulation of the HMC or SE Operating System Message applet for the specified LPAR.

--msgtimeout *<interval>*
specifies the timeout for retrieving operating system messages in milliseconds. The default value is 5000 ms.

-M *<name>* **or --msgfilename** *<name>*
specifies a file to which the operating system messages are written in addition to stdout. If no file is specified, the operating system messages are written to stdout only.

Example: The following command opens an emulation of the SE Operating Systems Messages applet with the operating system instance that runs on LPAR `SZ01LP02`. During the emulation session, the operating system messages are written to a file, `SZ01LP02.transcript`.

```
# snipl SZ01LP02 -L 192.0.2.4 -P -i -M SZ01LP02.transcript
Enter password:
Warning : No default configuration file could be found/opened.
processing......
...
```

z/VM mode

With **snipl** in z/VM mode, you can log on, reset, or log off a z/VM guest virtual machine.

Setting up snipl for z/VM mode

The Linux instance where **snipl** runs requires access to the systems management API of all z/VM systems that host z/VM guest virtual machines you want to work with.

snipl in z/VM mode uses the systems management application programming interfaces (APIs) of z/VM. How **snipl** communicates with the API on the z/VM system depends on your z/VM system version and on your system setup.

If **snipl** in z/VM mode repeatedly reports “RPC: Port mapper failure - RPC timed out”, it is most likely that the z/VM system is inaccessible, or not set up correctly. Although only one of the communication methods uses RPC, this method is the fallback method that is tried if the other method fails.

Using a SMAPI request server

snipl can access the systems management API through a SMAPI request server. The following configuration is required for the z/VM systems you want to work with:

- An AF_INET based SMAPI request server must be configured.
- A port on which the request server listens must be set up.
- A z/VM user ID to be specified with the **snipl** command must be set up. This user ID must be authorized for the request server.

For more information, see *z/VM Systems Management Application Programming*, SC24-6234.

Using a VSMSERVE service machine

snipl can access the systems management API through a VSMSERVE service machine on your z/VM system. The following configuration is required for the z/VM systems you want to work with:

- The VSMSERVE service machine must be configured and authorized for the directory manager.

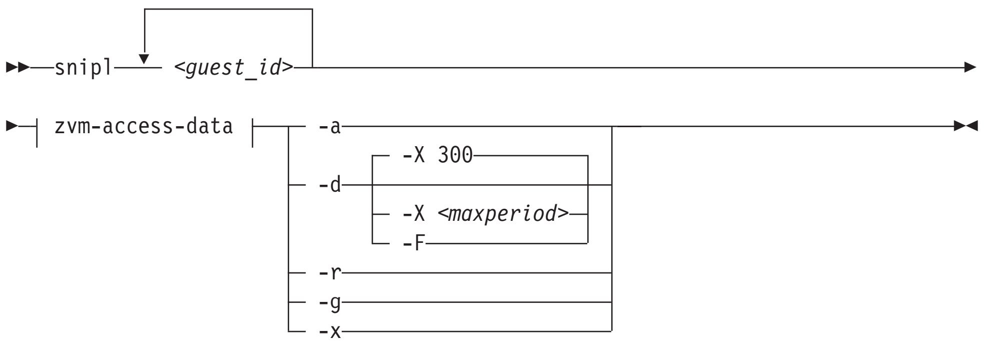
- The `vsmapi` service must be registered.
- A z/VM user ID to be specified with the snipl command must be set up. This user ID must be authorized for VSMSERVE.

For more information, see *z/VM Systems Management Application Programming*, SC24-6122-02 or earlier.

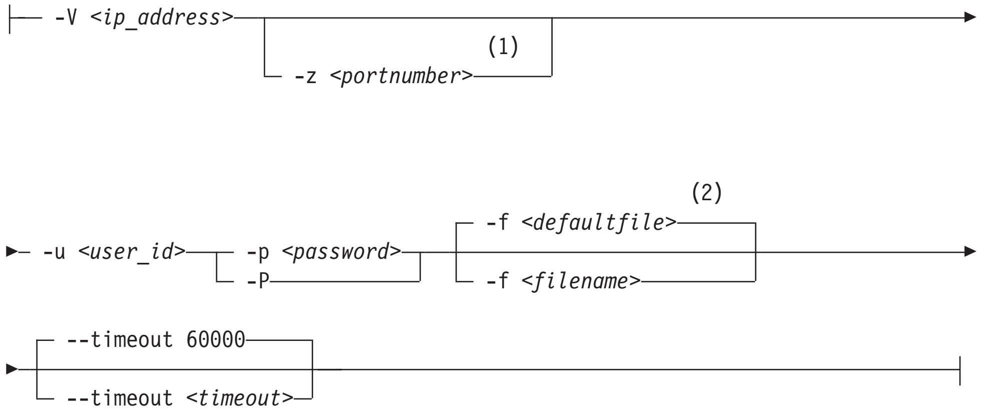
Command-line syntax (z/VM mode)

In z/VM mode, the **`snipl`** command requires specification for a guest virtual machine, credentials, and other access data for the systems management API. There are also several optional parameters.

snipl command syntax (z/VM mode)



zvm-access-data:



Notes:

1 Required for connections through a SMAPI request server, unless the port is specified through a configuration file.

2 **-V**, **-u**, and **-p** can be omitted if the required data is specified through a configuration file.

Where:

<guest_id>
specifies the z/VM guest virtual machine that you want to work with. Specify multiple z/VM user IDs to perform the same action for multiple z/VM guest virtual machines.

If you are using a **snipl** configuration file that defines an alias for a z/VM guest virtual machine, you can specify the alias.

You can omit this parameter for the **-x** option if other specifications on the command line identify a section in the configuration file.

-V *<ip_address>* **or --vmserver** *<ip_address>*
specifies the IP address or host name of the SMAPI request server or

VSMSERVE service machine through which the specified z/VM guest virtual machines are controlled. This option can be omitted if defined in the configuration file.

-z *<portnumber>* **or --port** *<portnumber>*
specifies the port at which the SMAPI request server listens.

-u *<user_id>* **or --userid** *<user_id>*
specifies a z/VM user ID that is authorized to access the SMAPI request server or VSMSERVE service machine. This option can be omitted if defined in the configuration file.

-p *<password>* **or --password** *<password>*
specifies the password for the z/VM user ID specified with **--userid**. This option can be omitted if defined in the configuration file.

-P or --promptpassword
prompts for a password in protected entry mode.

-f *<filename>* **or --configfilename** *<filename>*
specifies the name of a configuration file that maps z/VM guest virtual machines to the corresponding specifications for the SMAPI request server or VSMSERVE service machine, the authorized z/VM user ID, and the password.

If no configuration file is specified, the user-specific default file `~/.snipl.conf` is used. If this file does not exist, the system default file `/etc/snipl.conf` is used.

Be sure that the command-line parameters you provide uniquely identify the configuration-file section you want to work with. If you specify multiple z/VM guest virtual machines on the command line, only the first specification is used to identify the section. If your specifications map to multiple sections, the first match is processed.

If conflicting specifications are provided through the command line and the configuration file, the command-line specification is used. If no configuration file is used, all required parameters must be specified on the command line.

For more information about the configuration file, see "The snipl configuration file" on page 135.

--timeout *<timeout>*
specifies the timeout in milliseconds for general management API calls. The default is 60000 ms.

-a or --activate
logs on the specified z/VM guest virtual machines.

-d or --deactivate
logs off the specified z/VM guest virtual machines.

-X *<maxperiod>* **or --shutdowntime** *<maxperiod>*
specifies the maximum period, in seconds, granted for graceful completion before CP FORCE commands are issued against the specified z/VM guest virtual machines. By default, the maximum period is 300 s.

-F or --force
immediately issues CP FORCE commands to log off the specified z/VM guest virtual machines. This parameter is equivalent to **-X 0**.

-r or --reset
logs off the specified z/VM guest virtual machines and then logs them back on.

-g or --getstatus
returns the status for the specified z/VM guest virtual machines.

-x or --listimages
lists the z/VM guest virtual machines as specified in a configuration-file section (see "The snipl configuration file"). You can identify the configuration file section with the **-V** parameter, by specifying a z/VM guest virtual machine, or by specifying a z/VM guest virtual machine and the **-u** parameter.

-v or --version
displays the version of **snipl** and exits.

-h or --help
displays a short usage description and exits. To view the man page enter **man snipl**.

Example

The following command logs on two z/VM guest virtual machines:

```
# snipl sndlnx04 sndlnx05 -V sandbox.www.example.com -u sndadm01 -p pw42play -a
Warning : No default configuration file could be found/opened.
processing......
* ImageActivate : Image sndlnx04 Request Successful
* ImageActivate : Image sndlnx05 Request Successful
```

The snipl configuration file

Use the **snipl** configuration file to provide parameter values to **snipl** instead of specifying all values on the command line.

See "Specifying access data for LPAR mode" on page 123 or "Command-line syntax (z/VM mode)" on page 132 about how to include a configuration file when issuing a **snipl** command.

A **snipl** configuration file contains one or more sections. Each section consists of multiple lines with specifications of the form <*keyword*>=<*value*> for either a z/VM system or an SE.

The following rules apply to the configuration file:
- Lines that begin with a number sign (#) are comment lines. A number sign in the middle of a line makes the remaining line a comment.
- Empty lines are permitted.
- The specifications are not case-sensitive.
- The same configuration file can contain sections for snipl in both LPAR mode and z/VM mode.
- In a <*keyword*>=<*value*> pair, one or more blanks are allowed before or after the equal sign (=).

Table 16 on page 136 summarizes the keywords for the configuration file and the command-line equivalents for LPAR mode and z/VM mode.

Table 16. snipl configuration file keywords

Keyword	Value for LPAR mode	Value for z/VM mode	Command-line equivalent
server (required)	Starts a configuration file section by specifying the IP address or host name of an HMC or SE.	Starts a configuration file section by specifying the IP address or host name of a SMAPI request server or VSMSERVE service machine.	(See note 1)
type (required)	LPAR	VM	(See note 1)
user (See note 2)	n/a	A z/VM user ID that is authorized for the SMAPI request server or VSMSERVE service machine.	**-u** or **--user**
password (See note 3)	The value for `community` in the SNMP settings of the SE. If not specified through either the configuration file or the command, the default, `public`, is used.	The password for the z/VM user ID specified with the **user** keyword. (See note 2)	**-p** or **--password**
port	n/a	Required if the **server** keyword specifies the IP address or host name of a SMAPI request server.	**-z** or **--port**
image A valid section must have one or more lines with this keyword.	An LPAR name as defined in the mainframe hardware configuration. If the **server** keyword specifies an HMC, the specification begins with the name that identifies the System z mainframe on the HMC, followed by a hyphen (-), followed by the LPAR name. You can define an alias name for the LPAR by appending a forward slash (/) to the LPAR name and specifying the alias following the slash.	A z/VM user ID that specifies a target z/VM guest virtual machine. You can define an alias name for the z/VM user ID by appending a forward slash (/) to the ID and specifying the alias, following the slash.	A list of one or more items that are separated by blanks and specified without a switch.

Note:

1. Jointly, the **server** and **type** keywords are equivalent to the command-line option **-L** for LPAR mode or to **-V** for z/VM mode.
2. Can be omitted and specified on the command line instead.
3. Do not include passwords in the **`snipl`** configuration file unless the security policy at your installation permits you to do so.

Figure 32 on page 137 shows a configuration file example with multiple sections, including sections for LPAR mode and for z/VM mode.

```
# z/VM system for Linux training sessions
server = sandbox.www.example.com
type = VM
password = pw42play
port = 44444
user = sndadm01
image = sndlnx01
image = sndlnx02
image = sndlnx03/tutor
image = sndlnx04
image = sndlnx05
image = sndcms01/c1

# SE for production SZ01
Server=192.0.2.4
type=LPAR
image=SZ01LP00
image=SZ01LP01
image=SZ01LP02
image=SZ01LP03

# HMC for test SZ02
Server=192.0.2.2
type=LPAR
image=Z02-SZ02LP00/Z0200
image=Z02-SZ02LP01
image=Z02-SZ02LP02
image=Z02-SZ02LP03

# Production VM 05 - uses VSMSERVE so no port
server = 192.0.2.20
type = VM
user = VM05MAIN
image = VM05G001
image = VM05G002
image = VM05G003
image = VM05G004
```

Figure 32. Example of a snipl configuration file

Examples

The examples that follow assume that the configuration file of Figure 32 is used.

- The following command logs on two z/VM guest virtual machines, `sndlnx01` and `sndlnx03` (with alias `tutor`). In the example, the command output shows that `sndlnx03` is already logged on.

```
# snipl sndlnx01 sndlnx03 -V sandbox.www.example.com -z 44444 -u sndadm01 -p pw42play -a
Warning : No default configuration file could be found/opened.
processing......
* ImageActivate : Image sndlnx01 Request Successful
* ImageActivate : Image sndlnx03 Image Already Active
```

  Assuming that the configuration file of Figure 32 is available at `/etc/xcfg`, an equivalent command would be:

```
# snipl sndlnx01 tutor -a -f /etc/xcfg
Server sandbox.www.example.com from config file /etc/xcfg is used
processing......
* ImageActivate : Image sndlnx01 Request Successful
* ImageActivate : Image sndlnx03 Image Already Active
```

Assuming that the configuration file of Figure 32 on page 137 is used by default, an equivalent command would be:

```
# snipl sndlnx01 tutor -a
Server sandbox.www.example.com from config file /etc/snipl.conf is used
processing......
* ImageActivate : Image sndlnx01 Request Successful
* ImageActivate : Image sndlnx03 Image Already Active
```

- The following command performs an IPL for an LPAR SZ01LP03:

```
# snipl SZ01LP03 -L 192.0.2.4 -l -P -A 5000
Enter password:
Warning : No default configuration file could be found/opened.
processing......
SZ01LP03: acknowledged.
```

  Assuming that the configuration file of Figure 32 on page 137 is available at /etc/xcfg, an equivalent command would be:

```
# snipl SZ01LP03 -l -P -A 5000 -f /etc/xcfg
Enter password:
Server 192.0.2.4 from config file /etc/xcfg is used
processing......
SZ01LP03: acknowledged.
```

  Assuming that the configuration file of Figure 32 on page 137 is used by default, an equivalent command would be:

```
# snipl SZ01LP03 -l -P -A 5000
Enter password:
Server 192.0.2.4 from config file /etc/snipl.conf is used
processing......
SZ01LP03: acknowledged.
```

- Assuming that the configuration file of Figure 32 on page 137 is available at /etc/xcfg, the following command lists the z/VM guest virtual machines as specified in the section for sandbox.www.example.com:

```
# snipl -V sandbox.www.example.com -f /etc/xcfg -x
available images for server sandbox.www.example.com and userid SNDADM01 :

        sndlnx01        sndlnx02        sndlnx03        sndlnx04
        sndlnx05        sndcms01
```

Connection errors and return codes

You might receive error indications from **snipl** or from the SE.

snipl return codes

Successful **snipl** commands return 0. If an error occurs, **snipl** writes a short message to stderr and completes with a return code other than 0.

The following return codes indicate **snipl** syntax errors or specifications that are not valid:

1 An unknown command option was specified.

2 A command option with an invalid value was specified.

3 A command option was specified more than once.

4	Conflicting command options was specified.
5	No command option was specified.
6	No SE, HMC, SMAPI request server or VSMSERVE service machine was specified on the command line or through a configuration file.
7	No LPAR or z/VM guest virtual machine was specified.
8	No z/VM user ID was specified on the command line or through a configuration file.
9	No password was specified on the command line or through a configuration file.
10	A specified LPAR or z/VM guest virtual machine does not exist on the specified SE or z/VM system.
22	More than one LPAR was specified for option **`--dialog`**.

The following return codes indicate setup errors or program errors:

30	An error occurred while loading one of the systems management API 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	A response from the HMC or SE could not be interpreted.
60	The response buffer is too small for a response from the HMC or SE.
90	A storage allocation failure occurred.
99	A program error occurred.

Connection errors

If a connection error occurs (for example, a timeout), **`snipl`** sends a message to `stderr`.

To recover connection errors, try again to issue the command. If the problem persists, a networking failure is most likely. In this case, increase the timeout value.

Return codes from the SE

Error messages from the SE have the format
`<`*`LPAR_name`*`>: <`*`message`*`> - rc is <`*`rc`*`>`.

In the message, <*rc*> is a return code from the network management application programming interfaces (HWMCAAPI) on the SE.

Example

```
LPARLNX1: not acknowledged – command was not successful – rc is 135921664
```

To interpret these return codes, see *System z Application Programming Interfaces*, SB10-7030. You can obtain this publication from IBM Resource Link at www.ibm.com/servers/resourcelink.

STONITH support (snipl for STONITH)

The STONITH implementation is part of the Heartbeat framework of the High Availability Project.

STONITH is usually used as part of this framework but can also be used independently. **snipl** provides a plug-in to STONITH.

For a general description of the STONITH technology go to linux-ha.org.

Before you begin

- STONITH requires a configuration file that maps LPARs and z/VM guest virtual machines to the specifications for the corresponding SE, HMC or z/VM system. The **snipl** for STONITH configuration file has the same syntax as the **snipl** configuration file, see "The snipl configuration file" on page 135.
- The SEs, HMCs and z/VM systems you want to work with must be set up as described in "Setting up snipl for LPAR mode" on page 121 and "Setting up snipl for z/VM mode" on page 131.

Using stonith

When using **stonith** commands for Linux on z/VM or for Linux in LPAR mode you must provide *<keyword>*=*<value>* pairs as described in "The snipl configuration file" on page 135. There are two ways to specify this information:

- On the command line with the **stonith** command, using the **-p** option and the **snipl_parm** keyword.
- Through a configuration file, using the **-p** option and the **snipl_file** keyword.

Unlike **snipl**, you must specify all parameters in the same way; all parameters on the command line or all parameters in the configuration file.

stonith syntax (simplified)

```
►►──stonith── -t lic_vps── -p──┬─ "snipl_param <parameters>"─┬──────────────►
                                └─ "snipl_file <file>"───────┘

►─ -T──┬─ on────┬─ <image>──────────────────────────────────────────────────►◄
       ├─ off───┤
       └─reset──┘
```

Where:

-t *lic_vps*
: specifies the "server type". For STONITH with **snipl**, the server type is always `lic_vps`.

-p specifies parameters.

snipl_param *<parameters>*
: specifies comma-separated *<keyword>*=*<value>* pairs with the same keywords as used in the configuration file (see "The snipl configuration file" on page 135).

  For LPAR mode the following keywords are required:
  - server
  - type
  - password

- image

For z/VM mode the following keywords are required:

- server
- port (required if the z/VM system has been configured with a SMAPI request server rather than a VSMSERVE service machine)
- type
- user
- password
- image

`snipl_file` *<parameters>*
specifies a configuration file (see "The snipl configuration file" on page 135). The configuration file must contain all required keywords, including the password. The configuration file must always be specified explicitly. No file is used by default.

`-T` specifies the action to be performed.

`-on`
activates the specified LPAR or logs on the specified z/VM virtual machine.

`-off`
deactivates the specified LPAR or logs off the specified z/VM virtual machine.

`-reset`
resets the specified LPAR or z/VM virtual machine.

<image>
specifies the LPAR or z/VM virtual machine you want to work with. If you use the **snipl_param** parameter, the contained **image** keyword must specify the same LPAR or z/VM virtual machine.

For more information, see the **`stonith`** man page.

Examples

- This example command resets the z/VM guest virtual machine `sndlnx04`:

```
# stonith -t lic_vps -p "snipl_param server=sandbox.www.example.com,type=vm\
,user=sndadm01,password=pw42play,image=sndlnx04" -T reset sndlnx04
```

  Note: Instead of using the continuation sign (\) at the end of the first line, you can specify the complete command on a single line.

- With `/etc/xcfg` as shown in Example of a snipl configuration file, the following command is equivalent:

```
# stonith -t lic_vps -p "snipl_file /etc/xcfg" -T reset sndlnx04
```

Part 3. Storage

There are several System z specific storage device drivers for Linux on System z.

Newest version

You can find the newest version of this publication at
www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Restrictions

For prerequisites and restrictions see
www.ibm.com/developerworks/linux/linux390/development_technical.html
www.ibm.com/developerworks/linux/linux390/development_restrictions.html

Chapter 10. DASD device driver

The DASD device driver provides access to all real or emulated direct access storage devices (DASD) that can be attached to the channel subsystem of an IBM mainframe.

DASD devices include various physical media on which data is organized in blocks or records or both. The blocks or records in a DASD can be accessed for read or write in random order.

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

SCSI disks that are attached through an FCP channel are not classified as DASD. They are handled by the zfcp driver (see Chapter 11, "SCSI-over-Fibre Channel device driver," on page 185).

Features

The DASD device driver supports a wide range of disk devices and disk functions.

- The DASD device driver has no dependencies on the adapter hardware that is used to physically connect the DASDs to the System z hardware. You can use any adapter that is supported by the System z hardware (see www.ibm.com/systems/z/connectivity for more information).
- The DASD device driver supports ESS virtual ECKD type disks
- The DASD device driver supports the control unit attached physical ECKD (Extended Count Key Data) and FBA (Fixed Block Access) devices as summarized in Table 17:

Table 17. Supported control unit attached DASD

Device format	Control unit type	Device type
ECKD	1750	3380 and 3390
ECKD	2107	3380 and 3390
ECKD	2105	3380 and 3390
ECKD	3990	3380 and 3390
ECKD	9343	9345
ECKD	3880	3390
FBA	6310	9336
FBA	3880	3370

All models of the specified control units and device types can be used with the DASD device driver. This includes large devices with more than 65520 cylinders, for example, 3390 Model A. Check the storage support statement to find out what works for a particular distribution.

- The DASD device driver provides a disk format with up to three partitions per disk. See “System z compatible disk layout” on page 147 for details.
- The DASD device driver provides an option for extended error reporting for ECKD devices. Extended error reporting can support high availability setups.
- The DASD device driver supports parallel access volume (PAV) and HyperPAV on storage devices that provide this feature. The DASD device driver handles dynamic PAV alias changes on storage devices. For more information about PAV and HyperPAV, see *How to Improve Performance with PAV*, SC33-8414.
- The DASD device driver supports High Performance FICON, including multitrack requests, on storage devices that provide this feature.

What you should know about DASD

The DASD device driver supports various disk layouts with different partitioning capabilities. The DASD device naming scheme helps you to keep track of your DASDs and DASD device nodes.

The IBM label partitioning scheme

Linux on System z supports the same standard DASD format that is also used by traditional mainframe operating systems, but it also supports any other Linux partition table.

The DASD device driver is embedded into the Linux generic support for partitioned disks. As a result, you can use any partition table format that is supported by Linux for your DASDs.

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

Linux on System z includes the IBM label partitioning scheme that preserves the System z IPL, label, and VTOC records. With this partitioning scheme, Linux can share a disk with other mainframe operating systems. For example, a traditional mainframe operating system can handle backup and restore for a partition that is used by Linux.

The following sections describe the layouts that are supported by the IBM label partitioning scheme:
- “System z compatible disk layout” on page 147
- “Linux disk layout” on page 149
- “CMS disk layout” on page 150

DASD partitions

Partitioning DASDs has the same advantages as for other disk types, but there are some prerequisites and a special tool, **fdasd**.

A DASD partition is a contiguous set of DASD blocks that is treated by Linux as an independent disk and by the traditional mainframe operating systems as a data set.

With the Linux disk layout (LDL) and the CMS disk layout, you always have a single partition only. This partition is defined by the LDL or CMS formatted area of the disk. With the compatible disk layout, you can have up to three partitions.

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

Limit data growth
: Runaway processes or undisciplined users can consume disk space to an extend that the operating system runs short of space for essential operations. Partitions can help to isolate the space that is available to particular processes.

Encapsulate your data
: If a file system gets damaged, this damage is likely to be restricted to a single partition. Partitioning can reduce the scope of data damage.

Recommendations

- Use **fdasd** to create or alter partitions on ECKD type DASDs that are formatted with the compatible disk layout. If you use another partition editor, it is your responsibility to ensure that partitions do not overlap. If they do, data damage occurs.
- Leave no gaps between adjacent partitions to avoid wasting space. Gaps are not reported as errors, and can be reclaimed only by deleting and re-creating one or more of the surrounding partitions and rebuilding the file system on them.

A disk need not be partitioned completely. You can begin by creating only one or two partitions at the start of your disk and convert the remaining space to a partition later.

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

System z compatible disk layout

With the compatible disk layout, a DASD can have up to three partitions that can be accessed by traditional mainframe operating systems.

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

Figure 33 illustrates a DASD with the compatible disk layout.

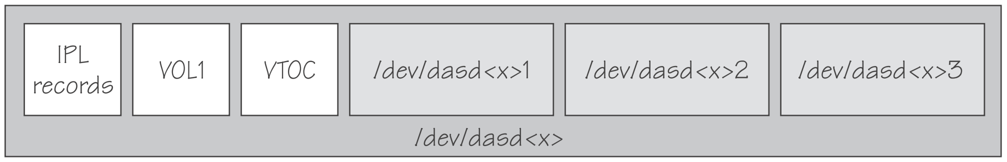


Figure 33. Compatible disk layout

The IPL records, volume label (VOL1), and VTOC of disks with the compatible disk layout are on the first two tracks of the disks. These tracks are not intended for use by Linux applications. Using the tracks can result in data loss.

Linux can address the device as a whole as `/dev/dasd<x>`, where `<x>` can be one to four letters that identify the individual DASD (see "DASD naming scheme" on page 151).

Disks with the compatible disk layout can have one to three partitions. Linux addresses the first partition as `/dev/dasd<x>1`, the second as `/dev/dasd<x>2`, and the third as `/dev/dasd<x>3`.

You use the **`dasdfmt`** command (see "dasdfmt - Format a DASD" on page 575) to format a disk with the compatible disk layout. You use the **`fdasd`** command (see "fdasd - Partition a DASD" on page 594) to create and modify partitions.

Volume label

The volume label includes information about the disk layout, the VOLSER, and a pointer to the VTOC.

The DASD volume label is in the third block of the first track of the device (cylinder 0, track 0, block 2). This block has a 4-byte key, and an 80-byte data area with the following content:

key
: for disks with the compatible disk layout, contains the four EBCDIC characters "VOL1" to identify the block as a volume label.

label identifier
: is identical to the key field.

VOLSER
: is a name that you can use to identify the DASD device. A volume serial number (VOLSER) can be one to six EBCDIC characters. If you want to use VOLSERs as identifiers for your DASD, be sure to assign unique VOLSERs.

You can assign VOLSERs from Linux by using the **`dasdfmt`** or **`fdasd`** command. These commands enforce that VOLSERs:

- Are alphanumeric
- Are uppercase (by uppercase conversion)
- Contain no embedded blanks
- Contain no special characters other than $, #, @, and %

Tip: Avoid special characters altogether.

Note: The VOLSER values `SCRTCH`, `PRIVAT`, `MIGRAT`, or `L`*`nnnnn`* (An "L" followed by 5 digits) are reserved for special purposes by other mainframe operating systems and should not be used by Linux.

These rules are more restrictive than the VOLSERs that are allowed by the traditional mainframe operating systems. For compatibility, Linux tolerates existing VOLSERs with lowercase letters and special characters other than $, #, @, and %. Enclose VOLSERs with special characters in single quotation marks if you must specify it, for example, as a command parameter.

VTOC address
: contains the address of a standard IBM format 4 data set control block (DSCB). The format is: *cylinder* (2 bytes) *track* (2 bytes) *block* (1 byte).

All other fields of the volume label contain EBCDIC space characters (code 0x40).

VTOC

Instead of a regular Linux partition table, Linux on System z, like other System z operating systems, uses a Volume Table Of Contents (VTOC).

The VTOC contains pointers to the location of every data set on the volume. In Linux on System z, these data sets form the Linux partitions.

The VTOC is on the second track (cylinder 0, track 1). It contains a number of records, each written in a separate data set control block (DSCB). The number of records depends on the size of the volume:

- One DSCB that describes the VTOC itself (format 4)
- One DSCB that is required by other operating systems but is not used by Linux. **fdasd** sets it to zeroes (format 5).
- For volumes with more than 65534 cylinders, 1 DSCB (format 7)
- For each partition:
  - On volumes with 65534 or less cylinders, one DSCB (format 1)
  - On volumes with more than 65534 cylinders, 1 format 8 and one format 9 DSCB

  The key of the format 1 or format 8 DSCB contains the data set name, which identifies the partition to z/OS, z/VM, and z/VSE.

The VTOC can be displayed with standard 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 example:

```
                            VM/DITTO DISPLAY VTOC                  LINE 1 OF 5
===>                                                          SCROLL ===> PAGE

CUU,193 ,VOLSER,LNX001   3390, WITH   100 CYLS, 15 TRKS/CYL, 58786 BYTES/TRK

--- FILE NAME --- (SORTED BY =,NAME  ,) ---- EXT    BEGIN-END     RELTRK,
1...5...10...15...20...25...30...35...40.... SQ   CYL-HD   CYL-HD      NUMTRKS
 *** VTOC EXTENT ***                          0     0  1     0  1      1,1
LINUX.VLNX001.PART0001.NATIVE                 0     0  2    46 11      2,700
LINUX.VLNX001.PART0002.NATIVE                 0    46 12    66 11    702,300
LINUX.VLNX001.PART0003.NATIVE                 0    66 12    99 14   1002,498
 *** THIS VOLUME IS CURRENTLY 100 PER CENT FULL WITH      0 TRACKS AVAILABLE

PF  1=HELP      2=TOP       3=END       4=BROWSE    5=BOTTOM    6=LOCATE
PF  7=UP        8=DOWN      9=PRINT    10=RGT/LEFT 11=UPDATE   12=RETRIEVE
```

The **ls** command on Linux might list this DASD and its partitions like this example:

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

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

Linux disk layout

The Linux disk layout does not have a VTOC, and DASD partitions that are formatted with this layout cannot be accessed by traditional mainframe operating systems.

You can format only ECKD type DASD with the Linux disk layout. Apart from accessing the disks as ECKD devices, you can also access them using the DASD DIAG access method. See "Enabling the DASD device driver to use the DIAG access method" on page 165 for how to enable DIAG.

Figure 34 illustrates a disk with the Linux disk layout.

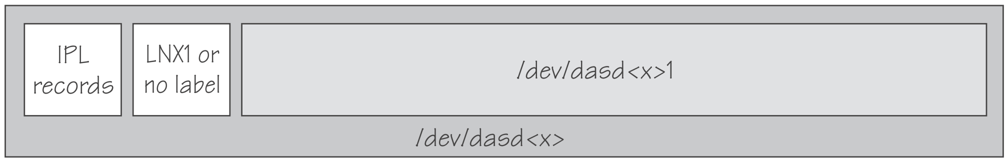


Figure 34. Linux disk layout

DASDs with the Linux disk layout either have an LNX1 label or are not labeled. The first records of the device are reserved for IPL records and the volume label, and are not intended for use by Linux applications. All remaining records are grouped into a single partition. You cannot have more than a single partition on a DASD that is formatted in the Linux disk layout.

Linux can address the device as a whole as `/dev/dasd<x>`, where *<x>* can be one to four letters that identify the individual DASD (see "DASD naming scheme" on page 151). Linux can access the partition as `/dev/dasd<x>1`.

You use the **`dasdfmt`** command (see "dasdfmt - Format a DASD" on page 575) to format a disk with the Linux disk layout.

CMS disk layout

The CMS disk layout applies only to Linux on z/VM. The disks are formatted with z/VM tools.

Both ECKD or FBA type DASD can have the CMS disk layout. DASD partitions that are formatted with this layout cannot be accessed by traditional mainframe operating systems. Apart from accessing the disks as ECKD or FBA devices, you can also access them using the DASD DIAG access method.

Figure 35 illustrates two variants of the CMS disk layout.

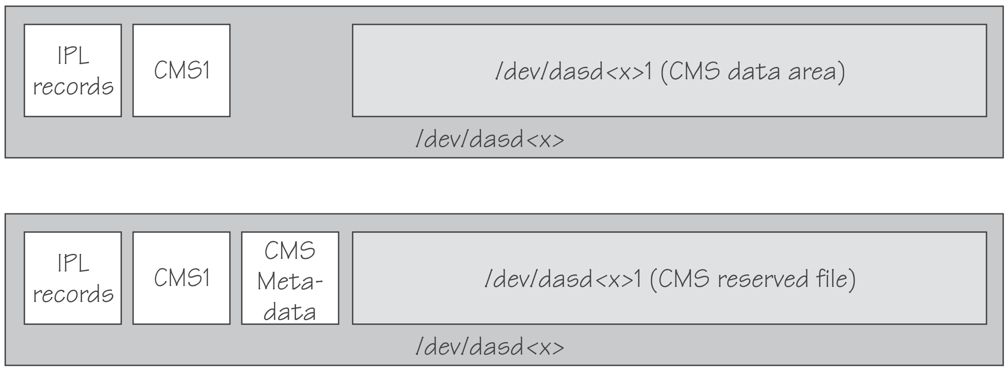


Figure 35. CMS disk layout

The first variant contains IPL records, a volume label (CMS1), and a CMS data area. Linux treats DASD like this equivalent to a DASD with the Linux disk layout, where the CMS data area serves as the Linux partition.

The second variant is a CMS reserved volume. In this variant, the DASD was reserved by a `CMS RESERVE fn ft fm` command. In addition to the IPL records and the volume label, DASD with the CMS disk layout also have CMS metadata. The CMS reserved file serves as the Linux partition.

For both variants of the CMS disk layout, you can have only a single Linux partition. The IPL record, volume label and (where applicable) the CMS metadata, are not intended for use by Linux applications.

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

"Enabling the DASD device driver to use the DIAG access method" on page 165 describes how to enable DIAG.

Disk layout summary

The available disk layouts differ in their support of device formats, the DASD DIAG access method, and the maximum number of partitions.

Table 18. Disk layout summary

Disk Layout	**ECKD device format**	**FBA device format**	**DIAG access method support (z/VM only)**	**Maximum number of partitions**	**Formatting tool**
CDL	Yes	No	No	3	**dasdfmt**
LDL	Yes	No	Yes	1	**dasdfmt**
CMS (z/VM only)	Yes	Yes	Yes	1	z/VM tools

DASD naming scheme

The DASD naming scheme maps device names and minor numbers to whole DASDs and to partitions.

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

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

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

The DASD device driver uses a device name of the form dasd<*x*> for each DASD. In the name, <*x*> is one to four lowercase letters. Table 19 on page 152 shows how the device names map to the available minor numbers.

Table 19. Mapping of DASD names to minor numbers

Name for device as a whole		**Minor number for device as a whole**		**Number of devices**
From	**To**	**From**	**To**	
dasda	dasdz	0	100	26
dasdaa	dasdzz	104	2804	676
dasdaaa	dasdzzz	2808	73108	17,576
dasdaaaa	dasdnwtl	73112	1048572	243,866
Total number of devices:				262,144

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

Examples

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

Creating device nodes

User space programs access DASDs by device nodes. Your distribution might create the device nodes for you, but you can also create your own device nodes.

About this task

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

Tip: Use the device names to construct your nodes (see "DASD naming scheme" on page 151).

Example

The following nodes use the form `/dev/<device_name>` for the device nodes. The assignment of minor numbers is according to Table 19.

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

Examples for udev-created DASD device nodes

The udev utility program can create device nodes that are based on device information from sysfs.

If your distribution provides udev, you can use udev to create DASD device nodes for you. Such device nodes typically include unique properties like device IDs or labels.

The DASD device driver assigns standard device names in the sequence in which DASDs are set online. To preserve the mapping between standard device nodes and DASDs across reboots, you must control the sequence in which the DASDs are set online. For example, use the `dasd=` kernel or module parameter.

Alternatively, you can use udev to create device nodes that are based on unique properties of a DASD and so identify a particular device. Such device nodes are independent of the sequence in which the devices are set online and can help you to reliably address an intended disk space.

For example, the device nodes can be based on the following information:
- The bus ID of the disk
- The disk label (VOLSER)
- 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.

Note: The format of the nodes that udev creates for you depends on distribution-specific configuration files. The device node descriptions that follow are according to generic rules as found in many distributions and in the `59-dasd.rules` file as shipped with the s390-tools package.

Device nodes that are based on bus IDs

udev creates device nodes of the form
`/dev/disk/by-path/ccw-<`*`device_bus_id`*`>`
for whole DASD and
`/dev/disk/by-path/ccw-<`*`device_bus_id`*`>-part<`*`n`*`>`
for the <*n*>th partition.

Device nodes that are based on VOLSERs

udev creates device nodes of the form
`/dev/disk/by-id/ccw-<`*`volser`*`>`
for whole DASD and
`/dev/disk/by-id/ccw-<`*`volser`*`>-part<`*`n`*`>`
for the <*n*>th partition.

If you want to use device nodes that are based on VOLSER, be sure that the VOLSERs in your environment are unique (see "Volume label" on page 148).

If you assign the same VOLSER to multiple devices, Linux can still access each device through its standard device node. However, only one of the devices can be accessed through the VOLSER-based device node. Thus, the node is ambiguous and might lead to unintentional data access.

Furthermore, if the VOLSER on the device that is addressed by the node is changed, the previously hidden device is not automatically addressed instead. To reassign the node, you must reboot Linux or force the kernel to reread the partition tables from disks, for example, by issuing:

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

You can assign VOLSERs to ECKD type devices with **dasdfmt** when formatting or later with **fdasd** when you are creating partitions.

Device nodes that are based on file system information
udev creates device nodes of the form
`/dev/disk/by-uuid/<uuid>`
where <*uuid*> is the UUID for the file system in a partition.

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

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

If you want to use device nodes that are based on file system labels, be sure that the labels in your environment are unique.

Additional device nodes
`/dev/disk/by-id` contains additional device nodes for the DASD and partitions, that are all based on a device identifier as contained in the `uid` attribute of the DASD.

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

Accessing DASD by udev-created device nodes

Use udev-created device nodes to access a particular physical disk space, regardless of the device name that is assigned to it.

Example

The following example is based on these assumptions:
- A DASD with bus ID 0.0.b100 has two partitions.
- The standard device node of the DASD is `dasdzzz`.
- udev creates the following device nodes for a DASD and its partitions:

```
/dev/disk/by-path/ccw-0.0.b100
/dev/disk/by-path/ccw-0.0.b100-part1
/dev/disk/by-path/ccw-0.0.b100-part2
```

Instead of issuing:

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

issue:

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

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

```
/dev/dasdzzz1 /temp1 ext3 defaults 0 0
/dev/dasdzzz2 /temp2 ext3 defaults 0 0
```

with these specifications:

```
/dev/disk/by-path/ccw-0.0.b100-part1 /temp1 ext3 defaults 0 0
/dev/disk/by-path/ccw-0.0.b100-part2 /temp2 ext3 defaults 0 0
```

You can make similar substitutions with other device nodes that udev provides for you (see “Examples for udev-created DASD device nodes” on page 153).

Building a kernel with the DASD device driver

Control the build options for the DASD device driver through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

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

Figure 36 gives an overview of the available DASD kernel configuration options and the corresponding modules.

```
Device Drivers --->
    ...
    Block devices --->                  (common code option CONFIG_BLK_DEV)
       ...
       --- S/390 block device drivers (depends on S390 && BLOCK) ---
       ...
       Support for DASD devices                         (CONFIG_DASD)
        ├─ Profiling support for dasd devices           (CONFIG_DASD_PROFILE)
        ├─ Support for ECKD Disks                       (CONFIG_DASD_ECKD)
        ├─ Support for FBA  Disks                       (CONFIG_DASD_FBA)
        ├─ Support for DIAG access to Disks             (CONFIG_DASD_DIAG)
        └─ Extended error reporting (EER)               (CONFIG_DASD_EER)
```

Figure 36. DASD kernel configuration menu options

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

  This option depends on CONFIG_CCW.

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

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

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

`CONFIG_DASD_DIAG`
This option provides support for accessing disks under z/VM with the Diagnose250 command. It can be compiled into the kernel or as a separate module, `dasd_diag_mod`. You must also enable the support for ECKD or FBA disks to get the device online.

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

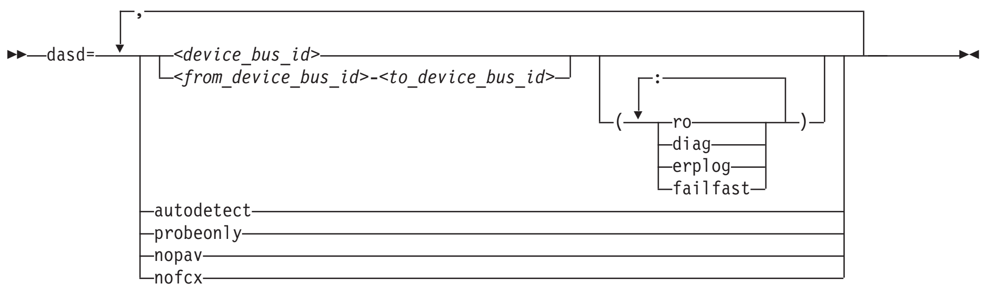
Setting up the DASD device driver

Configure the DASD device driver through the `dasd=` kernel or module parameter. You might also have to create a device node for the extended error reporting facility.

Kernel parameters

If the base module of the DASD device driver has been compiled into the kernel, you configure the device driver by adding parameters to the kernel parameter line.

DASD kernel parameter syntax



where:

autodetect
causes the DASD device driver to allocate device names and the corresponding minor numbers to all DASD devices and set them online during the boot process. See "DASD naming scheme" on page 151 for the naming scheme.

The device names are assigned in order of ascending subchannel numbers. Auto-detection can yield confusing results if you change your I/O configuration and reboot, or if your Linux instance runs as a z/VM guest because the devices might appear with different names and minor numbers after rebooting.

probeonly
causes the DASD device driver to reject any "open" syscall with `EPERM`.

autodetect,probeonly
causes the DASD device driver to assign device names and minor numbers

as for auto-detect. All devices regardless of whether they are accessible as DASD return `EPERM` to any “open” requests.

nopav suppresses parallel access volume (PAV and HyperPAV) enablement for Linux instances that run in LPAR mode. The **`nopav`** keyword has no effect for Linux on z/VM.

nofcx suppresses accessing the storage server with the I/O subsystem in transport mode (also known as High Performance FICON).

<device_bus_id>
specifies a single DASD.

<from_device_bus_id>-<to_device_bus_id>
specifies the first and last DASD in a range. All DASD devices with bus IDs in the range are selected. The device bus-IDs *<from_device_bus_id>* and *<to_device_bus_id>* need not correspond to actual DASD.

(ro) accesses the specified device or device range in read-only mode.

(diag) forces the device driver to access the device (range) with the DIAG access method.

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

(failfast)
immediately returns “failed” for an I/O operation when the last path to a DASD is lost.

Attention: Enable immediate failure of I/O requests only in setups where a failed I/O request can be recovered outside the scope of a single DASD (see “Enabling and disabling immediate failure of I/O requests” on page 170).

If you supply a DASD kernel parameter with device specifications `dasd=`*`<device-list1>`*`,`*`<device-list2>`*` ...`, the device names and minor numbers are assigned in the order in which the devices are specified. The names and corresponding minor numbers are always assigned, even if the device is not present, or not accessible.

If you use **`autodetect`** in addition to explicit device specifications, device names are assigned to the specified devices first and device-specific parameters, like **`ro`**, are observed. The remaining devices are handled as described for **`autodetect`**.

Example

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

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

Table 20 shows the resulting allocation of device names and minor numbers:

Table 20. Example mapping of device names and minor numbers to devices

Minor	Name	To access
0	dasda	device 0.0.7000 as a whole
1	dasda1	the first partition on 0.0.7000

Table 20. Example mapping of device names and minor numbers to devices (continued)

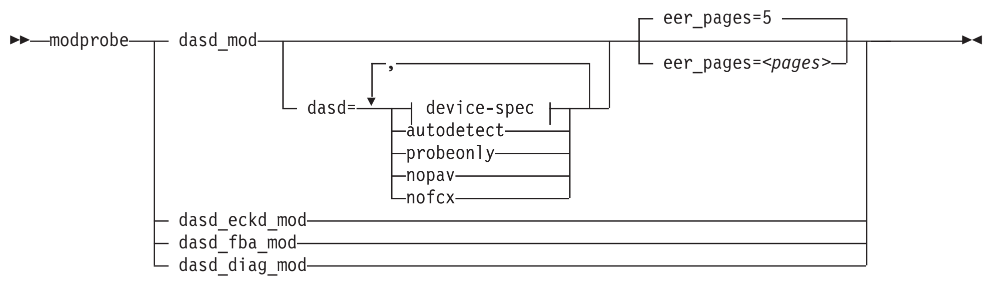
Minor	Name	To access
2	dasda2	the second partition on 0.0.7000
3	dasda3	the third partition on 0.0.7000
4	dasdb	device 0.0.7001 as a whole
5	dasdb1	the first partition on 0.0.7001
6	dasdb2	the second partition on 0.0.7001
7	dasdb3	the third partition on 0.0.7001
8	dasdc	device 0.0.7002 as a whole
9	dasdc1	the first partition on 0.0.7002
10	dasdc2	the second partition on 0.0.7002
11	dasdc3	the third partition on 0.0.7002
12	dasdd	device 0.0.7005 as a whole
13	dasdd1	the first partition on 0.0.7005 (read-only)
14	dasdd2	the second partition on 0.0.7005 (read-only)
15	dasdd3	the third partition on 0.0.7005 (read-only)
16	dasde	device 0.0.7006 as a whole
17	dasde1	the first partition on 0.0.7006
18	dasde2	the second partition on 0.0.7006
19	dasde3	the third partition on 0.0.7006

Module parameters

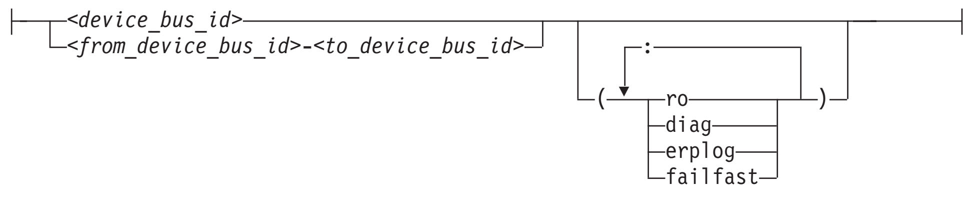
If the base module of the DASD device driver has been built as a separate module, you configure the device driver through module parameters. You specify these parameters when you are loading the base module.

Any components of the DASD device driver that have been compiled as separate modules must be loaded before they can be used.

DASD module parameter syntax



device-spec:



Where:

dasd_mod
: loads the device driver base module.

  When you are loading the base module, you can specify the `dasd=` parameter. The variables and key words have the same meaning as the corresponding kernel parameters(see "Kernel parameters" on page 156).

  If the extended error reporting feature is compiled into this module (see "Building a kernel with the DASD device driver" on page 155), you can use the `eer_pages` parameter to determine the number of pages that are used for internal buffering of error records.

dasd_eckd_mod
: loads the ECKD module.

dasd_fba_mod
: loads the FBA module.

dasd_diag_mod
: loads the DIAG module.

The DASD base component is required by the other modules. **modprobe** takes care of this dependency for you and ensures that the base module is loaded automatically, if necessary.

Hint: **modprobe** might return before udev has created all device nodes for the specified DASDs. If you need to assure that all nodes are present, for example in scripts, follow the **modprobe** command with:

```
# udevadm settle
```

For command details see the **modprobe** man page.

Examples

The following example specifies a range of DASD devices and two individual DASD devices:

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

For the same mainframe setup, the resulting allocation of device nodes and minor numbers would be the same as in Table 20 on page 157.

The following example specifies that High Performance FICON are to be suppressed for all DASDs:

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

Assuring that a device node exists for extended error reporting

Applications that use the extended error reporting facility require a character device to access the extended error data.

Before you begin

- You need the device node only if you want to support applications that use the extended error reporting for ECKD type DASD.
- You need a DASD device driver that has been built with support for extended error reporting (see "Building a kernel with the DASD device driver" on page 155)
- Your distribution might create the required device node for you, for example, with udev.

  To check if there is already a node issue:

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

  Typically, the node is called `/dev/dasd_eer`.

Procedure

If there is no device node, perform the following steps to create one:

1. Find out the major and minor number for your monitor device by reading the `dev` attribute of the device representation in sysfs:

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

   The value of the `dev` attribute is of the form <*major*>:<*minor*>.
2. Create the device node by issuing a command of the form:

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

   where <*node*> is your device node.

Example

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

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

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

Working with DASDs

You might have to prepare DASDs for use, configure troubleshooting functions, or configure special device features for your DASDs.

See "Working with newly available devices" on page 12 to avoid errors when you are working with devices that have become available to a running Linux instance.

- "Preparing an ECKD type DASD for use"
- "Preparing an FBA-type DASD for use" on page 163
- "Accessing DASD by force" on page 164
- "Enabling the DASD device driver to use the DIAG access method" on page 165
- "Using extended error reporting for ECKD type DASD" on page 167
- "Setting a DASD online or offline" on page 168
- "Enabling and disabling logging" on page 169
- "Enabling and disabling immediate failure of I/O requests" on page 170
- "Setting the timeout for I/O requests" on page 170
- "Working with DASD statistics in debugfs" on page 172
- "Accessing full ECKD tracks" on page 176
- "Handling lost device reservations" on page 178
- "Reading and resetting the reservation state" on page 179
- "Displaying DASD information" on page 180

Preparing an ECKD type DASD for use

Before you can use an ECKD type DASD as a Linux on System z disk, you must format it with a suitable disk layout. You must then create a file system or define a swap space.

Before you begin

- The base component and the ECKD component of the DASD device driver must have been compiled into the kernel or have been loaded as modules.
- The DASD device driver must have recognized the device as an ECKD type device.
- You need to know the device bus-ID for your DASD.

About this task

If you format the DASD with the compatible disk layout, you must create one, two, or three partitions. You can then use your partitions as swap areas or to create a Linux file system.

Procedure

Perform these steps to prepare the DASD:

1. Issue **lsdasd** (see "lsdasd - List DASD devices" on page 621) to find out if the device is online. If necessary, set the device online using **chccwdev** (see "chccwdev - Set CCW device attributes" on page 543).

   Example:

   ```
   # chccwdev -e 0.0.b100
   ```

2. Format the device with the **dasdfmt** command (see "dasdfmt - Format a DASD" on page 575 for details). The formatting process can take hours for large DASDs. If you want to use the CMS disk layout, and your DASD is already formatted with the CMS disk layout, skip this step.

   Tips:
   - Use the largest possible block size, ideally 4096; the net capacity of an ECKD DASD decreases for smaller block sizes. For example, a DASD formatted with a block size of 512 byte has only half of the net capacity of the same DASD formatted with a block size of 4096 byte.
   - Use the **-p** option to display a progress bar.

   Example: Assuming that /dev/dasdzzz is a valid device node for 0.0.b100:

   ```
   # dasdfmt -b 4096 -p /dev/dasdzzz
   ```

3. Proceed according to your chosen disk layout:
   - If you have formatted your DASD with the Linux disk layout or the CMS disk layout, skip this step and continue with step 4. 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 594 for details).

     Example: To start the partitioning tool in interactive mode for partitioning a device /dev/dasdzzz issue:

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

     If you create three partitions for a DASD /dev/dasdzzz, the device nodes for the partitions are /dev/dasdzzz1, /dev/dasdzzz2, and /dev/dasdzzz3.

     Result: **fdasd** creates the partitions and updates the partition table (see "VTOC" on page 149).

4. Depending on the intended use of each partition, create a file system on the partition or define it as a swap space.
   - Either create a file system of your choice, for example, with the Linux **mke2fs** command (see the man page for details).

     Note: Do not make the block size of the file system smaller than the block size that was used for formatting the disk with the **dasdfmt** command.

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

Example:

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

- Or define the partition as a swap space with the **mkswap** command (see the man page for details).

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

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

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

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

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

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

Preparing an FBA-type DASD for use

Before you can use an FBA-type DASD as a Linux on System z disk, you must create a file system or define a swap space.

Before you begin

- The base component and the FBA component of the DASD device driver must have been compiled into the kernel or have been loaded as modules.
- The DASD device driver must have recognized the device as an FBA device.
- You must know the device bus-ID or the device node through which the DASD can be addressed. The DASD device nodes have the form `/dev/dasd<x>`, where *<x>* can be one to four lowercase alphabetic characters.

Procedure

Perform these steps to prepare the DASD:

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

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

   ```
   # ls dasdzzy*
   ```

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

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

See Table 19 on page 152 for the mapping of device names and minor numbers.

2. Depending on the intended use of the partition, create a file system on it or define it as a swap space.
   - Either create a file system of your choice, for example, with the Linux **mke2fs** command (see the man page for details).

     Example:

     ```
     # mke2fs -b 4096 /dev/dasdzzy1
     ```

   - Or define the partition as a swap space with the **mkswap** command (see the man page for details).
3. Mount the file system to the mount point of your choice in Linux or enable your swap partition.

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

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

What to do next

To access FBA devices, use the DIAG access method (see "Enabling the DASD device driver to use the DIAG access method" on page 165 for more information).

Accessing DASD by force

A Linux instance can encounter DASDs that are locked by another system.

Such a DASD is referred to as "externally locked" or "boxed". The Linux instance cannot analyze a DASD while it is externally locked.

About this task

To check whether a DASD has been externally locked, read its availability attribute. This attribute should be "good". If it is "boxed", the DASD has been externally locked. Because a boxed DASD might not be recognized as DASD, it might not show up in the device driver view in sysfs. If necessary, use the device category view instead (see "Device views in sysfs" on page 13).

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

Procedure

1. Optional: To read the availability attribute of a DASD, issue a command of this form:

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

Example: This example shows that a DASD with device bus-ID 0.0.b110 (device number 0xb110) has been externally locked.

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

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

2. To force a boxed DASD online, write `force` to the online device attribute. Issue a command of this form:

```
# echo force > /sys/bus/ccw/devices/<device_bus_id>/online
```

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

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

Results

If the external lock is successfully broken or if the lock has been surrendered by the time the command is processed, the device is analyzed and set online. If it is not possible to break the external lock (for example, because of a timeout, or because it is an FBA-type DASD), the device remains in the boxed state. This command might take some time to complete.

For information about breaking the lock of a DASD that has already been analyzed see “tunedasd - Adjust low-level DASD settings” on page 680.

Enabling the DASD device driver to use the DIAG access method

Linux on z/VM can use the DIAG access method to access DASDs with the help of z/VM functions.

Before you begin

This section applies only to Linux instances and DASDs for which all of the following conditions are true:

- The Linux instance runs as a z/VM guest.
- The Linux instance has kernel that has been compiled with the CONFIG_DASD_DIAG option (see “Building a kernel with the DASD device driver” on page 155).
- The device can be of type ECKD with either LDL or CMS disk layout, or it can be a device of type FBA.
- The DIAG component (dasd_diag_mod) must be loaded or compiled into the kernel.
- The component that corresponds to the DASD type (dasd_eckd_mod or dasd_fba_mod) must be loaded or compiled into the kernel.
- The DASD is offline.
- The DASD does not represent a parallel access volume alias device.

About this task

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

Alternatively, you can specify `diag` on the command line, for example during IPL, to force the device driver to access the device (range) with the DIAG access method.

Procedure

Issue a command of this form:

```
# echo <flag> > /sys/bus/ccw/devices/<device_bus_id>/use_diag
```

where *<device_bus_id>* identifies the DASD.
If the DIAG access method is not available and you set the `use_diag` attribute to `1`, you cannot set the device online (see "Setting a DASD online or offline" on page 168).

Note: When switching between an enabled and a disabled DIAG access method on FBA-type DASD, first reinitialize the DASD, for example, with CMS format or by overwriting any previous content. Switching without initialization might cause data-integrity problems.
For more details about DIAG, see *z/VM CP Programming Services*, SC24-6179.

Example

In this example, the DIAG access method is enabled for a DASD with device number 0xb100.

1. Ensure that the driver is loaded (only applicable when compiled as module):

   ```
   # modprobe dasd_diag_mod
   ```

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

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

3. Ensure that the device is offline:

   ```
   # echo 0 > online
   ```

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

   ```
   # echo 1 > use_diag
   ```

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

   ```
   # echo 1 > online
   ```

Using extended error reporting for ECKD type DASD

Control the extended error reporting feature for individual ECKD type DASD through the `eer_enabled` sysfs attribute. Use the character device of the extended error reporting module to obtain error records.

Before you begin

To use the extended error reporting feature, you need:

- A kernel that includes the extended error reporting feature, either compiled into the kernel or as a separate module (see "Building a kernel with the DASD device driver" on page 155.
- A misc character device (see "Assuring that a device node exists for extended error reporting" on page 160).
- ECKD type DASD

About this task

The extended error reporting feature is disabled by default.

Procedure

To enable extended error reporting, issue a command of this form:

```
# echo 1 > /sys/bus/ccw/devices/<device_bus_id>/eer_enabled
```

where `/sys/bus/ccw/devices/<device_bus_id>` represents the device in sysfs.
When it is enabled on a device, a specific set of errors generates records and might have further side effects.
To disable extended error reporting, issue a command of this form:

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

What to do next

You can obtain error records for all DASD for which extended error reporting is enabled from the character device of the extended error reporting module (see "Assuring that a device node exists for extended error reporting" on page 160). The device supports these file operations:

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

`close`
You can close the node as usual.

`read`
Blocking read and non-blocking read are supported. When a record is partially read and then purged, the next read returns an I/O error -EIO.

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

Setting a DASD online or offline

Use the **chccwdev** command or the `online` sysfs attribute of the device to set DASDs online or offline.

About this task

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

When you set a DASD offline, the deregistration process is synchronous, unless the device is disconnected. For disconnected devices, the deregistration process is asynchronous.

Procedure

Use the **chccwdev** command ("chccwdev - Set CCW device attributes" on page 543) to set a DASD online or offline.
Alternatively, you can write 1 to the device's sysfs online attribute to set it online or `0` to set it offline. In contrast to the sysfs attribute, the **chccwdev** command triggers a cio_settle for you and waits for the cio_settle to complete.
Outstanding I/O requests are cancelled when you set a device offline. To wait indefinitely for outstanding I/O requests to complete before setting the device offline, use the **chccwdev** option `--safeoffline` or the sysfs attribute `safe_offline`.

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
  ```

- To complete outstanding I/O requests and then set a DASD with device bus-ID 0.0.4711 offline, issue:

  ```
  # chccwdev -s 0.0.4711
  ```

  or

  ```
  # echo 1 > /sys/bus/ccw/devices/0.0.4711/safe_offline
  ```

  If an outstanding I/O request is blocked, the command might wait forever. Reasons for blocked I/O requests include reserved devices that can be released or disconnected devices that can be reconnected.

1. Try to resolve the problem that blocks the I/O request and wait for the command to complete.
2. If you cannot resolve the problem, issue **chccwdev -d** to cancel the outstanding I/O requests. The data will be lost.

Dynamic attach and detach

You can dynamically attach devices to a running Linux on System z instance, for example, from z/VM.

When a DASD is attached, Linux attempts to initialize it according to the DASD device driver configuration (see "Kernel parameters" on page 156). 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 21).

Attention: Do not detach a device that is still being used by Linux. Detaching devices might cause the system to hang or crash. Ensure that you unmount a device and set it offline before you detach it.

See "Working with newly available devices" on page 12 to avoid errors when working with devices that have become available to a running Linux instance.

Be careful to avoid errors when working with devices that have become available to a running Linux instance.

Enabling and disabling logging

Use the `dasd=` kernel or module parameter or use the `erplog` sysfs attribute to enable or disable error recovery processing (ERP) logging.

Procedure

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

- Use the `dasd=` parameter when you load the base module of the DASD device driver.

  Example:

  To define a device range (0.0.7000-0.0.7005) and enable logging, change the parameter line to contain:

  ```
  dasd=0.0.7000-0.0.7005(erplog)
  ```
- Use the sysfs attribute `erplog` to turn ERP-related logging on or off.

  Logging can be enabled for a specific device by writing `1` to the `erplog` attribute.

  Example:

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

  To disable logging, write `0` to the `erplog` attribute.

  Example:

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

Enabling and disabling immediate failure of I/O requests

Prevent devices in mirror setups from being blocked while paths are unavailable by making I/O requests fail immediately.

About this task

By default, if all path have been lost for a DASD, the corresponding device in Linux waits for one of the paths to recover. I/O requests are blocked while the device is waiting.

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

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

Procedure

Use one of these methods:

- You can enable immediate failure of I/O requests when you load the base module of the DASD device driver.

  Example:

  To define a device range (0.0.7000-0.0.7005) and enable immediate failure of I/O requests specify:

  ```
  dasd=0.0.7000-0.0.7005(failfast)
  ```
- You can use the sysfs attribute `failfast` of a DASD to enable or disable immediate failure of I/O requests.

  To enable immediate failure of I/O requests, write `1` to the `failfast` attribute.

  Example:

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

  To disable immediate failure of I/O requests, write `0` to the `failfast` attribute.

  Example:

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

Setting the timeout for I/O requests

DASD I/O requests can time out at two levels in the software stack.

About this task

When the DASD device driver receives an I/O request from an application, it issues one or more low-level I/O requests to the affected storage system. Both the initial I/O request from the application and the resulting low-level requests to the storage system can time out. You set the timeout values through two sysfs attributes of the DASD.

expires

specifies the maximum time, in seconds, that the DASD device driver waits for a response to a low-level I/O request from a storage server.

The default for the maximum response time depends on the type of DASD:

ECKD	uses the default that is provided by the storage server.
FBA	300 s
DIAG	50 s

If the maximum response time is exceeded, the DASD device driver cancels the request. Depending on your setup, the DASD device driver might then try the request again, possibly in combination with other recovery actions.

timeout

specifies the time interval, in seconds, within which the DASD device driver must respond to an I/O request from a software layer above it. If the specified time expires before the request is completed, the DASD device driver cancels all related low-level I/O requests to storage systems and reports the request as failed.

This setting is useful in setups where the software layer above the DASD device driver requires an absolute upper limit for I/O requests.

A value of `0` means that there is no time limit. This value is the default.

Procedure

You can use the `expires` and `timeout` attributes of a DASD to change the timeout values for that DASD.

1. To find out the current timeout values, issue commands of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/expires
# cat /sys/bus/ccw/devices/<device_bus_id>/timeout
```

Example:

```
# cat /sys/bus/ccw/devices/0.0.7008/expires
30
# cat /sys/bus/ccw/devices/0.0.7008/timeout
0
```

In the example, a maximum response time of 30 seconds applies to the storage server for a DASD with bus ID `0.0.7008`. No total time limit is set for I/O requests to this DASD.

2. To set different timeout values, issue commands of this form:

```
# echo <max_wait> > /sys/bus/ccw/devices/<device_bus_id>
# echo <total_max> > /sys/bus/ccw/devices/<device_bus_id>/timeout
/expires
```

where:

<max_wait>
is the new maximum response time, in seconds, for the storage server. The value must be a positive integer.

<total_max>
is the new maximum total time in seconds. The value must be a positive integer or `0`. `0` disables this timeout setting.

<device_bus_id>
is the device bus-ID of the DASD.

Example:

```
# echo 60 > /sys/bus/ccw/devices/0.0.7008/expires
# echo 120 > /sys/bus/ccw/devices/0.0.7008/timeout
```

This example sets timeout values for a DASD with bus ID `0.0.7008`. The maximum response time for the storage server is set to 60 seconds and the overall time limit for I/O requests is set to 120 seconds.

Working with DASD statistics in debugfs

Gather DASD statistics and display the data with the **dasdstat** command.

Before you begin

- Your DASD device driver must have been compiled with profiling support (see “Building a kernel with the DASD device driver” on page 155).
- You need a kernel that supports debugfs and debugfs must be mounted.
- Instead of accessing raw DASD performance data in debugfs, you can use the **dasdstat** command to obtain more structured data (see “dasdstat - Display DASD performance statistics” on page 579).

About this task

The DASD performance data is contained in the following subdirectories of *<mountpoint>*`/dasd`, where *<mountpoint>* is the mount point of debugfs:

- A directory `global` that represents all available DASDs taken together.
- For each DASD, one directory with the name of the DASD block device with which the DASD is known to the DASD device driver (for example, `dasda`, `dasdb`, and `dasdc`).
- For each CCW device that corresponds to a DASD, a directory with the bus ID as the name.

  Block devices that are not set up for PAV or HyperPAV map to exactly one CCW device and the corresponding directories contain the same statistics.

  With PAV or HyperPAV, a bus ID can represent a base device or an alias device. Each base device is associated with a particular block device. The alias devices are not permanently associated with the same block device. At any one time, a DASD block device is associated with one or more CCW devices. Statistics that are based on bus ID, therefore, show more detail for PAV and HyperPAV setups.

Each of these directories contains a file `statistics` that you can use to perform these tasks:

- Start and stop data gathering.
- Reset statistics counters.
- Read statistics.

To control data gathering at the scope of a directory in *<mountpoint>*`/dasd`, issue a command of this form:

```
# echo <keyword> > <mountpoint>/dasd/<directory>/statistics
```

Where:

<directory>
is one of the directories in *<mountpoint>*/dasd.

<keyword>
specifies the action to be taken:

  on to start data gathering.

  off
  to stop data gathering.

  reset
  to reset the statistics counters.

To read performance data, issue a command of this form:

```
# cat <mountpoint>/dasd/<directory>/statistics
```

Examples for gathering and reading DASD statistics in debugfs

Use the **echo** command to start and stop data gathering for individual devices or across all DASDs. Use the **cat** command to access the raw performance data.

The following examples assume that debugfs is mounted at /sys/kernel/debug.

- To start data gathering for summary data across all available DASDs:

  ```
  # echo on > /sys/kernel/debug/dasd/global/statistics
  ```

- To stop data gathering for block device dasdb:

  ```
  # echo off > /sys/kernel/debug/dasd/dasdb/statistics
  ```

- To reset the counters for CCW device 0.0.b301:

  ```
  # echo reset > /sys/kernel/debug/dasd/0.0.b301/statistics
  ```

- To read performance data for dasda, assuming that the degbugfs mount point is /sys/kernel/debug, issue:

```
# cat  /sys/kernel/debug/dasd/dasda/statistics
start_time 1283518578.085869197
total_requests 0
total_sectors 0
total_pav 0
total_hpf 0
histogram_sectors 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_io_times 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_io_times_weighted 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_time_build_to_ssch 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_time_ssch_to_irq 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_time_ssch_to_irq_weighted 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_time_irq_to_end 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_ccw_queue_length 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
total_read_requests 0
total_read_sectors 0
total_read_pav 0
total_read_hpf 0
histogram_read_sectors 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_read_times 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_read_time_build_to_ssch 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_read_time_ssch_to_irq 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_read_time_irq_to_end 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
histogram_read_ccw_queue_length 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Interpreting the data rows

The raw DASD performance data in the `statistics` directories in debugfs is organized into labeled data rows.

This section explains the raw data in the individual data rows of the statistics. Use the **`dasdstat`** command to obtain more structured data.

start_time
: is the UNIX epoch time stamp when data gathering was started or when the counters were last reset.

  Tip: Use the **`date`** tool to convert the time stamp to a more readily human-readable format. See the **`date`** man page for details.

Single counters
: have a single integer as the statistics data. All rows with labels that begin with `total_` are of this data type.

  The following rows show data for the sum of all requests, read and write:

  total_requests
  : is the number of requests that have been processed.

  total_sectors
  : is the sum of the sizes of all requests, in units of 512-byte sectors.

  total_pav
  : is the number of requests that were processed through a PAV alias device.

  total_hpf
  : is the number of requests that used High Performance FICON.

  The following rows show data for read requests only:

  total_read_requests
  : is the number of read requests that have been processed.

  total_read_sectors
  : is the sum of the sizes of all read requests, in units of 512-byte sectors.

total_read_pav
is the number of read requests that were processed through a PAV alias device.

total_read_hpf
is the number of read requests that used High Performance FICON.

Linear histograms
have a series of 32 integers as the statistics data. The integers represent a histogram, with a linear scale, of the number of requests in the request queue each time a request has been queued. The first integer shows how often the request queue contained zero requests, the second integer shows how often the queue contained one request, and the n-th value shows how often the queue contained n-1 requests.

histogram_ccw_queue_length
is the histogram data for all requests, read and write.

histogram_read_ccw_queue_length
is the histogram data for read requests only.

Logarithmic histograms
have a series of 32 integers as the statistics data. The integers represent a histogram with a logarithmic scale:

- The first integer always represents all measures of fewer than 4 units
- The second integer represents measures of 4 or more but less than 8 units
- The third integer represents measures of 8 or more but less than 16 units
- The n-th integer ($1 < n < 32$) represents measures of 2^n or more but less than 2^{n+1} units
- The 32nd integer represents measures of 2^{32} (= 4G = 4,294,967,296) units or more.

The following rows show data for the sum of all requests, read and write:

histogram_sectors
is the histogram data for request sizes. A unit is a 512-byte sector.

histogram_io_times
is the histogram data for the total time that is needed from creating the cqr to its completion in the DASD device driver and return to the block layer. A unit is a microsecond.

histogram_io_times_weighted
is the histogram data of the total time, as measured for `histogram_io_times`, devided by the requests size in sectors. A unit is a microsecond per sector.

This metric is deprecated and there is no corresponding histogram data for read requests.

histogram_time_build_to_ssch
is the histogram data of the time that is needed from creating the cqr to submitting the request to the subchannel. A unit is a microsecond.

histogram_time_ssch_to_irq
is the histogram data of the time that is needed from submitting the request to the subchannel until an interrupt indicates that the request has been completed. A unit is a microsecond.

histogram_time_ssch_to_irq_weighted
: is the histogram data of the time that is needed from submitting the request to the subchannel until an interrupt indicates that the request has been completed, divided by the request size in 512-byte sectors. A unit is a microsecond per sector.

  This metric is deprecated and there is no corresponding histogram data for read requests.

histogram_time_irq_to_end
: is the histogram data of the time that is needed from return of the request from the channel subsystem, until the request is returned to the block layer. A unit is a microsecond.

The following rows show data for read requests only:

histogram_read_sectors
: is the histogram data for read request sizes. A unit is a 512 byte sector.

histogram_read_io_times
: is the histogram data, for read requests, for the total time needed from creating the cqr to its completion in the DASD device driver and return to the block layer. A unit is a microsecond.

histogram_read_time_build_to_ssch
: is the histogram data, for read requests, of the time needed from creating the cqr to submitting the request to the subchannel. A unit is a microsecond.

histogram_read_time_ssch_to_irq
: is the histogram data, for read requests, of the time needed from submitting the request to the subchannel until an interrupt indicates that the request has been completed. A unit is a microsecond.

histogram_read_time_irq_to_end
: is the histogram data, for read requests, of the time needed from return of the request from the channel subsystem, until the request is returned to the blocklayer. A unit is a microsecond.

Accessing full ECKD tracks

In raw-track access mode, the DASD device driver accesses full ECKD tracks, including record zero and the count and key data fields.

Before you begin

- This section applies to ECKD type DASD only.
- The DASD has to be offline when you change the access mode.
- The DIAG access method must not be enabled for the device.

About this task

With this mode, Linux can access an ECKD device regardless of the track layout. In particular, the device does not need to be formatted for Linux.

For example, with raw-track access mode Linux can create a backup copy of any ECKD device. Full-track access can also enable a special program that runs on Linux to access and process data on an ECKD device that is not formatted for Linux.

By default, the DASD device driver accesses only the data fields of ECKD devices. In default access mode, you can work with partitions, file systems, and files in the file systems on the DASD.

When using a DASD in raw-track access mode be aware that:

- In memory, each track is represented by 64 KB of data, even if the track occupies less physical disk space. Therefore, a disk in raw-track access mode appears bigger than in default mode.
- Programs must write and should read data in multiples of complete 64 KB tracks. Read requests for less than 64 KB are allowed, but are not optimal as the DASD device driver always reads full tracks. The minimum is a single track. The maximum is eight tracks by default but can be extended to up to 16 tracks.

  The maximum number of tracks depends on the maximum number of sectors as specified in the `max_sectors_kb` sysfs attribute of the DASD. This attribute is located in the block device branch of sysfs at `/sys/block/dasd<x>/queue/max_sectors_kb`. In the path, `dasd<x>` is the device name that is assigned by the DASD device driver.

  To extend the maximum beyond eight tracks, set the `max_sectors_kb` to the maximum amount of data to be processed in a single read or write operation. For example, to extend the maximum to reading or writing 16 tracks at a time, set `max_sectors_kb` to `1024` (16 x 64).
- Programs must write only valid ECKD tracks of 64 KB.
- Programs must use direct I/O to prevent the Linux block layer from splitting tracks into fragments. The DASD device driver must read a split track multiple times, which might slow down the reading process. Open the block device with option O_DIRECT or work with programs that use direct I/O.

  For example, the options `iflag=direct` and `oflag=direct` cause **dd** to use direct I/O. When using **dd**, also specify the block size with the `bs=` option. The block size determines the number of tracks that are processed in a single I/O operation. The block size must be a multiple of 64 KB and can be up to 1024 KB. Specifying a larger block size often results in better performance. If you receive disk image data from a pipe, also use the option `iflag=fullblock` to ensure that full blocks are written to the DASD device.

  Tools cannot directly work with partitions, file systems, or files within a file system. For example, **fdasd** and **dasdfmt** cannot be used.

Procedure

To change the access mode, issue a command of this form:

```
# echo <switch> > /sys/bus/ccw/devices/<device_bus_id>/raw_track_access
```

where:

<switch>
  is 1 to activate raw data access and 0 to deactivate raw data access.

<device_bus_id>
  identifies the DASD.

Example

The following example creates a backup of a DASD `0.0.7009` on a DASD `0.0.70a1`.

The initial commands ensure that both devices are offline and that the DIAG access method is not enabled for either of them. The subsequent commands activate the raw-track access mode for the two devices and set them both online. The **lsdasd** command that follows shows the mapping between device bus-IDs and device names.

The **dd** command for the copy operation specifies direct I/O for both the input and output device and the block size of 1024 KB. After the copy operation is completed, both devices are set offline. The access mode for the original device is then set back to the default and the device is set back online.

```
#cat /sys/bus/ccw/devices/0.0.7009/online
1
# chccwdev -d 0.0.7009
# cat /sys/bus/ccw/devices/0.0.7009/use_diag
0
# cat /sys/bus/ccw/devices/0.0.70a1/online
0
# cat /sys/bus/ccw/devices/0.0.70a1/use_diag
0
# echo 1 > /sys/bus/ccw/devices/0.0.7009/raw_track_access
# echo 1 > /sys/bus/ccw/devices/0.0.70a1/raw_track_access
# chccwdev -e 0.0.7009,0.0.70a1
# lsdasd 0.0.7009 0.0.70a1
Bus-ID     Status      Name      Device  Type  BlkSz  Size      Blocks
==============================================================================
0.0.7009   active      dasdf     94:20   ECKD  4096   7043MB    1803060
0.0.70a1   active      dasdj     94:36   ECKD  4096   7043MB    1803060
# echo 1024 > /sys/block/dasdf/queue/max_sectors_kb
# echo 1024 > /sys/block/dasdj/queue/max_sectors_kb
# dd if=/dev/dasdf of=/dev/dasdj bs=1024k iflag=direct oflag=direct
# chccwdev -d 0.0.7009,0.0.70a1
# echo 0 > /sys/bus/ccw/devices/0.0.7009/raw_track_access
# chccwdev -e 0.0.7009
```

Handling lost device reservations

A DASD reservation by your Linux instance can be lost if another system unconditionally reserves this DASD.

About this task

This other system then has exclusive I/O access to the DASD for the duration of the unconditional reservation. Such unconditional reservations can be useful for handling error situations where:

- Your Linux instance cannot gracefully release the DASD.
- Another system requires access to the DASD, for example, to perform recovery actions.

After the DASD is released by the other system, your Linux instance might process pending I/O requests and write faulty data to the DASD. How to prevent pending I/O requests from being processed depends on the reservation policy. There are two reservation policies:

ignore All I/O operations for the DASD are blocked until the DASD is released by the second system. When using this policy, reboot your Linux instance before the other system releases the DASD. This policy is the default.

fail All I/O operations are returned as failed until the DASD is set offline or until the reservation state is reset. When using this policy, set the DASD offline and back online after the problem is resolved. See "Reading and resetting the reservation state" about resetting the reservation state to resume operations.

Procedure

Set the reservation policy with a command of this form:

```
# echo <policy> > /sys/bus/ccw/devices/<device_bus_id>/reservation_policy
```

where:

<device_bus_id>
: specifies the DASD.

<policy>
: is one of the available policies, `ignore` or `fail`.

Examples

- The command of this example sets the reservation policy for a DASD with bus ID `0.0.7009` to `fail`.

```
# echo fail > /sys/bus/ccw/devices/0.0.7009/reservation_policy
```

- This example shows a small scenario. The first two commands confirm that the reservation policy of the DASD is `fail` and that the reservation has been lost to another system. Assuming that the error that had occurred has already been resolved and that the other system has released the DASD, operations with the DASD are resumed by setting it offline and back online.

```
# cat /sys/bus/ccw/devices/0.0.7009/reservation_policy
fail
# cat /sys/bus/ccw/devices/0.0.7009/last_known_reservation_state
lost
# chccwdev -d 0.0.7009
# chccwdev -e 0.0.7009
```

Reading and resetting the reservation state

How the DASD device driver handles I/O requests depends on the `last_known_reservation_state` sysfs attribute of the DASD.

About this task

The `last_known_reservation_state` attribute reflects the reservation state as held by the DASD device driver and can differ from the actual reservation state. Use the **`tunedasd -Q`** command to find out the actual reservation state. The `last_known_reservation_state` sysfs attribute can have the following values:

none The DASD device driver has no information about the device reservation state. I/O requests are processed as usual. If the DASD is reserved by another system, the I/O requests remain in the queue until they time out, or until the reservation is released.

reserved
The DASD device driver holds a valid reservation for the DASD and I/O requests are processed as usual. The DASD device driver changes this state

if notified that the DASD is no longer reserved to this system. The new state depends on the reservation policy (see "Handling lost device reservations" on page 178).

ignore The state is changed to `none`.

fail The state is changed to `lost`.

lost The DASD device driver had reserved the DASD, but subsequently another system has unconditionally reserved the DASD (see "Handling lost device reservations" on page 178). The device driver processes only requests that query the actual device reservation state. All other I/O requests for the device are returned as failed.

When the error that led another system to unconditionally reserve the DASD is resolved and the DASD has been released by this other system, there are two methods for resuming operations.

- Setting the DASD offline and back online.
- Resetting the reservation state of the DASD.

Attention: Do not resume operations by resetting the reservation state unless your system setup maintains data integrity on the DASD despite:

- The I/O errors that are caused by the unconditional reservation
- Any changes to the DASD through the other system

You reset the reservation state by writing `reset` to the `last_known_reservation_state` sysfs attribute of the DASD. Resetting is possible only for the `fail` reservation policy (see "Handling lost device reservations" on page 178) and only while the value of the `last_known_reservation_state` attribute is `lost`.

To find out the reservation state of a DASD issue a command of this form:

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

where <*device_bus_id*> specifies the DASD.

Example

The command in this example queries the reservation state of a DASD with bus ID `0.0.7009`.

```
# cat /sys/bus/ccw/devices/0.0.7009/last_known_reservation_state
reserved
```

Displaying DASD information

Use tools to display information about your DASDs, or read the attributes of the devices in sysfs.

About this task

There are several methods to display DASD information:

- Use **`lsdasd -l`** (see "lsdasd - List DASD devices" on page 621) to display summary information about the device settings and the device geometry of multiple DASDs.

- Use **`dasdview`** (see "dasdview - Display DASD structure" on page 582) to display details about the contents of a particular DASD.
- Read information about a particular DASD from sysfs, as described in this section.

The sysfs representation of a DASD is a directory of the form `/sys/bus/ccw/devices/<device_bus_id>`, where *<device_bus_id>* is the bus ID of the DASD. This sysfs directory contains a number of attributes with information about the DASD.

Table 21. Attributes with DASD information

Attribute	**Explanation**
alias	1 if the DASD is a parallel access volume (PAV) alias device. `0` if the DASD is a PAV base device or has not been set up as a PAV device. For an example about how to use PAV, see *How to Improve Performance with PAV*, SC33-8414. This attribute is read-only.
discipline	Indicates the base discipline, ECKD or FBA, that is used to access the DASD. If DIAG is enabled, this attribute might read DIAG instead of the base discipline. This attribute is read-only.
eer_enabled	1 if the DASD is enabled for extended error reporting, `0` if it is not enabled (see "Using extended error reporting for ECKD type DASD" on page 167).
erplog	1 if error recovery processing (ERP) logging is enabled, `0` if ERP logging is not enabled (see "Enabling and disabling logging" on page 169).
expires	Indicates the time, in seconds, that the DASD device driver waits for a response to an I/O request from a storage server. If this time expires, the device driver considers a request as failed and cancels it (see "Setting the timeout for I/O requests" on page 170).
failfast	1 if I/O operations are returned as failed immediately when the last path to the DASD is lost. `0` if a wait period for a path to return expires before an I/O operation is returned as failed. See "Enabling and disabling immediate failure of I/O requests" on page 170.
last_known_reservation_state	The reservation state as held by the DASD device driver. Values can be: **none** The DASD device driver has no information about the device reservation state. **reserved** The DASD device driver holds a valid reservation for the DASD. **lost** The DASD device driver had reserved the device, but this reservation has been lost to another system. See "Reading and resetting the reservation state" on page 179 for details.
online	1 if the DASD is online, `0` if it is offline (see "Setting a DASD online or offline" on page 168).
raw_track_access	1 if the DASD is in raw-track access mode, `0` if it is in default access mode (see "Accessing full ECKD tracks" on page 176).
readonly	1 if the DASD is read-only, `0` if it can be written to. This attribute is a device driver setting and does not reflect any restrictions that are imposed by the device itself. This attribute is ignored for PAV alias devices.
reservation_policy	Shows the reservation policy of the DASD. Possible values are `ignore` and `fail`. See "Handling lost device reservations" on page 178 for details.

Table 21. Attributes with DASD information (continued)

Attribute	Explanation
status	Reflects the internal state of a DASD device. Values can be: **unknown** Device detection has not started yet. **new** Detection of basic device attributes is in progress. **detected** Detection of basic device attributes has finished. **basic** The device is ready for detecting the disk layout. Low-level tools can set a device to this state when changing the disk layout, for example, when formatting the device. **unformatted** The disk layout detection found no valid disk layout. The device is ready for use with low-level tools like **dasdfmt**. **ready** The device is in an intermediate state. **online** The device is ready for use.
timeout	Indicates the time, in seconds, within which the DASD device driver must respond to an I/O request from a software layer above it. If the specified time expires before the request is completed, the DASD device driver cancels all related low-level I/O requests to storage systems and reports the request as failed (see "Setting the timeout for I/O requests" on page 170).
uid	A device identifier of the form *<vendor>*.*<serial>*.*<subsystem_id>*.*<unit_address>*.*<minidisk_identifier>* where *<vendor>* is the specification from the vendor attribute. *<serial>* is the serial number of the storage system. *<subsystem_id>* is the ID of the logical subsystem to which the DASD belongs on the storage system. *<unit_address>* is the address that is used within the storage system to identify the DASD. *<minidisk_identifier>* is an identifier that the z/VM system assigns to distinguish between minidisks on the DASD. This part of the uid is only present for Linux on z/VM and if the z/VM version and service level support this identifier. This attribute is read-only.
use_diag	1 if the DIAG access method is enabled, 0 if the DIAG access method is not enabled (see "Enabling the DASD device driver to use the DIAG access method" on page 165). Do not enable the DIAG access method for PAV alias devices.
vendor	Identifies the manufacturer of the storage system that contains the DASD. This attribute is read-only.

There are some more attributes that are common to all CCW devices (see "Device attributes" on page 11).

Procedure

Issue a command of this form to read an attribute:

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

where <*attribute*> is one of the attributes of Table 21 on page 181.

Example

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

```
# cat /sys/bus/ccw/devices/0.0.b100/alias
0
# cat /sys/bus/ccw/devices/0.0.b100/discipline
ECKD
# cat /sys/bus/ccw/devices/0.0.b100/eer_enabled
0
# cat /sys/bus/ccw/devices/0.0.b100/erplog
0
# cat /sys/bus/ccw/devices/0.0.b100/expires
30
# cat /sys/bus/ccw/devices/0.0.b100/failfast
0
# cat /sys/bus/ccw/devices/0.0.b100/last_known_reservation_state
reserved
# cat /sys/bus/ccw/devices/0.0.b100/online
1
# cat /sys/bus/ccw/devices/0.0.b100/raw_track_access
0
# cat /sys/bus/ccw/devices/0.0.b100/readonly
1
# cat /sys/bus/ccw/devices/0.0.b100/reservation_policy
ignore
# cat /sys/bus/ccw/devices/0.0.b100/status
online
# cat /sys/bus/ccw/devices/0.0.b100/timeout
120
# 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 11. SCSI-over-Fibre Channel device driver

The SCSI-over-Fibre Channel device driver for Linux on System z (zfcp device driver) supports virtual QDIO-based System z SCSI-over-Fibre Channel adapters (FCP devices) and attached SCSI devices (LUNs).

System z adapter hardware typically provides multiple channels, with one port each. You can configure a channel to use the Fibre Channel Protocol (FCP). This *FCP channel* is then virtualized into multiple FCP devices. Thus, an FCP device is a virtual QDIO-based System z SCSI-over-Fibre Channel adapter with a single port.

A single physical port supports multiple FCP devices. Using N_Port ID virtualization (NPIV) you can define virtual ports and establish a one-to-one mapping between your FCP devices and virtual ports (see "N_Port ID Virtualization for FCP channels" on page 191).

On Linux, an FCP device is represented by a CCW device that is listed under `/sys/bus/ccw/drivers/zfcp`. Do not confuse FCP devices with SCSI devices. A SCSI device is identified by a LUN.

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

Features

The zfcp device driver supports a wide range of SCSI devices, various hardware adapters, specific topologies, and specific features that depend on the System z hardware.

- Linux on System z can use various SAN-attached SCSI device types, including SCSI disks, tapes, CD-ROMs, and DVDs. For a list of supported SCSI devices, see

  `www.ibm.com/systems/z/connectivity`
- SAN access through the following hardware adapters:
  - FICON
  - FICON Express
  - FICON Express2
  - FICON Express4 (as of System z9)
  - FICON Express8 (as of System z10®)
  - FICON Express8S (as of zEnterprise)

  You can order hardware adapters as features for mainframe systems.

  See *Fibre Channel Protocol for Linux and z/VM on IBM System z*, SG24-7266 for more details about using FCP with Linux on System z.
- The zfcp device driver supports switched fabric and point-to-point topologies.
- As of zEnterprise, the zfcp device driver supports end-to-end data consistency checking.
- As of zEnterprise and FICON Express8S, the zfcp device driver supports the data router hardware feature to improve performance by reducing the path length.

For information about SCSI-3, the Fibre Channel Protocol, and Fibre Channel related information, see www.t10.org and www.t11.org

What you should know about zfcp

The zfcp device driver is a low-level driver or host-bus adapter driver that supplements the Linux SCSI stack.

Figure 37 illustrates how the device drivers work together.

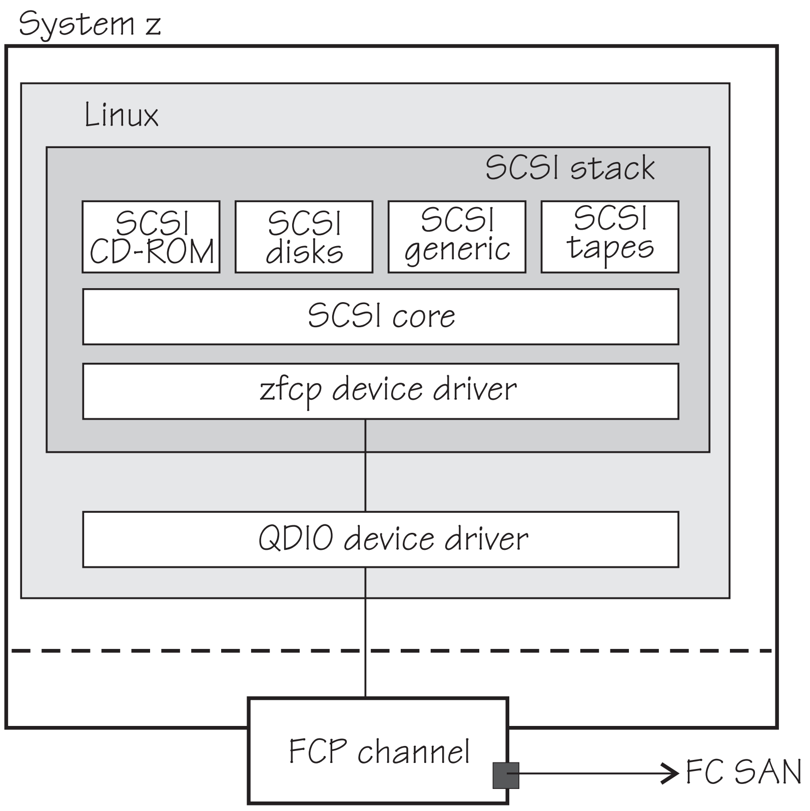


Figure 37. Device drivers supporting the Linux on System z FCP environment

For an introduction to the concepts of Fibre Channel Protocol support, and how various SCSI devices can be configured to build an IBM mainframe FCP environment, see *Fibre Channel Protocol for Linux and z/VM on IBM System z*, SG24-7266.

sysfs structures for FCP devices and SCSI devices

FCP devices are CCW devices. In the sysfs device driver view, remote target ports with their LUNs are nested below the FCP devices.

When Linux is booted, it senses the available FCP devices and creates directories of the form:

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

where <*device_bus_id*> is the device bus-ID that corresponds to an FCP device. You use the attributes in this directory to work with the FCP device.

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

The zfcp device driver automatically adds port information when the FCP device is set online and when remote storage ports (*target ports*) are added. Each added target port extends this structure with a directory of the form:

`/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>`
where <*wwpn*> is the worldwide port name (WWPN) of the target port. You use the attributes of this directory to work with the port.

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

With NPIV-enabled FCP devices, Linux uses automatic LUN scanning by default. The zfcp sysfs branch ends with the target port entries. For FCP devices that are not NPIV-enabled, or if automatic LUN scanning is disabled, see "Configuring SCSI devices" on page 207.

Information about zfcp objects and their associated objects in the SCSI stack is distributed over the sysfs tree. To ease the burden of collecting information about zfcp devices, ports, units, and their associated SCSI stack objects, a command that is called **`lszfcp`** is provided with s390-tools. See "lszfcp - List zfcp devices" on page 640 for more details about the command.

See also "Mapping the representations of a SCSI device in sysfs" on page 209.

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.

Alternatively, you can use udev to create device nodes that are based on unique properties of a SCSI device and so identify a particular device. Such device nodes are independent of the sequence in which the devices are set online and can help you to reliably address an intended disk space.

Multipath

Users of SCSI-over-Fibre Channel attached devices should always consider setting up and using redundant paths through their Fibre Channel Storage Area Network.

Path redundancy improves the availability of the LUNs. In Linux, you can set up path redundancy using the device-mapper multipath tool. For information about multipath devices and multipath partitions, see the chapter about multipathing in *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413.

Examples of udev-created SCSI device nodes

The udev utility program can create SCSI device nodes that are based on device information from sysfs.

Note: The format of the nodes that udev creates for you depends on distribution-specific configuration files. The device node descriptions in this section are according to generic rules as found in many distributions and in the udev rules file as it is shipped with the s390-tools packages.

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

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

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

Device nodes that are 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 worldwide port number of the target port and LUN is the logical unit number that represents the target SCSI device.

Device nodes that are based on file system information
udev creates device nodes of the form

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

where *<uuid>* is a unique file-system identifier (UUID) for the file system in a partition.

If a file system label is 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 that is generated by querying the device.

Example

For a SCSI device that is assigned the device name sda, has two partitions that are labeled boot and SWAP-sda2 respectively, a device bus-ID 0.0.3c1b (device number 0x3c1b), and a UUID 7eaf9c95-55ac-4e5e-8f18-065b313e63ca for the first and b4a818c8-747c-40a2-bfa2-acaa3ef70ead for the second partition, udev creates the following device nodes:

For the whole SCSI device:

- `/dev/sda` (standard device node according to the SCSI device naming scheme)

- `/dev/disk/by-path/ccw-0.0.3c1b-zfcp-0x500507630300c562:0x401040ea00000000`
- `/dev/disk/by-id/scsi-36005076303ffc56200000000000010ea`
- `/dev/disk/by-id/wwn-0x6005076303ffc56200000000000010ea`

For the first partition:

- `/dev/sda1` (standard device node according to the SCSI device naming scheme)
- `/dev/disk/by-label/boot`
- `/dev/disk/by-path/ccw-0.0.3c1b-zfcp-0x500507630300c562:0x401040ea00000000-part1`
- `/dev/disk/by-id/scsi-36005076303ffc56200000000000010ea-part1`
- `/dev/disk/by-uuid/7eaf9c95-55ac-4e5e-8f18-065b313e63ca`
- `/dev/disk/by-id/wwn-0x6005076303ffc56200000000000010ea-part1`

For the second partition:

- `/dev/sda2` (standard device node according to the SCSI device naming scheme)
- `/dev/disk/by-label/SWAP-sda2`
- `/dev/disk/by-path/ccw-0.0.3c1b-zfcp-0x500507630300c562:0x401040ea00000000-part2`
- `/dev/disk/by-id/scsi-36005076303ffc56200000000000010ea-part2`
- `/dev/disk/by-uuid/b4a818c8-747c-40a2-bfa2-acaa3ef70ead`
- `/dev/disk/by-id/wwn-0x6005076303ffc56200000000000010ea-part2`

Device nodes by-uuid use a unique file-system identifier that does not relate to the partition number.

Creating SCSI device nodes with mknod

You can create your own device nodes with the **mknod** command.

Procedure

Issue:

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

See “Finding the major and minor numbers for a device” on page 212 if you need to create your own nodes.

Partitioning a SCSI device

You can partition SCSI devices that are attached through an FCP channel in the same way that you can partition SCSI attached devices on other platforms.

About this task

Use the **fdisk** command to partition a SCSI disk, not **fdasd**.

If your distribution provides udev, udev might create device nodes for your partitions. See your distribution documentation for details. If you need to create your own nodes for your partitions, see “Finding the major and minor numbers for a device” on page 212.

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 38, the zfcp HBA API support includes a user space library.

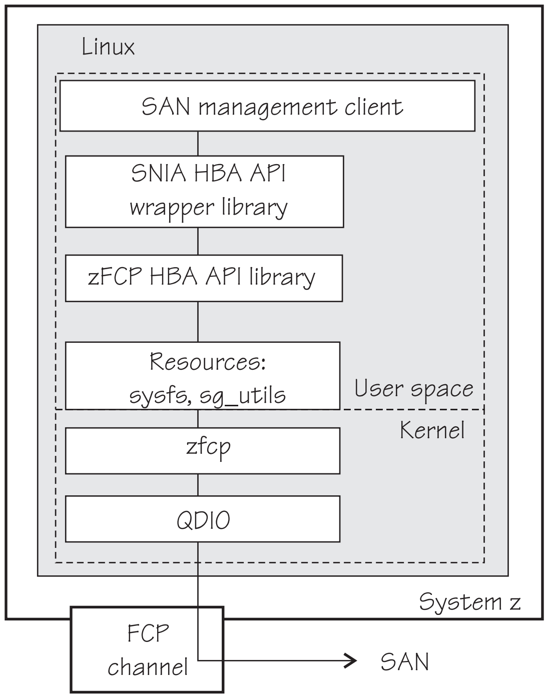


Figure 38. zfcp HBA API support modules

The zFCP HBA API support uses the SNIA (Storage Networking Industry Association) library, hbaapi_src_<*x.x*>.tgz, which can be found at
`hbaapi.sourceforge.net`

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

In a Linux on System z environment HBAs are usually virtualized and are shown as *FCP devices*. FCP devices are represented by CCW devices that are listed in `/sys/bus/ccw/drivers/zfcp`. Do not confuse FCP devices with SCSI devices. A SCSI device is a disk device that is identified by a LUN.

Technically it is also possible for applications to use the zFCP HBA API library directly, however, this is not the preferred method.

For information about setting up the HBA API support, see “Installing the zfcp HBA API library” on page 194.

N_Port ID Virtualization for FCP channels

Through N_Port ID Virtualization (NPIV), the sole port of an FCP channel appears as multiple, distinct ports with separate port identification.

NPIV support can be configured on the SE per CHPID and LPAR for an FCP channel. The zfcp device driver supports NPIV error messages and adapter attributes. See “Displaying FCP channel and device information” on page 197 for the Fibre Channel adapter attributes.

For more information, see the connectivity page at
`www.ibm.com/systems/z/connectivity`

See also the chapter on NPIV in *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413.

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

Building a kernel with the zfcp device driver

Control the build options for the zfcp device driver through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Figure 39 summarizes the kernel configuration menu options that are relevant to the zfcp device driver:

```
Device Drivers --->
   ...
   SCSI device support --->
      ...
      --- SCSI support type (disk, tape, CD-ROM) (depends on SCSI) ---
      ...
      SCSI low-level drivers --->         (common code option CONFIG_SCSI_LOWLEVEL)
         ...
         FCP host bus adapter driver for IBM eServer zSeries (CONFIG_ZFCP)*
```

Figure 39. zfcp kernel configuration menu options

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

In addition, the CONFIG_QDIO option and the following common code options are required:
- CONFIG_SCSI
- CONFIG_BLK_DEV_INTEGRITY (Required only for end-to-end data consistency support)

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

As for Linux on any platform, you need the common code options for specific devices and file systems you want to support. For example:

- SCSI disks support and PC-BIOS disk layout support

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

Setting up the zfcp device driver

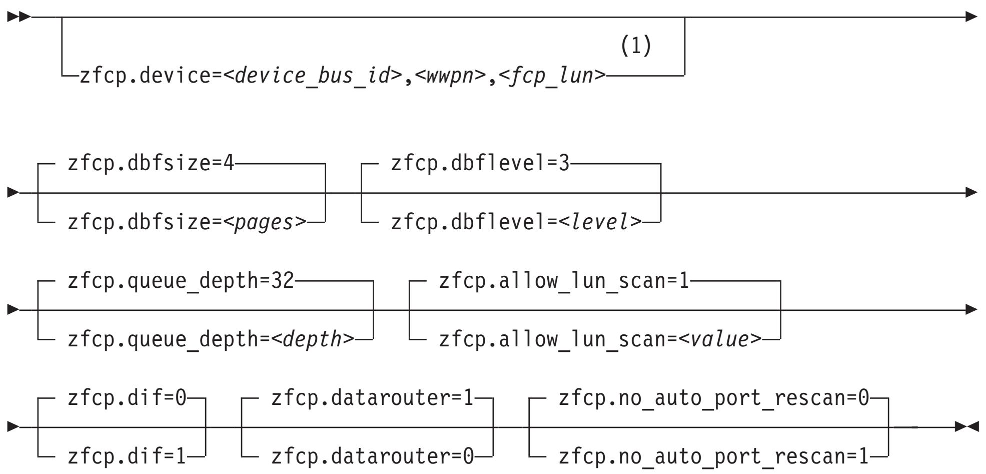
Configure the zfcp device driver through the kernel or module parameters. You might also need to install the zfcp HBA API library.

zfcp device driver kernel parameters

If the zfcp device driver was compiled into the kernel, you configure the device driver by adding parameters to the kernel parameter line.

Use the zfcp.device kernel parameter to enable a SCSI device to be used as initial device. This parameter enables only a single SCSI LUN. For production systems, consider a multipath setup with two or more redundant paths to each volume.

zfcp kernel parameter syntax



Notes:

1 For experimental use only. Do not use on production systems.

where:

`zfcp.device=``<device_bus_id>,<wwpn>,<fcp_lun>`

Attention: This parameter can break distribution-specific tools. Do not use on production systems.

The `zfcp.device=` parameter enables a single SCSI device to be used as an initial device. For example, as the device with the root file system for experimental setups. This parameter enables only a single SCSI LUN with a single access path.

Production systems typically require access to multiple volumes and a multipath setup with two or more redundant paths to each volume. This environment cannot be attained with the `zfcp.device=` parameter. See the documentation that is provided by your distributor about which tools to use to set up SCSI devices in a production environment.

`<`*`device_bus_id`*`>`
: specifies the FCP device through which the SCSI device is attached.

`<`*`wwpn`*`>`
: specifies the target port through which the SCSI device is attached.

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

`zfcp.dbfsize=``<`*`pages`*`>`
specifies the number of pages to be used for the debug feature.

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

hba	FCP device.
san	Storage Area Network.
rec	Error Recovery Process.
scsi	SCSI

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

```
zfcp.dbfsize=6
```

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

`zfcp.dbflevel=``<`*`level`*`>`
sets the initial log level of the debug feature. The value is an integer in the range 0 - 6, where greater numbers generate more detailed information. The default is 3.

`zfcp.queue_depth=``<`*`depth`*`>`
specifies the number of commands that can be issued simultaneously to a SCSI device. The default is 32. The value that you set here is used as the default queue depth for new SCSI devices. You can change the queue depth for each SCSI device that uses the queue_depth sysfs attribute, see "Setting the queue depth" on page 213.

`zfcp.allow_lun_scan=``<`*`value`*`>`
disables the automatic LUN scan for FCP devices that run in NPIV mode if set to 0, n, or N. To enable the LUN scanning set the parameter to 1, y, or Y. When the LUN scan is disabled, all LUNs must be configured through the unit_add zfcp attribute in sysfs. LUN scan is enabled by default.

`zfcp.dif=`
turns end-to-end data consistency checking on if set to 1, y, or Y and off if set to 0, n, or N. The default is 0.

`zfcp.datarouter=`
enables (if set to 1, y, or Y) or disables (if set to 0, n, or N) support for the hardware data routing feature. The default is 1.

Note: The hardware data routing feature becomes active only for FCP devices that are based on adapter hardware with hardware data routing support.

`zfcp.no_auto_port_rescan=`
turns the automatic port rescan feature off (if set to 1, y, or Y) or on (if set to 0, n, or N). The default is 0. Automatic rescan is always performed when an adapter is set online and when user-triggered writes to the sysfs attribute port_rescan occur.

Example

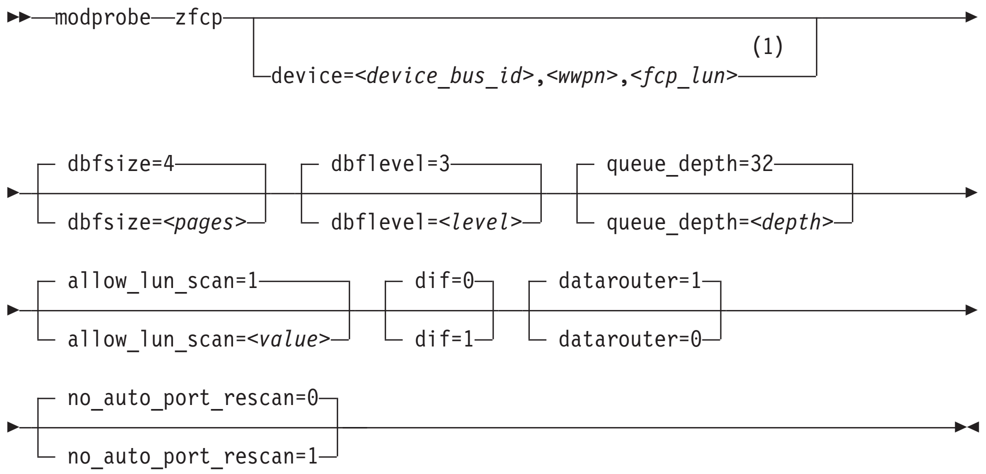
Use the following kernel parameters to enable end-to-end data consistency checking and the hardware data routing feature:

```
zfcp.dif=1 zfcp.datarouter=1
```

zfcp module parameters

If the zfcp device driver was built as a separate module, the device driver is configured through module parameters when the module is loaded.

zfcp module parameter syntax



Notes:

1 For experimental use only. Do not use on production systems.

The variables have the same meaning as the corresponding kernel parameters with prefix `zfcp.` in "zfcp device driver kernel parameters" on page 192.

Installing the zfcp HBA API library

You need several packages to use the zfcp HBA API library.

Before you begin

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

- lib-zfcp-hbaapi-2.<*x*>.tar.gz, the zfcp HBA API library. In the library name, <*x*> represents the newest available version.

- The libsysfs-2.1 package, a library interface to sysfs.
- The sg_utils package, a utility for devices that use SCSI commands.
- The doxygen package, optional for documentation.

About this task

When installing the library the default is to use the SNIA library. Should you want to build a standalone version you need to set the compile option `--enable-vendor-lib=no` in the configuration file.

Procedure

Perform the following steps to install the library:

1. Go to www.ibm.com/developerworks/linux/linux390/zfcp-hbaapi.html.
2. Click the link for the latest zfcp HBA API library version for the Development stream.
3. Download the source package lib-zfcp-hbaapi-2.<*x*>.tar.gz.
4. Compile and install the package:

   ```
   # tar xzf lib-zfcp-hbaapi-2.<x>.tar.gz
   # cd libzfcphbaapi-2.<x>
   # ./configure
   # make
   # make install
   ```

5. Optional: Build and install documentation. For this step you require the package doxygen.

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

Results

You have installed:
- Shared and static versions of libzfcphbaapi at `/usr/local/lib`.
- If you built a standalone version, the header file hbaapi.h at `/usr/local/include`.
- Optionally, the documentation package at `/usr/local/share/doc/zfcphbaapi-2.<x>`.

If you have built the vendor library with the SNIA library, there are two entry points:
- HBA_RegisterLibrary, used if SNIA V1 was installed.
- HBA_RegisterLibraryV2 , used if SNIA V2 was installed.

The SNIA library expects a configuration file called `/etc/hba.conf` that contains the path to the vendor-specific library with the actual implementation. 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, must supply the function HBA_RegisterLibrary that returns all function pointers to the wrapper library and thus makes them available to the application.

Note: The exact contents of the library depends on the version, see "API provided by the zfcp HBA API support" on page 221.

Working with FCP devices

Set an FCP device online before you attempt to perform any other tasks.

Working with FCP devices comprises the following tasks:

- "Setting an FCP device online or offline"
- "Displaying FCP channel and device information" on page 197
- "Recovering a failed FCP device" on page 200
- "Finding out whether NPIV is in use" on page 201
- "Logging I/O subchannel status information" on page 202

Setting an FCP device online or offline

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

Before you begin

This procedure is intended for experimental environments only and might interfere with distribution-specific tools.

See the documentation that is provided by your distributor about which tools to use for a persistent configuration in a production environment.

About this task

See "Working with newly available devices" on page 12 to avoid errors when you work with devices that have become available to a running Linux instance.

Setting an FCP device online registers it with the Linux SCSI stack and updates the symbolic port name for the device on the FC name server. For FCP setups that use NPIV mode, the device bus-ID and the host name of the Linux instance are added to the symbolic port name.

Setting an FCP device online also automatically runs the scan for ports in the SAN and waits for this port scan to complete.

To check if setting the FCP device online was successful, you can use a script that first sets the FCP device online and after this operation completes checks if the WWPN of a target port has appeared in sysfs.

When you set an FCP device offline, the port and LUN subdirectories are preserved. Setting an FCP device offline in sysfs interrupts the communication between Linux and the FCP channel. After a timeout has expired, the port and LUN attributes indicate that the ports and LUNs are no longer accessible. The transition of the FCP device to the offline state is synchronous, unless the device is disconnected.

For disconnected devices, writing 0 to the online sysfs attribute triggers an asynchronous deregistration process. When this process is completed, the device with its ports and LUNs is no longer represented in sysfs.

When the FCP device is set back online, the SCSI device names and minor numbers are freshly assigned. The mapping of devices to names and numbers might be different from what they were before the FCP device was set offline.

Procedure

There are two methods for setting an FCP device online or offline:
- Use the **chccwdev** command (“chccwdev - Set CCW device attributes” on page 543). This is the preferred method.
- Alternatively, you can write 1 to an FCP device's online attribute to set it online, or 0 to set it offline.

Examples

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

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

  or

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

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

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

  or

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

Displaying FCP channel and device information

For each online FCP device, there is a number of read-only attributes in sysfs that provide information about the corresponding FCP channel and FCP device.

Before you begin

The FCP device must be online for the FCP channel information to be valid.

About this task

The following tables summarize the relevant attributes.

Table 22. Attributes with FCP channel information

Attribute	Explanation
card_version	Version number that identifies a particular hardware feature.
hardware_version	Number that identifies a hardware version for a particular feature. The initial hardware version of a feature is zero. This version indicator is increased only for hardware modifications of the same feature. Appending hardware_version to card_version results in a hierarchical version indication for a physical adapter.
lic_version	Microcode level.
peer_wwnn	WWNN of peer for a point-to-point connection.
peer_wwpn	WWPN of peer for a point-to-point connection.
peer_d_id	Destination ID of the peer for a point-to-point connection.

Table 23. Attributes with FCP device information

Attribute	Explanation
in_recovery	Shows if the FCP channel is in recovery (0 or 1).

For the attributes availability, cmb_enable, and cutype, see "Device directories" on page 11. The status attribute is reserved.

Table 24. Relevant transport class attributes, fc_host attributes

Attribute	Explanation
maxframe_size	Maximum frame size.
node_name	Worldwide node name (WWNN).
permanent_port_name	WWPN associated with the physical port of the FCP channel.
port_id	A unique ID (N_Port_ID) assigned by the fabric. In an NPIV setup, each virtual port is assigned a different port_id.
port_name	WWPN associated with the FCP device. If N_Port ID Virtualization is not available, the WWPN of the physical port (see permanent_port_name).
port_type	The port type indicates the topology of the port.
serial_number	The 32-byte serial number of the adapter hardware that provides the FCP channel.
speed	Speed of FC link.
supported_classes	Supported FC service class.
supported_speeds	Supported speeds.
symbolic_name	The symbolic port name that is registered with the FC name server.
tgid_bind_type	Target binding type.

Table 25. Relevant transport class attributes, fc_host statistics

Attribute	Explanation
reset_statistics	Writeable attribute to reset statistic counters.
seconds_since_last_reset	Seconds since last reset of statistic counters.
tx_frames	Transmitted FC frames.
tx_words	Transmitted FC words.
rx_frames	Received FC frames.
rx_words	Received FC words.
lip_count	Number of LIP sequences.
nos_count	Number of NOS sequences.
error_frames	Number of frames that are received in error.
dumped_frames	Number of frames that are lost because of lack of host resources.
link_failure_count	Link failure count.
loss_of_sync_count	Loss of synchronization count.
loss_of_signal_count	Loss of signal count.
prim_seq_protocol_err_count	Primitive sequence protocol error count.

Table 25. Relevant transport class attributes, fc_host statistics (continued)

Attribute	Explanation
invalid_tx_word_count	Invalid transmission word count.
invalid_crc_count	Invalid CRC count.
fcp_input_requests	Number of FCP operations with data input.
fcp_output_requests	Number of FCP operations with data output.
fcp_control_requests	Number of FCP operations without data movement.
fcp_input_megabytes	Megabytes of FCP data input.
fcp_output_megabytes	Megabytes of FCP data output.

Procedure

Use the **cat** command to read an attribute.

- Issue a command of this form to read an attribute:

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

  where:

  <device_bus_id>
  : specifies an FCP device that corresponds to the FCP channel.

  <attribute>
  : is one of the attributes in Table 22 on page 197 or Table 23 on page 198.

- To read attributes of the associated SCSI host use:

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

  where:

  <host_name>
  : is the ID of the SCSI host.

  <attribute>
  : is one of the attributes in Table 24 on page 198.

- To read the statistics' attributes:

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

  where:

  <host_name>
  : is the ID of the SCSI host.

  <attribute>
  : is one of the attributes in Table 25 on page 198.

Examples

- In this example, information is displayed about an FCP channel that corresponds to an FCP device with bus ID 0.0.3d0c:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/hardware_version
0x00000000
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/lic_version
0x00000302
```

- Alternatively you can use **lszfcp** (see “lszfcp - List zfcp devices” on page 640) to display attributes of an FCP channel:

```
# lszfcp -b 0.0.3d0c -a
0.0.3d0c host0
Bus = "ccw"
    availability        = "good"
    card_version        = "0x0005"
    cmb_enable          = "0"
    cutype              = "1731/03"
    devtype             = "1732/03"
    failed              = "0"
    hardware_version    = "0x00000000"
    in_recovery         = "0"
    lic_version         = "0x00000302"
    modalias            = "ccw:t1731m03dt1732dm03"
    online              = "1"
    peer_d_id           = "0x000000"
    peer_wwnn           = "0x0000000000000000"
    peer_wwpn           = "0x0000000000000000"
    status              = "0x5400000a"
    uevent              = "DRIVER=zfcp"
Class = "fc_host"
    active_fc4s         = "0x00 0x00 0x01 0x00 ..."
    dev_loss_tmo        = "60"
    maxframe_size       = "2112 bytes"
    node_name           = "0x5005076400c3c03f"
    permanent_port_name = "0x500507601d801a2e"
    port_id             = "0x68f880"
    port_name           = "0x500507601d801a2e"
    port_state          = "Online"
    port_type           = "NPort (fabric via point-to-point)"
    serial_number       = "IBM0200000003C03F"
    speed               = "8 Gbit"
    supported_classes   = "Class 2, Class 3"
    supported_fc4s      = "0x00 0x00 0x01 0x00 ..."
    supported_speeds    = "1 Gbit, 4 Gbit"
    symbolic_name       = ""
    tgtid_bind_type     = "wwpn (World Wide Port Name)"
Class = "scsi_host"
    active_mode         = "Initiator"
    can_queue           = "4096"
    cmd_per_lun         = "1"
    host_busy           = "0"
    megabytes           = "0 0"
    proc_name           = "zfcp"
    prot_capabilities   = "0"
    prot_guard_type     = "0"
    queue_full          = "0 167297"
    requests            = "83 0 3"
    seconds_active      = "157"
    sg_prot_tablesize   = "0"
    sg_tablesize        = "538"
    state               = "running"
    supported_mode      = "Initiator"
    unchecked_isa_dma   = "0"
    unique_id           = "15387"
    utilization         = "1 1 0"
```

Recovering a failed FCP device

Failed FCP devices are automatically recovered by the zfcp device driver. You can read the `in_recovery` attribute to check whether recovery is under way.

Before you begin

The FCP device must be online.

Procedure

Perform these steps to find out the recovery status of an FCP device and, if needed, start or restart recovery:

1. Issue a command of this form:

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

   The value is `1` if recovery is under way and `0` otherwise. If the value is `0` for a non-operational FCP device, recovery might have failed. Alternatively, the device driver might have failed to detect that the FCP device is malfunctioning.
2. To find out whether recovery failed, read the `failed` attribute. Issue a command of this form:

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

   The value is `1` if recovery failed and `0` otherwise.
3. You can start or restart the recovery process for the FCP device by writing `0` to the `failed` attribute. Issue a command of this form:

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

Example

In the following example, an FCP device with a device bus-ID 0.0.3d0c is malfunctioning. The first command reveals that recovery is not already under way. The second command manually starts recovery for the FCP device:

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

Finding out whether NPIV is in use

The FCP setup runs in NPIV mode if the `port_type` attribute of the FCP device attribute contains the string "NPIV". Alternatively, if the applicable `permanent_port_name` and `port_name` are not the same and are not NULL.

Procedure

Read the `port_type` attribute of the FCP device.
For example:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.1940/host0/fc_host/host0/port_type
NPIV VPORT
```

Alternatively, compare the values of the `permanent_port_name` attribute and the `port_name`.

Tip: You can use **`lszfcp`** (see "lszfcp - List zfcp devices" on page 640) to list the FCP device attributes.

Example

```
# lszfcp -b 0.0.1940 -a
0.0.1940 host0
Bus = "ccw"
    availability        = "good"
    ...
Class = "fc_host"
    ...
    node_name           = "0x5005076400c1ebae"
    permanent_port_name = "0x50050764016219a0"
    port_id             = "0x65ee01"
    port_name           = "0xc05076ffef805388"
    port_state          = "Online"
    port_type           = "NPIV VPORT"
    ...
    symbolic_name       = "DEVNO: 0.0.1940 NAME: mylinux"
    ...
```

The port_type attribute directly indicates that NPIV is used. The example also shows that permanent_port_name is different from port_name and neither is NULL. The example also shows the symbolic_name attribute that shows the symbolic port name that was registered on the FC name server.

Logging I/O subchannel status information

When severe errors occur for an FCP device, the FCP device driver triggers a set of log entries with I/O subchannel status information.

The log entries are available through the SE Console Actions Work Area with the View Console Logs function. In the list of logs, these entries have the prefix 1F00. The content of the entries is intended for support specialists.

Working with target ports

You can scan for ports, display port information, recover a port, or remove a port.

Working with target ports comprises the following tasks:
- “Scanning for ports”
- “Displaying port information” on page 203
- “Recovering a failed port” on page 205
- “Removing ports” on page 205

Scanning for ports

Newly available target ports are discovered. However, you might want to trigger a port scan to re-create accidentally removed port information or to assure that all ports are present.

Before you begin

The FCP device must be online.

About this task

The zfcp device driver automatically adds port information to sysfs when:
- The FCP device is set online
- Target ports are added to the Fibre Channel fabric, unless the module parameter no_auto_port_rescan is set to 1. See “zfcp module parameters” on page 194.

Scanning for ports might take some time to complete. Commands that you issue against ports or LUNs while scanning is in progress are delayed and processed when port scanning is completed.

Use the `port_rescan` attribute if a target port was accidentally deleted from the adapter configuration or if you are unsure whether all ports were added to sysfs.

Procedure

Issue a command of this form:

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

where *<device_bus_id>* specifies the FCP device through which the target ports are attached.

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

Example

In this example, a port with WWPN 0x500507630303c562 is already configured for an FCP device with bus ID 0.0.3d0c. An additional target port with WWPN 0x500507630300c562 is automatically configured by triggering a port scan.

```
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/port_rescan
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
0x500507630300c562
```

Displaying port information

For each target port, there is a number of read-only sysfs attributes with port information.

About this task

Table 26 summarizes the relevant attributes.

Table 26. Attributes with port information

Attribute	Explanation
access_denied	This attribute is obsolete. The value is always `0`.
in_recovery	Shows if port is in recovery (0 or 1)

Table 27. Transport class attributes with port information

Attribute	Explanation
node_name	WWNN of the remote port (target port).
port_name	WWPN of the remote port.
port_id	Destination ID of the remote port
port_state	State of the remote port.
roles	Role of the remote port (usually FCP target).

Table 27. Transport class attributes with port information (continued)

Attribute	Explanation
scsi_target_id	Linux SCSI ID of the remote port.
supported_classes	Supported classes of service.

Procedure

Use the **cat** command to read an attribute.

- Issue a command of this form to read an attribute:

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

  where:

  <device_bus_id>
  : specifies the FCP device.

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

  <attribute>
  : is one of the attributes in Table 26 on page 203.

- To read attributes of the associated target port, 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 27 on page 203.

Tip: With the HBA API package installed, you can also use the **zfcp_ping** and **zfcp_show** commands to find out more about your ports. See "Tools for investigating your SAN configuration" on page 223.

Examples

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

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

- To display transport class attributes of a target port you can use **lszfcp**:

```
# lszfcp -p 0x500507630300c562 -a
0.0.3d0c/0x500507630300c562 rport-0:0-0
Class = "fc_remote_ports"
    dev_loss_tmo        = "60"
    fast_io_fail_tmo    = "off"
    maxframe_size       = "2048 bytes"
    node_name           = "0x5005076303ffc562"
    port_id             = "0x652113"
    port_name           = "0x500507630300c562"
    port_state          = "Online"
    roles               = "FCP Target"
    scsi_target_id      = "0"
    supported_classes   = "Class 2, Class 3"
```

Recovering a failed port

Failed target ports are automatically recovered by the zfcp device driver. You can read the `in_recovery` attribute to check whether recovery is under way.

Before you begin

The FCP device must be online.

Procedure

Perform these steps to find out the recovery status of a port and, if needed, start or restart recovery:

1. Issue a command of this form:

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

   where:

   <device_bus_id>
   : specifies the FCP device.

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

   The value is `1` if recovery is under way and `0` otherwise. If the value is `0` for a non-operational port, recovery might have failed or the device driver might have failed to detect that the port is malfunctioning.
2. To find out whether recovery failed, read the `failed` attribute. Issue a command of this form:

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

   The value is `1` if recovery failed, and `0` otherwise.
3. You can start or restart the recovery process for the port by writing `0` to the `failed` attribute. Issue a command of this form:

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

Example

In the following example, a port with WWPN 0x500507630300c562 that is attached through an FCP device with bus ID 0.0.3d0c is malfunctioning. The first command reveals that recovery is not already under way. The second command manually starts recovery for the port:

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

Removing ports

Removing unused ports can save FCP channel resources. Additionally setting the `no_auto_port_rescan` attribute avoids unnecessary attempts to recover unused remote ports.

Before you begin

The FCP device must be online.

About this task

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

You cannot remove a port while SCSI devices are configured for it (see "Configuring SCSI devices" on page 207) or if the port is in use, for example, by error recovery.

Note: The next port scan will attach all available ports, including any previously removed ports. To prevent removed ports from being reattached automatically, use zoning or the no_auto_port_rescan module parameter, see "zfcp module parameters" on page 194.

Procedure

To remove a port from an FCP device, write the WWPN of the port to the `port_remove` attribute of the FCP device.

Issue a command of this form:

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

where:

<device_bus_id>
specifies the FCP device.

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

Example

In this example, two ports with WWPN 0x500507630303c562 and 0x500507630300c562 are configured for an FCP device with bus ID 0.0.3d0c. The port with WWPN 0x500507630303c562 is then removed.

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

Working with SCSI devices

In an NPIV setup with auto lun scan, the SCSI devices are configured automatically. Otherwise, you must configure FCP LUNs to obtain SCSI devices. In both cases, you can configure SCSI devices, display information, and remove SCSI devices.

Working with SCSI devices comprises the following tasks:
- "Configuring SCSI devices" on page 207
- "Mapping the representations of a SCSI device in sysfs" on page 209
- "Displaying information about SCSI devices" on page 210
- "Finding the major and minor numbers for a device" on page 212
- "Setting the queue depth" on page 213

- “Recovering failed SCSI devices” on page 214
- “Updating the information about SCSI devices” on page 215
- “Setting the SCSI command timeout” on page 216
- “Controlling the SCSI device state” on page 216
- “Removing SCSI devices” on page 217

Configuring SCSI devices

FCP devices that use NPIV mode detect the LUNs automatically and no configuring is necessary. If needed, write the LUN to be configured to the sysfs `unit_add` attribute of the applicable target port.

For each FCP device that uses NPIV mode and if you did not disable automatic LUN scanning (see “zfcp device driver kernel parameters” on page 192), the LUNs are configured for you. In this case, *no* FCP LUN entries are created under `/sys/bus/ccw/drivers/zfcp/`*<device_bus_id>*`/`*<wwpn>*.

To find out whether an FCP device is using NPIV mode, check the `port_type` attribute. For example:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.1901/host0/fc_host/host0/port_type
NPIV VPORT
```

To find out whether automatic LUN scanning is enabled, check the current setting of the module parameter zfcp.allow_lun_scan. The example below shows automatic LUN scanning as turned on.

```
# cat /sys/module/zfcp/parameters/allow_lun_scan
Y
```

Automatically attached SCSI devices

FCP devices that use NPIV mode detect the LUNs automatically and no configuring is necessary.

In this case, *no* FCP LUN entries are created under `/sys/bus/ccw/drivers/zfcp/`*<device_bus_id>*`/`*<wwpn>*.

What to do next

To check whether a SCSI device is registered, check for a directory with the name of the LUN in `/sys/bus/scsi/devices`. If there is no SCSI device for this LUN, the LUN is not valid in the storage system, or the FCP device is offline in Linux.

Manually configured FCP LUNs and their SCSI devices

For FCP devices that do not use NPIV mode, or if automatic LUN scanning is disabled, FCP LUNs must be configured manually to obtain SCSI devices.

Before you begin

Attention: Use this procedure only to dynamically test configuration settings.

This procedure is intended for experimental environments only and might interfere with distribution-specific tools.

See the documentation that is provided by your distributor about which tools to use for a persistent configuration in a production environment. You can always specify additional zfcp module parameters as explained in Chapter 3, “Kernel and module parameters,” on page 25

Procedure

If your FCP device does not use NPIV mode, or if you have disabled automatic LUN scanning, proceed as follows:

To configure a SCSI device for a target port, write the device's LUN to the port's `unit_add` attribute. Issue a command of this form:

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

where:

<fcp_lun>
: is the LUN of the SCSI device to be configured. The LUN is a 16 digit hexadecimal value padded with zeros, for example 0x4010403300000000.

<device_bus_id>
: specifies the FCP device.

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

This command starts a process with multiple steps:

1. It creates a directory in `/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>` with the LUN as the directory name. The directory is part of the list of all LUNs to configure. Without NPIV or with auto LUN scanning disabled, zfcp registers only FCP LUNs contained in this list with the Linux SCSI stack in the next step.
2. It initiates the registration of the SCSI device with the Linux SCSI stack. The FCP device must be online for this step.
3. It waits until the Linux SCSI stack registration completes successfully or returns an error. It then returns control to the shell. A successful registration creates a sysfs entry in the SCSI branch (see “Mapping the representations of a SCSI device in sysfs” on page 209).

Example

In this example, a target port with WWPN 0x500507630300c562 is attached through an FCP device with bus ID 0.0.3d0c. A SCSI device with LUN 0x4010403200000000 is already configured for the port. An additional SCSI device with LUN 0x4010403300000000 is added to the port.

```
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x*
0x4010403200000000
# echo 0x4010403300000000 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/unit_add
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x500507630300c562/0x*
0x4010403200000000
0x4010403300000000
```

What to do next

To check whether a SCSI device is registered for the configured LUN, check for a directory with the name of the LUN in `/sys/bus/scsi/devices`. If there is no SCSI device for this LUN, the LUN is not valid in the storage system, or the FCP device is offline in Linux.

To see which LUNs are currently configured for the port, list the contents of `/sys/bus/ccw/drivers/zfcp/`*`<device_bus_id>`*`/`*`<wwpn>`*.

Mapping the representations of a SCSI device in sysfs

Each SCSI device that is configured is represented by multiple directories in sysfs, in particular, within the SCSI branch. Only manually configured LUNs are also represented within the zfcp branch.

About this task

The directory in the sysfs SCSI branch has the following form:

`/sys/bus/scsi/devices/`*`<scsi_host_no>`*`:0:`*`<scsi_id>`*`:`*`<scsi_lun>`*

where:

<scsi_host_no>
: is the SCSI host number that corresponds to the FCP device.

<scsi_id>
: is the SCSI ID of the target port.

<scsi_lun>
: is the LUN of the SCSI device.

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

For manually configured LUNs, see "Manually configured FCP LUNs and their SCSI devices" on page 207 for details about the directory in the zfcp branch.

Figure 40 shows how the directory name is composed in the sysfs SCSI branch. The sysfs zfcp branch only exists for manually configured FCP LUNs. For manually configured FCP LUNs, the directory name is composed of attributes of consecutive directories and you can find the name of the directory in the sysfs SCSI branch by reading the corresponding attributes in the zfcp branch.

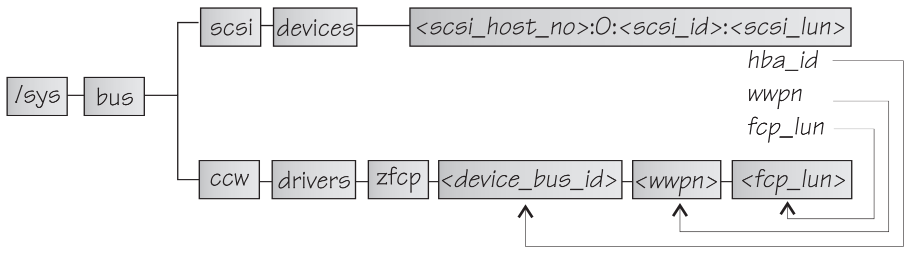


Figure 40. SCSI devices in sysfs

The `hba_id`, `wwpn`, and `fcp_lun` attributes of the SCSI device in the SCSI branch match the names of the *<device_bus_id>*, *<wwpn>*, and *<fcp_lun>* directories for the same SCSI device in the zfcp branch.

Procedure

Use **lszfcp** (see “lszfcp - List zfcp devices” on page 640) to map the two representations of a SCSI device.

Example

This example shows how to 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 for an FCP device with bus ID 0.0.3d0c, maps to SCSI device "0:0:0:0".

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

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

Displaying information about SCSI devices

For each SCSI device, there is a number of read-only attributes in sysfs that provide information for the device.

About this task

Table 28 summarizes the read-only attributes for manually configured FCP LUNs, including those attributes that indicate whether the device access is restricted by access control software on the FCP channel.

Table 28. Attributes of manually configured FCP LUNs with device access information

Attribute	Explanation
access_denied	Flag that indicates whether access to the device is restricted by the FCP channel. The value is 1 if access is denied and `0` if access is permitted. If access is denied to your Linux instance, confirm that your SCSI devices are configured as intended. Also, be sure that you really want to share a SCSI device. For shared access to a SCSI device, preferably use NPIV (see “N_Port ID Virtualization for FCP channels” on page 191). You might also use different FCP channels or target ports.
access_shared	This attribute is obsolete. The value is always `0`.
access_readonly	This attribute is obsolete. The value is always `0`.
in_recovery	Shows if unit is in recovery (0 or 1)

Table 29 on page 211 lists further read-only attributes with information about the SCSI device.

Table 29. SCSI device class attributes

Attribute	Explanation
device_blocked	Flag that indicates whether the device is in blocked state (1) or not (0).
iocounterbits	The number of bits used for I/O counters.
iodone_cnt	The number of completed or rejected SCSI commands.
ioerr_cnt	The number of SCSI commands that completed with an error.
iorequest_cnt	The number of issued SCSI commands.
queue_type	The type of queue for the SCSI device. The value can be one of the following types: • none • simple • ordered
model	The model of the SCSI device, received from inquiry data.
rev	The revision of the SCSI device, received from inquiry data.
scsi_level	The SCSI revision level, received from inquiry data.
type	The type of the SCSI device, received from inquiry data.
vendor	The vendor of the SCSI device, received from inquiry data.
fcp_lun	The LUN of the SCSI device in 64-bit format.
hba_id	The bus ID of the SCSI device.
wwpn	The WWPN of the remote port.

Procedure

Issue a command of this form to read an attribute of a manually configured FCP LUN:

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

where:

<device_bus_id>
: specifies the FCP device.

<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 28 on page 210.

Use the **lszfcp** command (see “lszfcp - List zfcp devices” on page 640) to display information about the associated SCSI device.
Alternatively, you can use sysfs to read the information. To read attributes of the associated SCSI device, use a command of this form:

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

where:

<device_name>
: is the name of the associated SCSI device.

<*attribute*>
is one of the attributes in Table 29 on page 211.

Tip: For SCSI-attached tape devices, you can display a summary of this information by using the **lstape** command (see "lstape - List tape devices" on page 633).

Examples

- In this example, information is displayed for a a manually configured FCP LUN with LUN 0x4010403200000000 that is accessed through a target port with WWPN 0x500507630300c562 and is attached through an FCP device with bus ID 0.0.3d0c. For the device access is permitted.

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

  For the device to be accessible, the access_denied attribute of the target port, 0x500507630300c562, must also be `0` (see "Displaying port information" on page 203).

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

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

Finding the major and minor numbers for a device

You can find the major and minor numbers of a SCSI device and of SCSI partitions from the device representation in the sysfs SCSI branch.

About this task

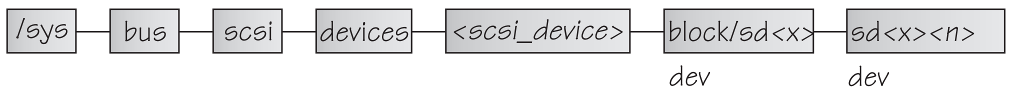


Figure 41. Major and minor numbers for SCSI devices in sysfs

In Figure 41, <*scsi_device*> is the directory that represents a SCSI device (compare Figure 40 on page 209). If the disk is partitioned, the block directory that follows contains directories of the form `sd<x><n>` that represent the partitions. `sd<x>` is a standard name that the SCSI stack assigned to the SCSI device and <*n*> is a positive integer that identifies the partition.

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

Procedure

Issue a command of this form to find the major and minor numbers of a SCSI device:

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

The command output shows the major and minor numbers, which are separated by a colon.

Example

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

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

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

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

Setting the queue depth

The Linux SCSI code automatically adjusts the queue depth as necessary. Changing the queue depth is usually a storage server requirement.

Before you begin

Check the documentation of the storage server that is used or contact your storage server support group to establish if there is a need to change this setting.

About this task

The value of the `zfcp.queue_depth` kernel parameter or the `queue_depth` sysfs attribute (see "zfcp device driver kernel parameters" on page 192) is used as the maximum queue depth of new SCSI devices. You can query the queue depth by issuing a command of this form:

```
# cat /sys/bus/scsi/devices/<SCSI device>/queue_depth
```

Example:

```
# cat /sys/bus/scsi/devices/0:0:19:1086537744/queue_depth
16
```

You can change the maximum queue depth of each SCSI device by writing to the queue_depth attribute, for example:

```
# echo 8 > /sys/bus/scsi/devices/0:0:19:1086537744/queue_depth
# cat /sys/bus/scsi/devices/0:0:19:1086537744/queue_depth
8
```

This method is useful on a running system where you want to make dynamic changes. If you want to make the changes persistent across IPLs, you can:

- Use the kernel or module parameter.
- Write a udev rule to change the setting for each new SCSI device.
- Use the appropriate tool or configuration file that is provided by your distribution.

Linux forwards SCSI commands to the storage server until the number of pending commands exceeds the queue depth. If the server lacks the resources to process a SCSI command, Linux queues the command for a later retry and decreases the queue depth counter. Linux then waits for a defined ramp-up period. If no indications of resource problems occur within this period, Linux increases the queue depth counter until reaching the previously set maximum value. To query the current value for the queue ramp-up period in milliseconds:

```
# cat /sys/bus/scsi/devices/0:0:13:1086537744/queue_ramp_up_period
120000
```

To set a new value for the queue ramp-up period in milliseconds:

```
# echo 1000 > /sys/bus/scsi/devices/0:0:13:1086537744/queue_ramp_up_period
```

Recovering failed SCSI devices

Failed SCSI devices are automatically recovered by the zfcp device driver. You can read the `in_recovery` attribute to check whether recovery is under way.

Before you begin

The FCP device must be online.

Procedure

Perform the following steps:

1. Issue a command of this form:

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

   where the variables have the same meaning as in "Configuring SCSI devices" on page 207.

The value is 1 if recovery is under way and 0 otherwise. If the value is 0 for a non-operational SCSI device, recovery might have failed. Alternatively, the device driver might have failed to detect that the SCSI device is malfunctioning.

2. To find out whether recovery failed, read the `failed` attribute. Issue a command of this form:

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

   The value is 1 if recovery failed, and 0 otherwise.

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

```
# echo 0 > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<fcp_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 and is attached through an FCP device with bus ID 0.0.3d0c. The first command reveals that recovery is not already under way. The second command manually starts recovery for the SCSI device:

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

Updating the information about SCSI devices

Use the `rescan` attribute of the SCSI device to detect changes to a storage device on the storage server that are made after the device was discovered.

Before you begin

The FCP device must be online.

About this task

The initial information about the available SCSI devices is discovered automatically when LUNs first become available.

Procedure

To update the information about a SCSI device issue a command of this form:

```
# echo <string> > /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/rescan
```

where <*string*> is any alphanumeric string and the other variables have the same meaning as in "Mapping the representations of a SCSI device in sysfs" on page 209.

Example

In the following example, the information about a SCSI device 1:0:18:1086537744 is updated:

```
# echo 1 > /sys/bus/scsi/devices/1:0:18:1086537744/rescan
```

Setting the SCSI command timeout

You can change the timeout if the default is not suitable for your storage system.

Before you begin

The FCP device must be online.

About this task

There is a timeout for SCSI commands. If the timeout expires before a SCSI command completes, error recovery starts. The default timeout is 30 seconds.

To find out the current timeout, read the `timeout` attribute of the SCSI device:

```
# cat /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/timeout
```

where the variables have the same meaning as in "Mapping the representations of a SCSI device in sysfs" on page 209.

The attribute value specifies the timeout in seconds.

Procedure

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

```
# echo <timeout> > /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/timeout
```

where <*timeout*> is the new timeout in seconds.

Example

In the following example, the timeout of a SCSI device 1:0:18:1086537744 is first read and then set to 45 seconds:

```
# cat /sys/bus/scsi/devices/1:0:18:1086537744/timeout
30
# echo 45 > /sys/bus/scsi/devices/1:0:18:1086537744/timeout
```

Controlling the SCSI device state

You can use the `state` attribute of the SCSI device to set a SCSI device back online if it was set offline by error recovery.

Before you begin

The FCP device must be online.

About this task

If the connection to a storage system is working but the storage system has a problem, the error recovery might set the SCSI device offline. This condition is indicated by a message like “Device offlined - not ready after error recovery”.

To find out the current state of the device, read the `state` attribute:

```
# cat /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/state
```

where the variables have the same meaning as in “Mapping the representations of a SCSI device in sysfs” on page 209. The state can be:

running
The SCSI device can be used for running regular I/O requests.

cancel The data structure for the device is being removed.

deleted
Follows the `cancel` state when the data structure for the device is being removed.

quiesce
No I/O requests are sent to the device, only special requests for managing the device. This state is used when the system is suspended.

offline
Error recovery for the SCSI device failed.

blocked
Error recovery is in progress and the device cannot be used until the recovery process is completed.

Procedure

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

Issue a command of this form:

```
# echo running > /sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>/state
```

Example

In the following example, SCSI device 1:0:18:1086537744 is offline and is then set online again:

```
# cat /sys/bus/scsi/devices/1:0:18:1086537744/state
offline
# echo running > /sys/bus/scsi/devices/1:0:18:1086537744/state
```

Removing SCSI devices

How to remove a SCSI device depends on whether your environment is set up to use NPIV.

Removing automatically attached SCSI devices

When running with NPIV and the automatic LUN scan, you can temporarily delete a SCSI device by writing 1 to the `delete` attribute of the directory that represents the device in the sysfs SCSI branch.

About this task

See “Mapping the representations of a SCSI device in sysfs” on page 209 about how to find this directory.

Note: These steps delete the SCSI device only temporarily, until the next automatic or user triggered Linux SCSI target scan. The scan automatically adds the SCSI devices again, unless the LUNs were deconfigured on the storage target. To permanently delete SCSI devices, you must disable automatic LUN scannning and configure all LUNs manually, see “Manually configured FCP LUNs and their SCSI devices” on page 207.

Procedure

Issue a command of this form:

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

Example

In this example, an NPIV SCSI device with LUN `0x4010403700000000` is to be removed. Before the device is deleted, the corresponding device in the sysfs SCSI branch is found with an **`lszfcp`** command.

```
# lszfcp -l 0x4010403700000000
0.0.3d0f/0x500507630300c567/0x4010403700000000 0:0:3:1
# echo 1 > /sys/bus/scsi/devices/0:0:3:1/delete
```

Removing manually configured FCP LUNs and their SCSI device

Use the `unit_remove` attribute of the appropriate target port to remove a SCSI device if your environment is not set up to use NPIV or if you disabled automatic LUN scan.

For details about disabling automatic LUN scan, see “zfcp module parameters” on page 194.

Attention: Use this procedure only to dynamically test configuration settings.

Before you begin

This procedure is intended for experimental environments only and might interfere with distribution-specific tools.

See the documentation that is provided by your distributor about which tools to use for a persistent configuration in a production environment.

Procedure

Follow these steps to remove a manually configured FCP LUN and its SCSI device:

1. Optional: To manually unregister the device, write 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 209 for information about how to find this directory. Issue a command of this form:

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

2. Remove the SCSI device from the target port by writing the device's LUN to the port's `unit_remove` attribute. Issue a command of this form:

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

   where the variables have the same meaning as in "Configuring SCSI devices" on page 207. Removing a LUN with unit_remove automatically unregisters the SCSI device first.

Example

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

1. Optionally, unregister the device:

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

2. Remove the device (if not done in previous step) and the LUN:

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

Confirming end-to-end data consistency checking

There are different types of end-to-end data consistency checking, with dependencies on hardware and software.

About this task

End-to-end data consistency checking is based on a data integrity field (DIF) that is added to transferred data blocks. DIF data is used to confirm that a data block originates from the expected source and was not modified during the transfer between the storage system and the FCP device. The SCSI standard defines several types of DIF. Data integrity extension (DIX) builds on DIF to extend consistency checking, for example, to the operating system, middleware, or an application.

You enable the zfcp device driver for end-to-end data consistency checking with the `zfcp.dif=` kernel or `dif=` module parameter (see "Setting up the zfcp device driver" on page 192). With end-to-end data consistency checking enabled, Linux automatically discovers which FCP devices and which SCSI devices support end-to-end data consistency checking. No further setup is required.

Note: SCSI devices for which end-to-end data consistency checking is enabled must be accessed with direct I/O. Direct I/O requires direct access through the block device or through a file system that fully supports end-to-end data consistency checking. For example, XFS provides this support. Expect error messages about invalid checksums when you use other access methods.

The zfcp device driver supports the following modes:

- The FCP device calculates and checks a DIF checksum (DIF type 1)

- The Linux block integrity layer calculates and checks a TCP/IP checksum, which the FCP device then translates to a DIF checksum (DIX type 1 with DIF type 1)

For SCSI devices for which end-to-end data consistency checking is used, there is a sysfs directory

`/sys/block/sd<x>/integrity`

In the path, `sd<x>` is the standard name of the SCSI device.

End-to-end data consistency checking is used only if all of the following components support it:

SCSI disk
: Check your storage server documentation about T10 DIF support and any restrictions.

System z hardware
: System z FCP adapter hardware supports end-to-end data consistency checking as of FICON Express8.

Hypervisor
: For Linux on z/VM, you require a z/VM version with guest support for end-to-end data consistency checking.

FCP device
: Check your FCP adapter hardware documentation about the support and any restrictions. For example, end-to-end data consistency checking might be supported only for disks with 512-byte block size.

Read the `prot_capabilities` sysfs attribute of the SCSI host that is associated with an FCP device to find out about its end-to-end data consistency checking support. The following values are possible:

0 The FCP device does not support end-to-end data consistency checking.

1 The FCP device supports DIF type 1.

16 The FCP device supports DIX type 1.

17 The FCP device supports DIX type 1 with DIF type 1.

Procedure

Issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/host<n>/scsi_host/host<n>/prot_capabilities
```

where *<device_bus_id>* identifies the FCP device and *<n>* is an integer that identifies the corresponding SCSI host.

Example

```
# cat /sys/bus/ccw/devices/0.0.1940/host0/scsi_host/host0/prot_capabilities
17
```

Scenario for finding available LUNs

There are several steps from setting an FCP device online to listing the available LUNs.

Procedure

1. Check for available FCP devices of type 1732/03:

```
# lscss -t 1732/03
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.3c02 0.0.0015  1732/03 1731/03      80  80  ff   36000000 00000000
```

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

2. Set the FCP device online:

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

   A port scan is performed automatically when the FCP device is set online.

3. Optional: Confirm that the FCP device is available and online:

```
# lszfcp -b 0.0.3c02 -a
0.0.3c02 host0
Bus = "ccw"
    availability        = "good"
...
    failed              = "0"
...
    in_recovery         = "0"
...
    online              = "1"
...
```

4. Optional: List the available ports:

```
# lszfcp -P
0.0.3c02/0x50050763030bc562 rport-0:0-0
0.0.3c02/0x500507630310c562 rport-0:0-1
0.0.3c02/0x500507630040727b rport-0:0-10
0.0.3c02/0x500507630e060521 rport-0:0-11
...
```

5. Scan for available LUNs on FCP device 0.0.3c02, port 0x50050763030bc562:

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

API provided by the zfcp HBA API support

The available packages, functions, and tools that are provided by the zfcp HBA API support depend on the installed version.

Programmers: This information is intended for programmers who want to write SAN management clients that run on Linux on System z.

Table 30 on page 222 gives an overview of available packages of zfcp HBA API and the binary files that are installed with those versions.

Table 30. zfcp HBA API library versions

Package	Installed binary files
zfcp HBA API library 2.1	libzfcphbaapi.so
zfcp HBA API library 2.0	libzfcphbaapi.so
zfcp HBA API library 1.4	san_disc command
zfcp HBA API library 1.3	libzfcphbaapi.so, san_disc command
zfcp HBA API library 1.2 and earlier	libzfcphbaapi.so

Functions provided

The zfcp HBA API implements Fibre Channel - HBA API (FC-HBA) functions as defined in the FC-HBA specification.

You can find the FC-HBA specification at www.t11.org. The following functions are available:
- HBA_CloseAdapter()
- HBA_FreeLibrary()
- HBA_GetAdapterAttributes()
- HBA_GetAdapterName()
- HBA_GetAdapterPortAttributes()
- HBA_GetDiscoveredPortAttributes()
- HBA_GetEventBuffer()
- HBA_GetFcpTargetMapping()
- HBA_GetFcpTargetMappingV2()
- HBA_GetNumberOfAdapters()
- HBA_GetRNIDMgmtInfo()
- HBA_GetVersion()
- HBA_LoadLibrary()
- HBA_OpenAdapter()
- HBA_RefreshAdapterConfiguration()
- HBA_RefreshInformation()
- HBA_RegisterForAdapterAddEvents()
- HBA_RegisterForAdapterEvents()
- HBA_RegisterForAdapterPortEvents()
- HBA_RegisterForAdapterPortStatEvents()
- HBA_RegisterForLinkEvents()
- HBA_RegisterForTargetEvents()
- HBA_RegisterLibrary()
- HBA_RegisterLibraryV2()
- HBA_RemoveCallback()
- HBA_SendCTPassThru()
- HBA_SendCTPassThruV2()
- HBA_SendLIRR()
- HBA_SendReadCapacity()
- HBA_SendReportLUNs()
- HBA_SendReportLUNsV2()
- HBA_SendRNID()
- HBA_SendRNIDV()
- HBA_SendRPL()
- HBA_SendRPS()
- HBA_SendScsiInquiry()
- HBA_SendSRL()
- HBA_SetRNIDMgmtInfo()

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

Note: ZFCP HBA API for Linux 3.16 can access only FCP devices, ports, and units that are configured in the operating system.

Tools for investigating your SAN configuration

As of version 2.1, the HBA API package includes the following tools that can help you to investigate your SAN configuration and to solve configuration problems.

zfcp_ping
: to probe a port in the SAN.

zfcp_show
: to retrieve information about the SAN topology and details about the SAN components.

See *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413 for details.

Environment variables

The zfcp HBA API support uses 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 12. Storage-class memory device driver supporting Flash Express

The storage-class memory device driver provides support of Flash Express.

The Flash Express memory is accessed as storage-class memory increments through extended asynchronous data mover (EADM) subchannels. Each increment is represented in Linux by a block device.

What you should know about storage-class memory

Storage-class memory (SCM) is a class of data storage devices that combines properties of both storage and memory.

To access storage-class memory from within an LPAR, one or more increments must be added to the I/O configuration of the LPAR. At least one EADM subchannel must be available to this LPAR. Because SCM supports multiple concurrent I/O requests, it is advantageous to configure multiple EADM subchannels. A typical number of EADM subchannels is 64.

Each increment is available for use through a device node as a block device. You can use the block device with standard Linux tools as you would use any other block device. Commonly used tools that work with block devices include: **fdisk**, **mkfs**, and **mount**.

Storage-class memory is useful for workloads with large write operations, that is, with a block size of 256 KB or more of data. Write operations with a block size of less than 256 KB of data might not perform optimally. Read operations can be of any size.

Storage-class memory device nodes

Applications access storage-class memory devices by device nodes. Normally, your distribution creates a device node for each storage increment. Alternatively, use the **mknod** command to create one.

The device driver uses a device name of the form /dev/scm<*x*> for an entire block device. In the name, <*x*> is one or two lowercase letters.

You can partition a block device into up to seven partitions. If you use partitions, the device driver numbers them from 1 - 7. The partitions then have device nodes of the form /dev/scm<*x*><*n*>, where <*n*> is a number in the range 1 - 7, for example /dev/scma1.

The following example shows two block devices, scma and scmb, where scma has one partition, scma1.

```
# lsblk
NAME     MAJ:MIN RM   SIZE RO MOUNTPOINT
scma     252:0    0    16G  0
`-scma1  252:1    0    16G  0
scmb     252:8    0    16G  0
```

If your distribution provides the storage-class memory device driver as a separate module, be sure to load the module before you check for the device node.

To check whether there already is a node, use for example, **lsblk** to list all block devices and look for "scm" entries.

To create storage-class memory device nodes issue commands of the form:

```
# mknod /dev/scma1 b <major> 1
# mknod /dev/scma2 b <major> 2
# mknod /dev/scma3 b <major> 3
...
```

Building a kernel with the storage-class memory device driver

Control the build options for the storage-class memory device driver through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Figure 42 summarizes the kernel configuration menu options that are relevant to the storage-class memory device driver:

```
I/O subsystem --->
   ...
   SCM bus driver                               (CONFIG_SCM_BUS)
   └─ Support for EADM subchannels              (CONFIG_EADM_SCH)
   ...
Device Drivers --->
   ...
   Block devices --->       (common code option CONFIG_BLK_DEV)
      ...
      Support for Storage Class Memory          (CONFIG_SCM_BLOCK)*
      └─ SCM force cluster writes               (CONFIG_SCM_BLOCK_CLUSTER_WRITE)
```

Figure 42. Storage-class memory kernel configuration menu options

CONFIG_SCM_BUS
is required to enable the storage-class memory bus device driver.

CONFIG_EADM_SCH
is required to enable support for EADM subchannels. Can be compiled into the kernel or as a separate module, eadm_sch.

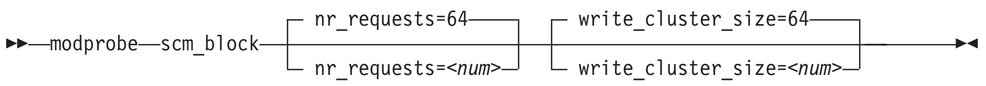
CONFIG_SCM_BLOCK
is required to enable support for storage-class memory block devices. Can be compiled into the kernel or as a separate module, scm_block.

CONFIG_SCM_BLOCK_CLUSTER_WRITE
is required to enable the read-modify-write algorithm of the storage-class memory block driver.

Setting up the storage-class memory device driver

Configure the storage-class memory device driver by using the module parameters.

Storage-class memory module parameter syntax



where

`nr_requests`
specifies the number of parallel I/O requests. Set this number to the number of EADM subchannels. The default is 64.

`write_cluster_size`
specifies the number of pages that are used by the read-modify-write algorithm (available if CONFIG_SCM_BLOCK_CLUSTER_WRITE=y). The default is 64, resulting in that all write requests smaller than 256 KiB are translated to 256 KiB writes. 1 KiB is 1024 bytes.

Working with storage-class memory increments

You can list storage-class memory increments and EADM subchannels.
- "Displaying EADM subchannels"
- "Listing storage-class memory increments" on page 228
- "Combining SCM devices with LVM" on page 228

Displaying EADM subchannels

Use the **`lscss`** command to list EADM subchannels.

About this task

The extended asynchronous data mover (EADM) subchannels are used to transfer data to and from the storage-class memory. At least one EADM subchannel must be available to the LPAR.

Procedure

To list EADM subchannels, issue:

```
# lscss --eadm
Device   Subchan.
-----------------
n/a      0.0.ff00
n/a      0.0.ff01
n/a      0.0.ff02
n/a      0.0.ff03
n/a      0.0.ff04
n/a      0.0.ff05
n/a      0.0.ff06
n/a      0.0.ff07
```

For more information about the **lscss** command, see “lscss - List subchannels” on page 617.

Listing storage-class memory increments

Use the **lsscm** command to see the status and attributes of storage-class memory increments.

About this task

Each storage-class memory increment can be accessed as a block device through a device node `/dev/scm<x>`. Optionally, you can partition a storage-class memory increment in up to seven partitions.

You can also use the **lsblk** command to list all block devices.

Procedure

To list all storage-class memory increments, their status, and attributes, issue:

```
# lsscm
SCM Increment    Size    Name  Rank D_state O_state Pers ResID
--------------------------------------------------------------
0000000000000000 16384MB scma     1       2       1    2     1
0000000400000000 16384MB scmb     1       2       1    2     1
```

See “lsscm - List storage-class memory increments” on page 630 for details about the **lsscm** command.

Combining SCM devices with LVM

You can use LVM to combine multiple SCM block devices into an arbitrary sized LVM device.

Example

Configure SCM as any other block devices in LVM. If your version of LVM does not accept SCM devices as valid LVM device types and issues an error message, add the SCM devices to the LVM configuration file `/etc/lvm/lvm.conf`. Add the following line to the section labeled “devices”:

```
types = [ "scm", 8 ]
```

Chapter 13. Channel-attached tape device driver

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

SCSI tape devices that are attached through an FCP channel are handled by the zfcp device driver (see Chapter 11, “SCSI-over-Fibre Channel device driver,” on page 185).

Features

The tape device driver supports a range of channel-attached tape devices and functions of these devices.

- The tape device driver supports channel-attached tape drives that are compatible with IBM 3480, 3490, 3590, and 3592 magnetic tape subsystems. Various models of these device types are handled (for example, the 3490/10).

  3592 devices that emulate 3590 devices are recognized and treated as 3590 devices.
- Logical character devices for non-rewinding and rewinding modes of operation (see “Tape device modes and logical devices”)
- Control operations through **mt** (see “Using the **mt** command” on page 233)
- Message display support (see “tape390_display - Display messages on tape devices and load tapes” on page 678)
- Encryption support (see “tape390_crypt - Manage tape encryption” on page 674)
- Up to 128 physical tape devices

What you should know about channel-attached tape devices

A naming scheme helps you to keep track of your tape devices, their modes of operation, and the corresponding device nodes.

Tape device modes and logical devices

The tape device driver supports up to 128 physical tape devices. Each physical tape device can be used as a character device in non-rewinding or in rewinding mode.

In non-rewinding mode, the tape remains at the current position when the device is closed. In rewinding mode, the tape is rewound when the device is closed. The tape device driver treats each mode as a separate logical device.

Both modes provide sequential (traditional) tape access without any caching done in the kernel.

You can use a channel-attached tape device in the same way as any other Linux tape device. You can write to it and read from it using standard Linux facilities such as GNU **tar**. You can perform control operations (such as rewinding the tape or skipping a file) with the standard tool **mt**.

Tape naming scheme

The tape device driver assigns minor numbers along with an index number when a physical tape device comes online.

The naming scheme for tape devices is summarized in Table 31.

Table 31. Tape device names and minor numbers

Device	Names	Minor numbers
Non-rewinding character devices	ntibm<*n*>	2×<*n*>
Rewinding character devices	rtibm<*n*>	2×<*n*>+1

where <*n*> is the index number that is assigned by the device driver. The index starts from 0 for the first physical tape device, 1 for the second, and so on. The name space is restricted to 128 physical tape devices, so the maximum index number is 127 for the 128th physical tape device.

The index number and corresponding minor numbers and device names are not permanently associated with a specific physical tape device. When a tape device goes offline, it surrenders its index number. The device driver assigns the lowest free index number when a physical tape device comes online. An index number with its corresponding device names and minor numbers can be reassigned to different physical tape devices as devices go offline and come online.

Tip: Use the **`lstape`** command (see "lstape - List tape devices" on page 633) to determine the current mapping of index numbers to physical tape devices.

When the tape device driver is loaded, it dynamically allocates a major number to channel-attached character tape devices. A different major number might be used when the device driver is reloaded, for example when Linux is rebooted.

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

Each of these directories has a `dev` attribute. The value of the `dev` attribute has the form <*major*>:<*minor*>, where <*major*> is the major number for the device and <*minor*> is the minor number specific to the logical device.

Example

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

```
# lstape --ccw-only
TapeNo  BusID      CuType/Model DevType/DevMod  BlkSize State   Op      MedState
0       0.0.01a1   3490/10      3490/40         auto    UNUSED  ---     UNLOADED
1       0.0.01a0   3480/01      3480/04         auto    UNUSED  ---     UNLOADED
2       0.0.0172   3590/50      3590/11         auto    IN_USE  ---     LOADED
N/A     0.0.01ac   3490/10      3490/40         N/A     OFFLINE ---     N/A
```

Table 32 on page 231 summarizes the resulting names and minor numbers.

Table 32. Example names and minor numbers

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

For the online 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 is 254. The minor numbers are as expected for the respective device names.

Creating device nodes

Applications access tape devices by device nodes. Your distribution might create device nodes for you, but you can also create your own device nodes.

About this task

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

Tip: Use the device names to construct your nodes (see "Tape naming scheme" on page 229).

Example: Defining standard tape nodes

In this example, the tape major number is assumed to be 254. The nodes use the standard form `/dev/<`*`device_name`*`>` for the device nodes and the assignment of minor numbers is according to Table 31 on page 230.

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

Examples for udev-created tape device nodes

If you use udev-created device nodes, be sure that you use the nodes according to your distribution.

Note: The format of the nodes that udev creates for you depends on distribution-specific configuration files in `/etc/udev/rules.d`. The following examples use hypothetical nodes that are provided for illustration purposes only.

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

Typically, udev creates device nodes that are based on the device names. Other udev-created device nodes can be based on device bus-IDs. The device name and, therefore, the standard device node of a device can change when Linux is rebooted. In contrast, the mapping of devices and device nodes that are based on bus-IDs is persistent across reboots. This mapping changes only if you change the bus-IDs of your devices. With these device nodes, udev helps you to reliably address a particular physical device.

The configuration file might instruct udev to create the following two nodes for each logical device:
- The standard node
- A node that is based on the device bus-ID

For a tape device with a device bus-ID 0.0.01ac, it would create four device nodes.

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

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

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

Accessing tapes by bus ID

Use device nodes that are based on device bus-IDs to ensure that you access a tape device with a particular bus ID, regardless of the assigned device name.

Example

This example assumes that udev creates the following device nodes for a device with bus ID `0.0.01ac`:
- `/dev/ntibm0`
- `/dev/tape/0.0.01ac/non-rewinding`

Instead of issuing:

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

issue:

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

Using the `mt` command

There are differences between the MTIO interface for channel-attached tapes and other tape drives. Correspondingly, some operations of the **mt** command are different for channel-attached tapes.

The **mt** command handles basic tape control in Linux. See the man page for general information about **mt**.

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 and unlock
: have no effect because channel-attached tape hardware does not support media locking.

setpartition and 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 and rewoffl and 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 238 for details).

Building a kernel with the tape device driver

Control the build options for the tape device driver through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

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

Figure 43 on page 234 summarizes the kernel configuration menu options that are relevant to the tape device driver:

```
Device Drivers --->
    ...
   Character devices --->
      ...
      S/390 tape device support                          (CONFIG_S390_TAPE)
         --- S/390 tape hardware support (depends on S390_TAPE) ---
         Support for 3480/3490 tape hardware             (CONFIG_S390_TAPE_34XX)
         Support for 3590 tape hardware                  (CONFIG_S390_TAPE_3590)
```

Figure 43. Tape kernel configuration menu options

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

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

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

Loading the tape device driver

If the tape_34xx or tape_3590 device drivers are not built into the kernel, you must load the kernel modules before you can work with the tape devices.

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

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

Tape module syntax

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

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

Working with tape devices

Typical tasks for working with tape devices include displaying tape information, controlling compression, and loading and unloading tapes.

For information about working with the channel measurement facility, see Chapter 44, “Channel measurement facility,” on page 503.

For information about displaying messages on a tape device's display unit, see “tape390_display - Display messages on tape devices and load tapes” on page 678.

See “Working with newly available devices” on page 12 to avoid errors when working with devices that have become available to a running Linux instance.

- “Setting a tape device online or offline”
- “Displaying tape information” on page 236
- “Enabling compression” on page 238
- “Loading and unloading tapes” on page 238

Setting a tape device online or offline

Set a tape device online or offline with the **chccwdev** command or through the `online` sysfs attribute of the device.

About this task

Setting a physical tape device online makes both corresponding logical devices accessible:
- The non-rewind character device
- The rewind character device

At any time, the device can be online to a single Linux instance only. You must set the tape device offline to make it accessible to other Linux instances in a shared environment.

Procedure

Use the **chccwdev** command (see “chccwdev - Set CCW device attributes” on page 543) to set a tape online or offline.
Alternatively, you can write 1 to the online attribute of the device to set it online; or write `0` to set it offline.

Results

When a physical tape device is set online, the device driver assigns an index number to it. This index number is used in the standard device nodes (see “Creating device nodes” on page 231) to identify the corresponding logical devices. The index number is in the range 0 - 127. A maximum of 128 physical tape devices can be online concurrently.

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

Your distribution might use udev to create alternative device nodes that distinguish devices by the physical device's bus ID instead of the index number. If you are using such device nodes, you do not need to know the index number (see “Examples for udev-created tape device nodes” on page 232).

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

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

where <*device_bus_id*> is the device bus-ID that corresponds to the physical tape device. The index number is the value in the TapeNo column of the command output. For more information about the **lstape** command, see “lstape - List tape devices” on page 633.

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 that the tape device driver assigned to the device, issue:

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

  In the example, the assigned index number is `2`. The standard device nodes for working with the device until it is set offline are then:
  - `/dev/ntibm2` for the non-rewinding device
  - `/dev/rtibm2` for the rewinding device
- To set a physical tape device with device bus-ID 0.0.015f offline, issue:

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

  or

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

Displaying tape information

Use the **`lstape`** command to display summary information about your tape devices, or read tape information from sysfs.

Each physical tape device is represented in a sysfs directory of the form
`/sys/bus/ccw/devices/<device_bus_id>`

where <*device_bus_id*> is the device bus-ID that corresponds to the physical tape device. This directory contains a number of attributes with information about the physical device. The attributes: blocksize, state, operation, and medium_state, might not show the current values if the device is offline.

Table 33. Tape device attributes

Attribute	Explanation
online	`1` if the device is online or `0` if it is offline (see “Setting a tape device online or offline” on page 235)
cmb_enable	`1` if channel measurement block is enabled for the physical device or `0` if it is not enabled (see Chapter 44, “Channel measurement facility,” on page 503)
cutype	Type and model of the control unit
devtype	Type and model of the physical tape device
blocksize	Currently used record size in bytes or `0` for auto

Table 33. Tape device attributes (continued)

Attribute	Explanation
state	State of the physical tape device, either of: **UNUSED** The device is not in use and is available to any operating system image in a shared environment. **IN_USE** The device is being used by a process on this Linux instance. **OFFLINE** The device is offline. **NOT_OP** Device is not operational.
operation	The current tape operation, for example: **---** No operation **WRI** Write operation **RFO** Read operation **MSN** Medium sense Several other operation codes exist, for example, for rewind and seek.
medium_state	The current state of the tape cartridge: **1** Cartridge is loaded into the tape device **2** No cartridge is loaded **0** The tape device driver does not have information about the current cartridge state

Procedure

Issue a command of this form to read an attribute:

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

where <*attribute*> is one of the attributes of Table 33 on page 236.

Example

The following **lstape** command displays information about a tape device with bus ID 0.0.015f:

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

This sequence of commands reads the same information from sysfs:

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

Enabling compression

Control Improved Data Recording Capability (IDRC) compression with the **mt** command provided by the RPM mt-st.

About this task

Compression is off after the tape device driver is loaded.

Procedure

To enable compression, issue:

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

or

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

where *<node>* is the device node for a character device, for example, `/dev/ntibm0`. To disable compression, issue:

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

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

Example

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

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

Loading and unloading tapes

Unload tapes with the **mt** command. How to load tapes depends on the stacker mode of your tape hardware.

Procedure

Unload tapes with a command of this form:

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

where <*node*> can be a device node for the non-rewinding device or for the rewinding device.
Whether 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 678) 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 678). 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 display unit of the tape device until the tape device receives the next tape movement command.

Chapter 14. XPRAM device driver

The XPRAM device driver supports System z expanded storage. Expanded storage is designed to overcome memory limitations for the S/390® architecture and for the z/Architecture 31-bit mode.

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

The XPRAM device driver is a block device driver that enables Linux on System z to access expanded storage. Thus, XPRAM can be used as a basis for fast swap devices and for fast file systems. Expanded storage can be swapped in or out of the main storage in 4 KB blocks. All XPRAM devices provide a block size of 4096 bytes.

XPRAM features

The XPRAM device driver automatically detects expanded storage and supports expanded storage partitions.

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

What you should know about XPRAM

There is a device node for each XPRAM partition. Expanded storage persists across reboots and, with suitable boot parameters, the stored data can be accessed by the rebooted Linux instance.

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 34. XPRAM device names, minor numbers, and partitions

Minor	Name	To access
0	slram0	the first partition or the entire expanded storage if there are no partitions
1	slram1	the second partition
2	slram2	the third partition

Table 34. XPRAM device names, minor numbers, and partitions (continued)

Minor	Name	To access
...	...	...
<n>	slram<n>	the (<n>+1)th partition
...	...	...
31	slram31	the 32nd partition

Creating device nodes

User space programs access XPRAM devices by *device nodes*.

About this task

Your distribution might create these device nodes for you or provide udev to create them (see "Device nodes provided by udev" on page 4).

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

Tip: Use the device names to construct your nodes (see "XPRAM partitions and device nodes" on page 241).

Example: Defining standard XPRAM nodes

The nodes use the standard form /dev/*<device_name>* for the device nodes and the assignment of minor numbers is according to Table 34 on page 241.

```
# mknod /dev/slram0 b 35 0
# mknod /dev/slram1 b 35 1
# mknod /dev/slram2 b 35 2
...
# mknod /dev/slram30 b 35 30
# mknod /dev/slram31 b 35 31
```

XPRAM use for diagnosis

Expanded storage persists across reboots, which makes it suitable for storing diagnostic information.

Issuing an IPL command to reboot Linux on System z does not reset expanded storage. Expanded storage is persistent across IPLs and can be used, for example, to store diagnostic information. The expanded storage is reset when the z/VM guest virtual machine is logged off or when the LPAR is deactivated.

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 or a new swap device for each changed partition. A device or partition is considered changed if its size has changed. All partitions that follow a changed partition are also considered changed even if their sizes are unchanged.

Building a kernel with the XPRAM device driver

To build a kernel with XPRAM support, you must select option CONFIG_BLK_DEV_XPRAM in the configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Figure 44 shows where to find CONFIG_BLK_DEV_XPRAM in the configuration menu.

```
Device Drivers --->
   ...
   Block devices --->
      ...
      XPRAM disk support                  (CONFIG_BLK_DEV_XPRAM)
```

Figure 44. XPRAM kernel configuration menu option

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

Setting up the XPRAM device driver

You use a kernel or module parameter to set up XPRAM partitions.

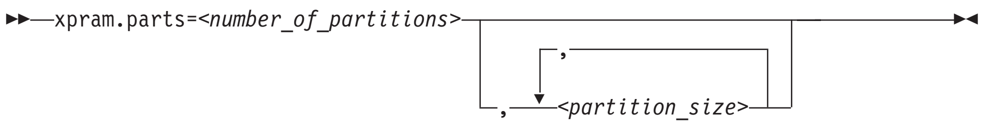
The syntax is different for the kernel parameters and the corresponding module parameters. By default the entire expanded storage is treated as a single partition.

See "Creating device nodes" on page 242 for information about the device nodes that you need to access the partitions.

Kernel parameters

If the XPRAM device driver has been compiled into the kernel, you can optionally partition the available expanded storage by adding the `xpram.parts` kernel parameters to the kernel parameter line.

XPRAM kernel parameter syntax



where:

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

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

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

Any remaining memory is divided as equally as possible among any partitions with a zero or blank size parameter. Dividing the remaining memory is subject to the following constraints:
- Blocks must be allocated in multiples of 4 KB.
- Addressing constraints might leave un-allocated areas of memory between partitions.

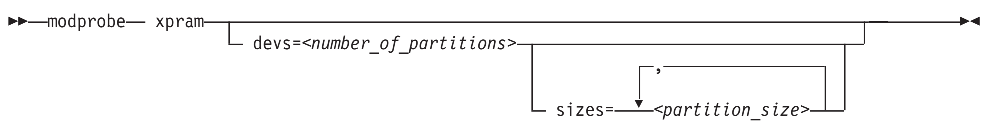
Examples

- The following specification allocates the extended storage into four partitions. Partition 1 has 2 GB (hex 800M), partition 4 has 4 GB, and partitions 2 and 3 use equal parts of the remaining storage. If the total amount of extended storage was 16 GB, then partitions 3 and 4 would each have approximately 5 GB.
  ```
  xpram.parts=4,0x800M,0,0,4g
  ```
- The following specification allocates the extended storage into three partitions. The partition 2 has 512 KB and the partitions 1 and 3 use equal parts of the remaining storage.
  ```
  xpram.parts=3,,512
  ```
- The following specification allocates the extended storage into two partitions of equal size.
  ```
  xpram.parts=2
  ```

Module parameters

If the XPRAM device driver has been built as a separate module, you can optionally partition the available expanded storage by using the `devs` and `sizes` module parameters when loading the module.

XPRAM module parameter syntax



where:

<number_of_partitions>
: is an integer in the range 1 - 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 fewer values than *<number_of_partitions>*, the missing values are interpreted as blanks. Blanks are treated like zeros.

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

Any remaining memory is divided as equally as possible among any partitions with a zero or blank size parameter. Dividing the remaining memory is subject to the following constraints:
- Blocks must be allocated in multiples of 4 KB.
- Addressing constraints might 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
```

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

Part 4. Networking

There are several System z specific network device drivers for Linux on System z.

Newest version

You can find the newest version of this publication at
www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Restrictions

For prerequisites and restrictions see
www.ibm.com/developerworks/linux/linux390/development_technical.html
www.ibm.com/developerworks/linux/linux390/development_restrictions.html

Example

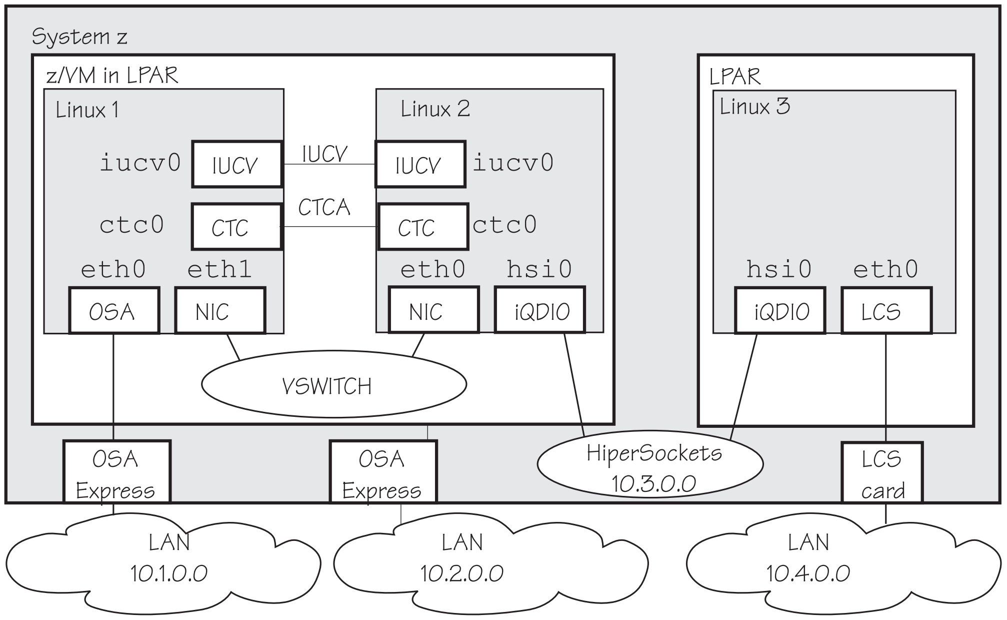


Figure 45. Networking example

In the example there are three Linux instances; two of them run as z/VM guests in one LPAR and a third Linux instance runs in another LPAR. Within z/VM, Linux instances can be connected directly by IUCV, virtual-CTC, or through a guest LAN or VSWITCH. Within and between LPARs, you can connect Linux instances through HiperSockets. OSA-Express cards running in either non-QDIO mode (called LCS here) or in QDIO mode can connect the System z mainframe to an external network.

Table 35 lists which control units and device type combinations are supported by the network device drivers.

Table 35. Supported device types, control units, and corresponding device drivers

Device type	Control unit	Device driver	Comment
1732/01	1731/01	qeth	OSA configured as OSD
1732/02	1731/02	qeth	OSA configured as OSX
1732/03	1731/02	qeth	OSA configured as OSM
1732/05	1731/05	qeth	HiperSockets
1732/06	1731/06	qeth	OSA configured as OSN
0000/00	3088/01	lcs	P/390
0000/00	3088/08	ctcm	Virtual CTC under z/VM
0000/00	3088/1e	ctcm	FICON channel
0000/00	3088/1f	lcs	2216 Nways Multiaccess Connector
0000/00	3088/1f	ctcm	ESCON channel
0000/00	3088/60	lcs	OSA configured as OSE (non-QDIO)
0000/00	3088/61	claw	Common link access to workstation

Chapter 15. qeth device driver for OSA-Express (QDIO) and HiperSockets

The qeth device driver supports a multitude of network connections, for example, connections through Open Systems Adapters (OSA), HiperSockets, guest LANs, and virtual switches.

Real connections that use OSA-Express

A System z mainframe offers OSA-Express adapters, which are real LAN-adapter hardware, see Figure 46. These adapters provide connections to the outside world, but can also connect virtual systems (between LPARs or between z/VM guest virtual machines) within the mainframe. The qeth driver supports these adapters if they are defined to run in queued direct I/O (QDIO) mode (defined as OSD or OSN in the hardware configuration). OSD-devices are the standard System z LAN-adapters, while OSN-devices serve as NCP-adapters. For details about OSA-Express in QDIO mode, see *OSA-Express Customer's Guide and Reference*, SA22-7935.

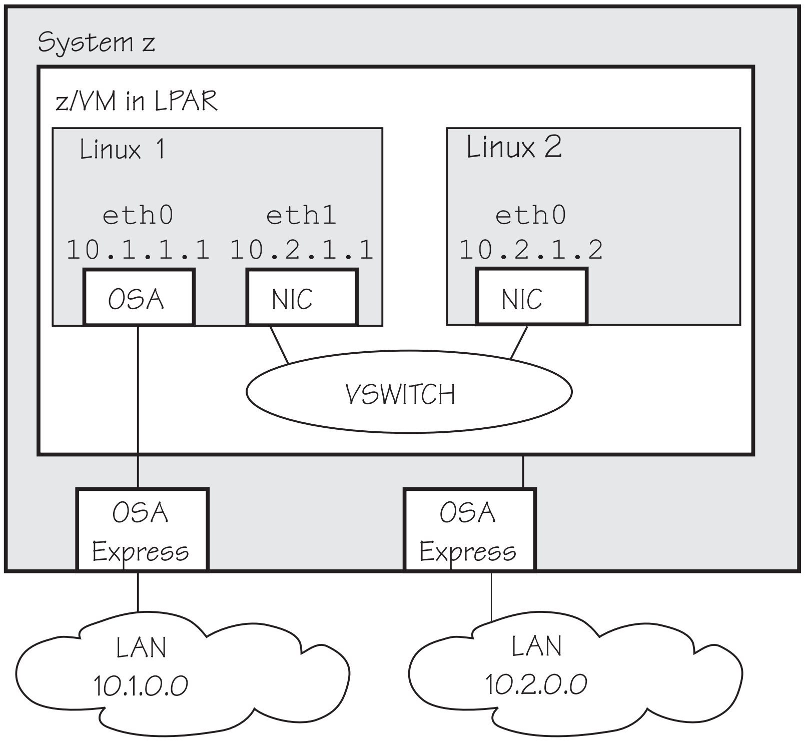


Figure 46. OSA-Express adapters are real LAN-adapter hardware

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

As of zEnterprise, the qeth device driver supports CHPIDs of type OSM and OSX:

OSM provides connectivity to the intranode management network (INMN) from Unified Resource Manager functions to a zEnterprise CPC.

OSX provides connectivity to and access control for the intraensemble data network (IEDN), which is managed by Unified Resource Manager functions. A zEnterprise CPC and zBX within an ensemble are connected through the IEDN. See *zEnterprise System Introduction to Ensembles*, GC27-2609 and *zEnterprise System Ensemble Planning and Configuring Guide*, GC27-2608 for more details.

HiperSockets

A System z mainframe offers internal connections that are called *HiperSockets*. These simulate QDIO network adapters and provide high-speed TCP/IP communication for operating system instances within and across LPARs. For details about HiperSockets, see *HiperSockets Implementation Guide*, SG24-6816.

Virtual connections for Linux on z/VM

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

Guest LAN

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

- Simulated OSA in layer 3 mode
- Simulated HiperSockets (layer 3) mode
- Simulated OSA in layer 2 mode

Each guest LAN is isolated from other guest LANs on the same system (unless some member of one LAN group acts as a router to other groups). See Figure 47 on page 253.

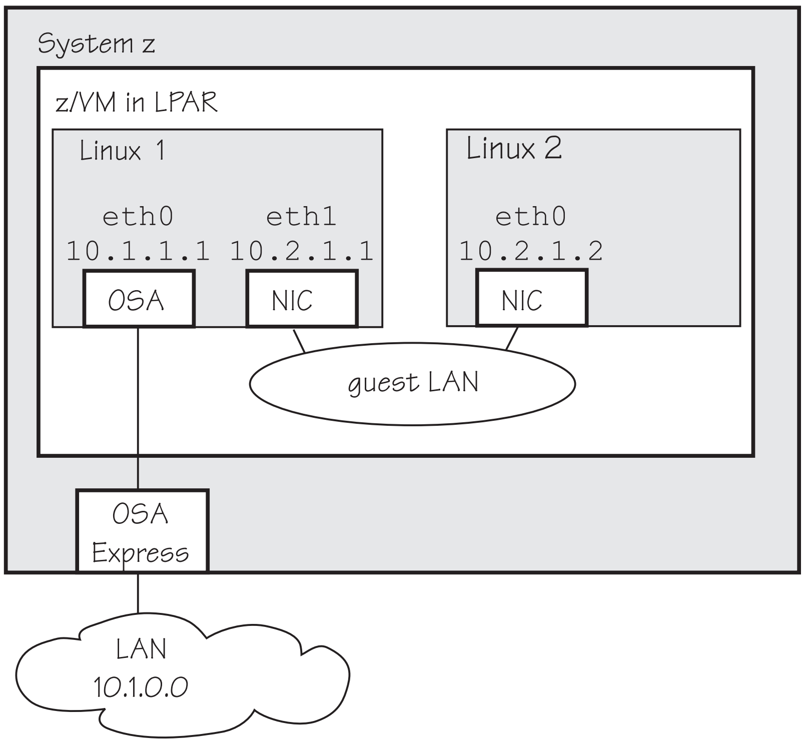


Figure 47. Guest LAN

Virtual switch
A virtual switch (VSWITCH) is a special-purpose guest LAN that provides external LAN connectivity through an additional OSA-Express device served by z/VM without the need for a routing virtual machine, see Figure 48.

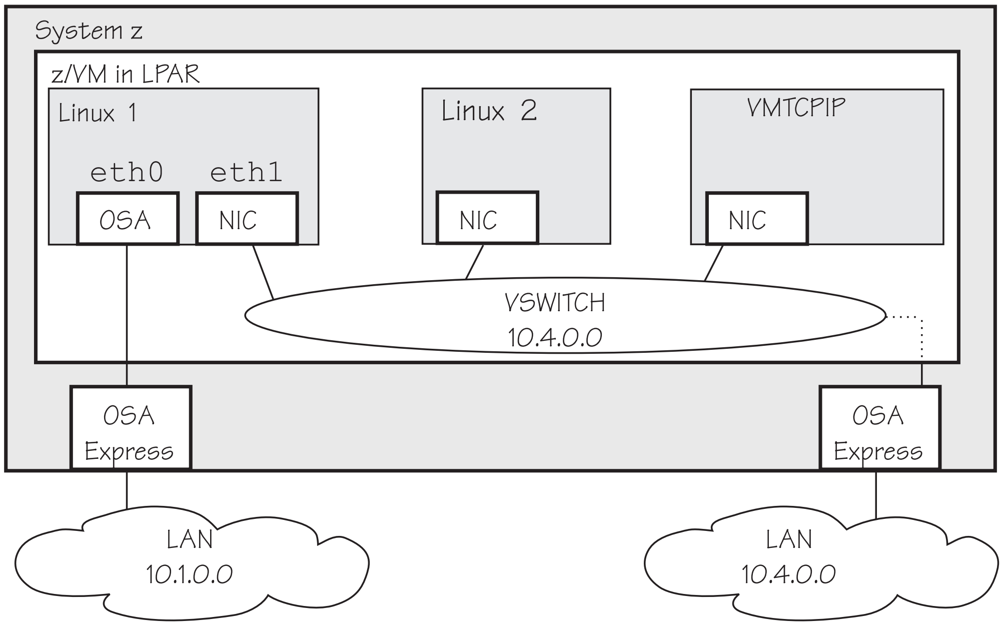


Figure 48. Virtual switch

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

The qeth device driver distinguishes between virtual NICs in QDIO mode or HiperSockets mode. It cannot detect whether the virtual network is a guest LAN or a VSWITCH.

HiperSockets bridge port
A HiperSockets bridge port connects a network defined by a virtual switch to a HiperSockets LAN. The two networks are combined into one logical network. If the VSWITCH is connected to an external Ethernet LAN, the HiperSockets LAN can then communicate outside the CEC as shown in Figure 49. You can thus connect a HiperSockets LAN to an external LAN without using a router.

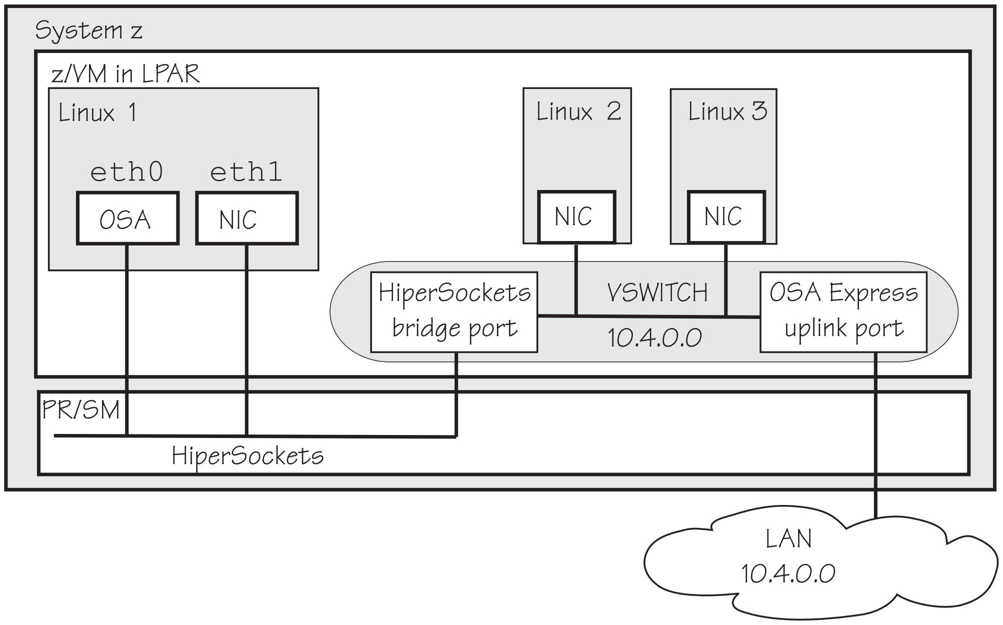


Figure 49. HiperSockets bridge port

For information about guest LANs, virtual switches, bridge ports and virtual HiperSockets, see *z/VM Connectivity*, SC24-6174.

The qeth network device driver supports the System z OSA-Express4S, OSA-Express3, OSA-Express2, and OSA-Express features and HiperSockets as shown in Table 36 and Table 37 on page 255:

Table 36. The qeth device driver support for HiperSockets and OSA-Express features - Part 1: zEnterprise CPCs

Feature	zEC12 and zBC12	z196 and z114
HiperSockets	Yes	Yes
OSA-Express4S	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet
OSA-Express3	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet

Table 36. The qeth device driver support for HiperSockets and OSA-Express features - Part 1: zEnterprise CPCs (continued)

Feature	zEC12 and zBC12	z196 and z114
OSA-Express2	Not supported	Gigabit Ethernet 1000Base-T Ethernet
OSA-Express	Not supported	Not supported

Table 37. The qeth device driver support for HiperSockets and OSA-Express features - Part 2: Earlier System z mainframes

Feature	System z10	System z9	eServer™ zSeries
HiperSockets	Yes	Yes (layer 3 only)	Yes (layer 3 only)
OSA-Express4S	Not supported	Not supported	Not supported
OSA-Express3	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Not supported	Not supported
OSA-Express2	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet	Not supported
OSA-Express	Not supported	Fast Ethernet Gigabit Ethernet 1000Base-T Ethernet	Fast Ethernet Gigabit Ethernet 1000Base-T Ethernet ATM (z900, needs RPQ)

Note: Unless otherwise indicated, OSA-Express refers to OSA-Express4S, OSA-Express3, OSA-Express2, and OSA-Express.

Device driver functions

The qeth device driver supports many networking transport protocol functions, as well as offload functions and problem determination functions.

The qeth device driver supports functions that are listed in Table 38 and Table 39 on page 257.

Table 38. Real connections

Function	OSA Layer 2	OSA Layer 3	HiperSockets Layer 2 Ethernet	HiperSockets Layer 3 Ethernet
Basic device or protocol functions				
IPv4/multicast/broadcast	Yes/Yes/Yes	Yes/Yes/Yes	Yes/Yes/Yes	Yes/Yes/Yes
IPv6/multicast	Yes/Yes	Yes/Yes	Yes/Yes	Yes/Yes
Non-IP traffic	Yes	Yes	Yes	No
VLAN IPv4/IPv6/non IP	sw/sw/sw	hw/sw/sw	sw/sw/sw	hw/sw/No
Linux ARP	Yes	No (hw ARP)	Yes	No
Linux neighbor solicitation	Yes	Yes	Yes	No
Unique MAC address	Yes (random)	No	Yes	Yes
Change MAC address	Yes	No	Yes	No

Table 38. Real connections (continued)

Function	**OSA Layer 2**	**OSA Layer 3**	**HiperSockets Layer 2 Ethernet**	**HiperSockets Layer 3 Ethernet**
Basic device or protocol functions				
Promiscuous mode	No	No	No	• Yes (for sniffer=1) • No (for sniffer=0)
MAC headers send/receive	Yes/Yes	faked/faked	Yes/Yes	faked/faked
ethtool support	Yes	Yes	Yes	Yes
Bonding	Yes	No	Yes	No
Priority queueing	Yes	Yes	Yes	Yes
Offload features				
TCP segmentation offload (TSO)	No	Yes	No	No
Inbound (rx) checksum	No	Yes	No	No
Outbound (tx) checksum	No	Yes	No	No
OSA/QETH specific features				
Special device driver setup for VIPA	No	required	No	Yes
Special device driver setup for proxy ARP	No	required	No	Yes
Special device driver setup for IP takeover	No	required	No	Yes
Special device driver setup for routing IPv4/IPv6	No/No	required/ required	No/No	Yes/Yes
Receive buffer count	Yes	Yes	Yes	Yes
Direct connectivity to z/OS	Yes by HW	Yes	no	Yes
SNMP support	Yes	Yes	No	No
Multiport support	Yes	Yes	No	No
Data connection isolation	Yes	Yes	No	No
Problem determination				
Hardware trace	No	Yes	No	No

Legend:

No	The function is not supported or not required.
Yes	The function is supported.
hw	The function is performed by hardware.
sw	The function is performed by software.
faked	The function is simulated.
required	The function requires special setup.

Table 39. Virtual NICs coupled to a z/VM VSWITCH or guest LAN

Function	Emulated OSA Layer 2	Emulated OSA Layer 3	Emulated HiperSockets Layer 3
Basic device or protocol features			
IPv4/multicast/broadcast	Yes/Yes/Yes	Yes/Yes/Yes	Yes/Yes/Yes
IPv6/multicast	Yes/Yes	Yes/Yes	No/No
Non-IP traffic	Yes	No	No
VLAN IPv4/IPv6/non IP	sw/sw/sw	hw/sw/No	hw/No/No
Linux ARP	Yes	No (hw ARP)	No
Linux neighbor solicitation	Yes	Yes	No
Unique MAC address	Yes	No	Yes
Change MAC address	Yes	No	No
Promiscuous mode	Yes	Yes	No
MAC headers send/receive	Yes/Yes	faked/faked	faked/faked
ethtool support	Yes	Yes	Yes
Bonding	Yes	No	No
Priority queueing	Yes	Yes	Yes
Offload features	No	No	No
OSA/QETH specific features			
Special device driver setup for VIPA	No	required	required
Special device driver setup for proxy ARP	No	required	required
Special device driver setup for IP takeover	No	required	required
Special device driver setup for routing IPv4/IPv6	No/No	required/required	required/required
Receive buffer count	Yes	Yes	Yes
Direct connectivity to z/OS	No	Yes	Yes
SNMP support	No	No	No
Multiport support	No	No	No
Data connection isolation	No	No	No
Problem determination			
Hardware trace	No	No	No

Legend:

No	The function is not supported or not required.
Yes	The function is supported.
hw	The function is performed by hardware.
sw	The function is performed by software.
faked	The function is simulated.
required	The function requires special setup.

What you should know about the qeth device driver

Interface names are assigned to qeth group devices, which map to subchannels and their corresponding device numbers and device bus-IDs. An OSA-Express adapter can handle both IPv4 and IPv6 packets.

Layer 2 and layer 3

The qeth device driver consists of a common core and two device disciplines: layer 2 and layer 3.

In layer 2 mode, OSA routing to the destination Linux instance is based on MAC addresses. A local MAC address is assigned to each interface of a Linux instance and registered in the OSA Address Table. These MAC addresses are unique and different from the MAC address of the OSA adapter. See “MAC headers in layer 2 mode” on page 261 for details.

In layer 3 mode, all interfaces of all Linux instances share the MAC address of the OSA adapter. OSA routing to the destination Linux instance is based on IP addresses. See “MAC headers in layer 3 mode” on page 262 for details.

The layer 2 discipline (qeth_l2)

The layer 2 discipline supports:

- OSA devices and z/VM virtual NICs that couple to VSWITCHes or QDIO guest LANs
- OSA devices for NCP
- HiperSockets devices (as of System z10)
- OSM (OSA-Express for Unified Resource Manager) devices
- OSX (OSA-Express for zBX) devices for IEDN

The layer 2 discipline is the default setup for OSA. On HiperSockets the default continues to be layer 3. See “Setting the layer2 attribute” on page 272 for details.

The network device in Linux must use the same layer as the VSWITCH or QDIO guest LAN in z/VM. By default, the qeth device driver uses layer 2. If the coupled VSWITCH or QDIO guest LAN uses layer 3, you must adapt the layer setting in Linux.

The layer 3 discipline (qeth_l3)

The layer 3 discipline supports:

- OSA devices and z/VM virtual NICs that couple to VSWITCHes or QDIO guest LANs running in layer 3 mode (with faked link layer headers)
- HiperSockets and HiperSockets guest LAN devices that are running in layer 3 mode (with faked link layer headers)
- OSX (OSA-Express for zBX) devices for IEDN

This discipline supports those devices that are not capable of running in layer 2 mode. Not all Linux networking features are supported and others need special setup or configuration. See Table 44 on page 269. Some performance-critical applications might benefit from being layer 3.

Layer 2 and layer 3 interfaces cannot communicate within a HiperSockets LAN or within a VSWITCH or guest LAN. However, a shared OSA adapter can convert traffic between layer 2 and layer 3 networks.

qeth group devices

The qeth device driver requires three I/O subchannels for each HiperSockets CHPID or OSA-Express CHPID in QDIO mode. One subchannel is for control reads, one for control writes, and the third is for data.

The qeth device driver uses the QDIO protocol to communicate with the HiperSockets and OSA-Express adapter (see Figure 50).

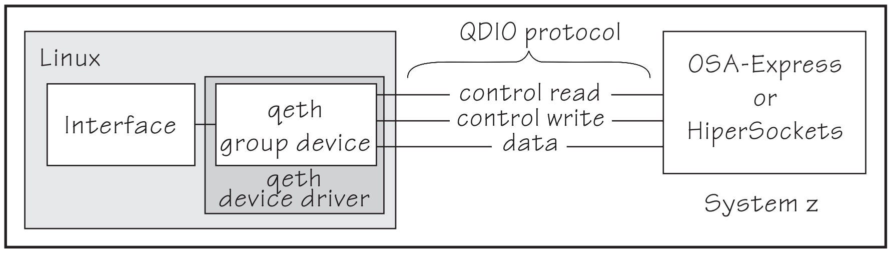


Figure 50. I/O subchannel interface

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

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

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

Overview of the steps for setting up a qeth group device

You must perform several steps before user-space applications on your Linux instance can use a qeth group device.

Before you begin

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

Procedure

Perform these steps to allow user-space applications on your Linux instance to use a qeth group device:

1. Create the qeth group device.

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

What to do next

These tasks and the configuration options are described in detail in "Working with qeth devices" on page 267.

qeth interface names and device directories

The qeth device driver automatically assigns interface names to the qeth group devices and creates the corresponding sysfs structures.

According to the type of CHPID and feature used, the naming scheme uses the following base names:

eth<*n*>
: for Ethernet features (including the OSA-Express ATM device when it emulates Ethernet in QDIO mode).

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". 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, assume that the first qeth Ethernet feature is set online and there already is one LCS Ethernet feature online. Then the LCS feature is named "eth0" and the qeth feature is named "eth1". See also "LCS interface names" on page 330.

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 279 and "Finding out the bus ID of a qeth interface" on page 280 provide information about mapping device bus-IDs and interface names.

Support for IP Version 6 (IPv6)

The qeth device driver supports IPv6 in many network setups.

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

- z/VM virtual NIC interfaces (VSWITCHES and guest LANs) running in layer 2 mode.

IPv6 is not supported on the ATM feature.

There are noticeable differences between the IP stacks for versions 4 and 6. Some concepts in IPv6 are different from IPv4, such as neighbor discovery, broadcast, and Internet Protocol security (IPsec). IPv6 uses a 16-byte address field, while the addresses under IPv4 are 4 bytes in length.

Stateless autoconfiguration generates unique IP addresses for all Linux instances, even if they share an OSA-Express adapter with other operating systems.

Be aware of the IP version when you specify IP addresses and when you use commands that return IP version-specific output (such as **qetharp**).

MAC headers in layer 2 mode

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

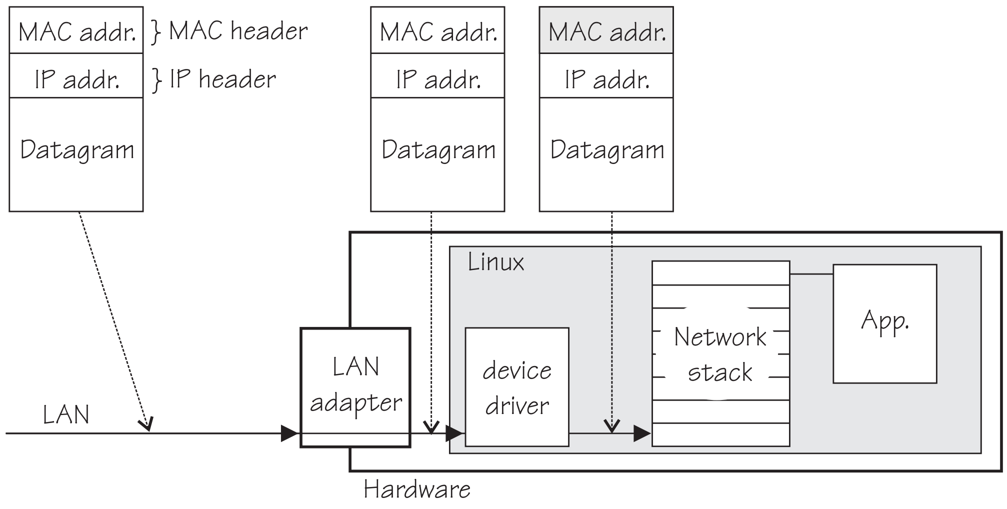


Figure 51. Standard IPv4 processing

MAC address handling as shown in Figure 51 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 51. This layer2-based forwarding requires unique MAC addresses for all concerned Linux instances.

In layer 2 mode, the Linux TCP/IP stack has full control over the MAC headers and the neighbor lookup. The Linux TCP/IP stack does not configure IPv4 or IPv6 addresses into the hardware, but requires a unique MAC address for the card. Users working with a directly attached OSA adapter should assign a unique MAC-address themselves.

For Linux instances that are directly attached to an OSA-Express adapter in QDIO mode, you should assign the MAC addresses yourself. You can change it by issuing the command:

```
ip link set addr <MAC address> dev <interface>
```

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

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

For HiperSockets connections, a MAC address is generated.

For connections within a QDIO-based z/VM VSWITCH or guest LAN environment, z/VM assigns the necessary MAC addresses to the virtual NICs.

MAC headers in layer 3 mode

A qeth layer 3 mode device driver is an Ethernet offload engine for IPv4, and a partial Ethernet offload engine for IPv6. Hence, there are some special things to understand about the layer 3 mode.

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

In layer 3 mode, the OSA-Express adapter in QDIO mode removes the MAC header with the MAC address from incoming IPv4 packets. It uses the registered IP addresses to forward a packet to the recipient TCP/IP stack. See Figure 52. Thus the OSA-Express adapter is able to deliver IPv4 packets to the correct Linux instances. Apart from broadcast packets, a Linux instance can get packets only for IP addresses it configured in the stack and registered with the OSA-Express adapter.

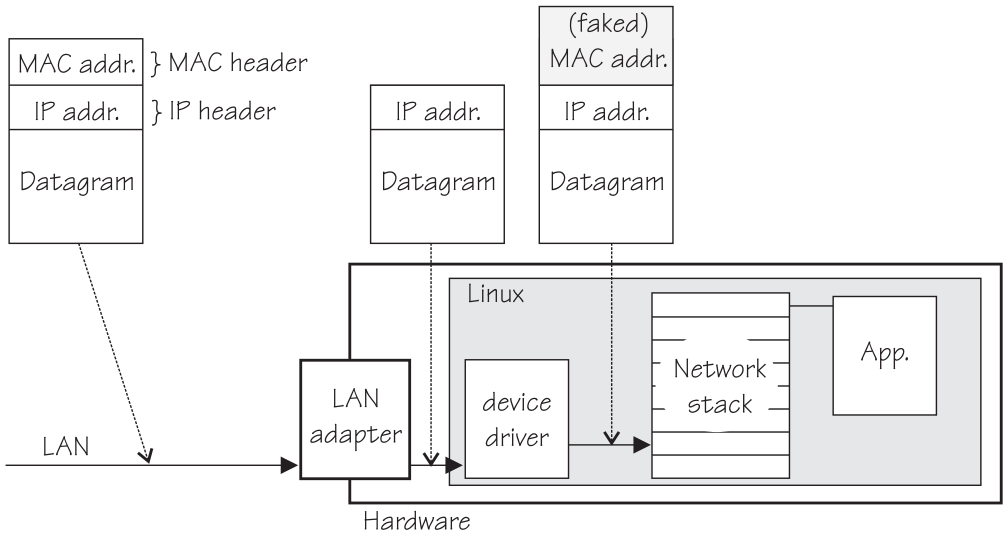


Figure 52. MAC address handling in layer3 mode

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

This lack of MAC headers can be a problem for applications that expect MAC headers. For examples of how such problems can be resolved, see "Setting up for DHCP with IPv4" on page 314.

Outgoing frames

The qeth device driver registers the layer 3 card as an Ethernet device. Therefore, the Linux TCP/IP stack will provide complete Ethernet frames to the device driver.

If the hardware does not require the Ethernet frame (for example, for IPv4) the driver removes the Ethernet header prior to sending the frame to the hardware. If necessary information like the Ethernet target address is not available (because of the offload functionality) the value is filled with the hardcoded address FAKELL.

Table 40. Ethernet addresses of outgoing frames

Frame	Destination address	Source address
IPv4	FAKELL	Real device address
IPv6	Real destination address	Real device address
Other packets	Real destination address	Real device address

Incoming frames

The device driver provides Ethernet headers for all incoming frames.

If necessary information like the Ethernet source address is not available (because of the offload functionality) the value is filled with the hardcoded address FAKELL.

Table 41. Ethernet addresses of incoming frames

Frame	Destination address	Source address
IPv4	Real device address	FAKELL
IPv6	Real device address	FAKELL
Other packets	Real device address	Real source address

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

While the faked headers are syntactically correct, the addresses are not authentic, and hence applications requiring authentic addresses will not work. Some examples are given in Table 42.

Table 42. Applications that react differently to faked headers

Application	Support	Reason
tcpdump	Yes	Displays only frames, fake Ethernet information is displayed.
iptables	Partially	As long as the rule does not deal with Ethernet information of an IPv4 frame.
dhcp	Yes	Is non-IPv4 traffic. (Note that DHCP does not work for Layer 3 HiperSockets.)

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.

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

HiperSockets layer 2 bridge port functionality

System z HiperSockets ports can be set up to receive all frames addressed to unknown MAC addresses.

When a HiperSockets port is configured as a member of a Linux software bridge, it must be set up to receive all frames addressed to unknown MAC addresses. For information about how to set up a software bridge, see the documentation that is provided by your distributor, or the bridging how-to available at: http://www.tldp.org/HOWTO/BRIDGE-STP-HOWTO/.

Permission to configure ports as bridge ports must be granted in IBM zEnterprise Unified Resource Manager (zManager). Multiple ports on one HiperSockets LAN can be configured as bridge ports, but only one is in the active state at any given time.

Bridge port functionality of a HiperSockets port is controlled by the following sysfs attributes in the `/sys/bus/ccwgroup/drivers/qeth/<device bus-ID>` directory:

bridge_role

Read-write attribute that controls the role of the port. Valid values are:

primary
Makes the HiperSockets device a primary bridge port.

secondary
Makes the HiperSockets device a secondary bridge port.

none Clears any bridge port role.

bridge_state
Read-only attribute that shows the state of the port. Valid values are:

active The HiperSockets device is configured as a bridge port, and is assigned the active role by the system. The device receives frames addressed to unknown MAC addresses.

standby
The HiperSockets device is set as a bridge port, but is not currently assigned the active role by the system. The device does not receive frames destined to unknown MAC addresses.

inactive
The HiperSockets device is not a bridge port.

bridge_hostnotify
Read-write attribute that controls the sending of notifications for the HiperSockets device. When you enable notifications (even if notifications were already enabled), udev events are emitted for all currently connected communication peers in quick succession. After that, a udev event is emitted every time a communication peer is connected, or a previously connected peer is disconnected. Any user space program that monitors these events must re-populate its list of registered peers every time the status of the bridge port device changes to enable notifications.

Valid values are:

1 The HiperSockets device is set to send notifications.

0 Notifications are disabled.

Notifications about the change of the state of bridge ports, and (if enabled) about registration and deregistration of communication peers on the HiperSockets LAN are delivered as udev events. The events are described in the file `Documentation/s390/qeth.txt` in the Linux kernel source tree.

Building a kernel with the qeth device driver

Control the build options for the qeth device driver through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

There are options that you must select in the Linux configuration menu to include the qeth device driver.

Figure 53 summarizes the kernel configuration menu options that are relevant to the qeth device driver:

```
I/O subsystem --->
   QDIO support                                              (CONFIG_QDIO)
   ...
Device Drivers --->
   ...
    Network device support --->           (common code option CONFIG_NETDEVICES)
      ...
      S/390 network device drivers (depends on NETDEVICES && S390) --->
         ...
         Gigabit Ethernet device support                     (CONFIG_QETH)*
         ├─ qeth layer 2 device support                      (CONFIG_QETH_L2)
         └─ qeth layer 3 device support                      (CONFIG_QETH_L3)
```

Figure 53. qeth kernel configuration menu options

CONFIG_QDIO
: This option provides the Queued Direct I/O base support for IBM System z.

  It is required if you want to work with qeth devices. It can be compiled into the kernel or as a separate module, qdio.

CONFIG_QETH
: This option is required if you want to work with qeth devices. It can be compiled into the kernel or as a separate module, qeth. This option depends on CCW, the common code option NETDEVICES, "IP: multicasting"(CONFIG_IP_MULTICAST), and "Networking support" (CONFIG_QDIO).

CONFIG_QETH_L2
: Select this option to be able to run qeth devices in layer 2 mode. It can be compiled into the kernel or as a separate module, qeth_l2. This option depends on "Gigabit Ethernet device support" (CONFIG_QETH).

CONFIG_QETH_L3
: Select this option to be able to run qeth devices in layer 3mode. It can be compiled into the kernel or as a separate module, qeth_l3.

  This option depends on "Gigabit Ethernet device support" (CONFIG_QETH).

Setting up the qeth device driver

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

Loading the qeth device driver modules

If the qeth device driver was not built into the kernel, you must load it before you can work with qeth devices.

Use the **modprobe** command to load the qeth device driver, and to automatically load all required additional modules in the correct order:

qeth module syntax

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

where:

qeth
is the core module that contains common functions that are used for both layer 2 and layer 3 disciplines.

qeth_l2
is the module that contains layer 2 discipline-specific code.

qeth_l3
is the module that contains layer 3 discipline-specific code.

When a qeth device is configured for a particular discipline, the driver tries to automatically load the corresponding discipline module. Automatic loading requires that automatic kernel module loading is enabled in the distribution.

Switching the discipline of a qeth device

To switch the discipline of a device, the network interface must be shut down and the device must be offline.

If the new discipline is accepted by the device driver, the old network interface is deleted. When the new discipline is set online the first time, the new network interface is created.

Removing the modules

Removing a module is not possible if there are cross dependencies between the discipline modules and the core module.

To release the dependencies from the core module to the discipline module, all devices of this discipline must be ungrouped. Now the discipline module can be removed. If all discipline modules are removed, the core module can be removed.

Working with qeth devices

Typical tasks for working with qeth devices include creating group devices, finding out the type of a network adapter, and setting a device online or offline.

About this task

Most of these tasks involve writing to and reading from attributes of qeth group devices in sysfs. Using sysfs is useful on a running system where you want to make dynamic changes. If you want to make the changes persistent across IPLs, your distribution might provide a configuration tool for this task.

Table 43 on page 268 and Table 44 on page 269 serve as both a task overview and a summary of the attributes and the possible values you can write to them. Underlined values are defaults.

Tip: Use the **`znetconf`** command to configure devices instead of using the attributes directly (see “znetconf - List and configure network devices” on page 699).

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

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

Table 43. qeth tasks and attributes common to layer2 and layer3

Task	**Corresponding attributes**	**Possible attribute values**
“Creating a qeth group device” on page 270	group	n/a
“Removing a qeth group device” on page 271	ungroup	0 or 1
“Setting the layer2 attribute” on page 272	layer2	0 or 1, see “Layer 2 and layer 3” on page 258[1]
“Assigning a port name” on page 273	portname	any valid port name
| “Using priority queueing” on page 274	priority_queueing	prio_queueing_vlan prio_queueing_skb prio_queueing_prec prio_queueing_tos no_prio_queueing no_prio_queueing:0 no_prio_queueing:1 no_prio_queueing:2 no_prio_queueing:3
“Specifying the number of inbound buffers” on page 276	buffer_count	integer in the range 8 - 128. The default is 64 for OSA devices and 128 for HiperSockets devices
“Finding out the maximum frame size” on page 277	inbuf_size	n/a, read-only
“Specifying the relative port number” on page 277	portno	integer, either 0 or 1, the default is 0
“Finding out the type of your network adapter” on page 278	card_type	n/a, read-only
“Setting a device online or offline” on page 279	online	0 or 1
“Finding out the interface name of a qeth group device” on page 279	if_name	n/a, read-only
“Finding out the bus ID of a qeth interface” on page 280	none	n/a
“Activating an interface” on page 280	none	n/a
“Deactivating an interface” on page 282	none	n/a
“Recovering a device” on page 283	recover	1
“Isolating data connections” on page 283	isolation	none, drop, forward
“Starting and stopping collection of QETH performance statistics” on page 285	performance_stats	0 or 1

Table 43. qeth tasks and attributes common to layer2 and layer3 (continued)

Task	Corresponding attributes	Possible attribute values
"Capturing a hardware trace" on page 286	hw_trap	arm disarm
[1]A value of -1 means that the layer is not set and that the default layer setting is used when the device is set online.		

Table 44. qeth tasks and attributes in layer 3 mode

Task	Corresponding attributes	Possible attribute values
"Setting up a Linux router" on page 288	route4 route6	primary_router secondary_router primary_connector secondary_connector multicast_router no_router
"Turning inbound checksum calculations on and off" on page 291	none	n/a
"Turning outbound checksum calculations on and off" on page 291	none	n/a
"Enabling and disabling TCP segmentation offload" on page 292	none	n/a
"Faking broadcast capability" on page 293	fake_broadcast [1]	0 or 1
"Taking over IP addresses" on page 293	ipa_takeover/enable	0 or 1 or toggle
	ipa_takeover/add4 ipa_takeover/add6 ipa_takeover/del4 ipa_takeover/del6	IPv4 or IPv6 IP address and mask bits
	ipa_takeover/invert4 ipa_takeover/invert6	0 or 1 or toggle
"Configuring a device for proxy ARP" on page 297	rxip/add4 rxip/add6 rxip/del4 rxip/del6	IPv4 or IPv6 IP address
"Configuring a device for virtual IP address (VIPA)" on page 298	vipa/add4 vipa/add6 vipa/del4 vipa/del6	IPv4 or IPv6 IP address
"Configuring a HiperSockets device for AF_IUCV addressing" on page 299	hsuid	1 to 8 characters
"Setting up a HiperSockets network traffic analyzer" on page 315	sniffer	0 or 1
[1] not valid for HiperSockets		

Tips:
- Use the **qethconf** command instead of using the attributes for IPA, proxy ARP, and VIPA directly (see "qethconf - Configure qeth devices" on page 658).
- Your distribution might also provide a distribution-specific configuration tool. See your distribution documentation for distribution-specific alternatives.

sysfs provides multiple paths through which you can access the qeth group device attributes. For example, if a device with bus ID 0.0.a100 corresponds to interface eth0:

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

all lead to the attributes for the same device. For example, the following commands are all equivalent and return the same value:

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

However, the path through `/sys/class/net` is available only while the device is online. Furthermore, it might lead to a different device if the assignment of interface names changes. A change can occur after rebooting or when devices are ungrouped and new group devices created.

Tips:

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

The following sections describe the tasks in detail.

Creating a qeth group device

Use the **znetconf** command to configure network devices. Alternatively, you can use sysfs.

Before you begin

You must know the device bus-IDs that correspond to the read, write, and data subchannel of your OSA-Express CHPID in QDIO mode or HiperSockets CHPID as defined in the IOCDS of your mainframe.

Procedure

To create a qeth group device, either:

- Issue the **znetconf** command to create and configure a group device. The command groups the correct bus-IDs for you and sets the device online. For information about the **znetconf** command, see "znetconf - List and configure network devices" on page 699.
- Write the device numbers of the subchannel triplet to the sysfs `group` attribute to only define a group device. Issue a command of the form:

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

Results

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

`/sys/bus/ccwgroup/drivers/qeth/<read_device_bus_id>`

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

Note: If you defined an OSA-Express CHPID in QDIO mode for a mainframe earlier than z990 you might need to assign the portname attribute (see "Assigning a port name" on page 273).

Example

In this example (see Figure 54), a single OSA-Express CHPID in QDIO mode is used to connect a Linux instance to a network.

Mainframe configuration:

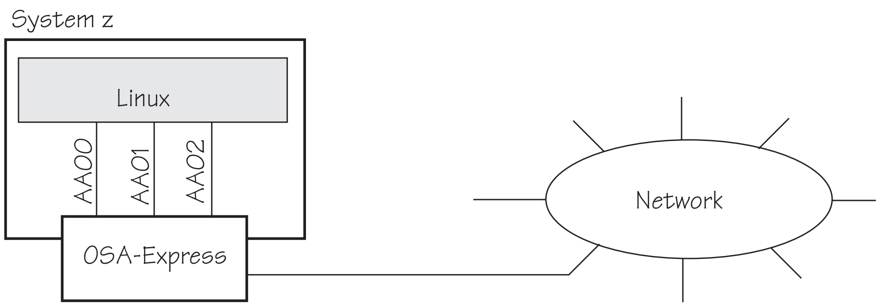


Figure 54. Mainframe configuration

Linux configuration:

Assuming that 0.0.aa00 is the device bus-ID that corresponds to the read subchannel:

```
# echo 0.0.aa00,0.0.aa01,0.0.aa02 > /sys/bus/ccwgroup/drivers/qeth/group
```

This command results in the creation of the following directories in sysfs:

- `/sys/bus/ccwgroup/drivers/qeth/0.0.aa00`
- `/sys/bus/ccwgroup/devices/0.0.aa00`
- `/sys/devices/qeth/0.0.aa00`

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

Removing a qeth group device

Use the `ungroup` sysfs attribute to remove a qeth group device.

Before you begin

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

Procedure

To remove a qeth group device, write 1 to the ungroup attribute. Issue a command of the form:

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

Example

This command removes device 0.0.aa00:

```
echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.aa00/ungroup
```

Setting the layer2 attribute

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

Before you begin

- To change a configured layer2 attribute, the network interface must be shut down and the device must be set offline.
- If you are using the layer2 option within a QDIO-based VSWITCH or guest LAN environment, avoid defining a VLAN with ID 1. Some switch vendors use ID 1 as the default value.

About this task

The qeth device driver attempts to load the layer 3 discipline for HiperSockets devices and layer 2 for non-HiperSockets devices.

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

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

HiperSockets
HiperSockets on layer 2 are supported as of System z10. There is no support for layer 2 to layer 3 conversion and, thus, no communication is possible between HiperSockets layer 2 interfaces and HiperSockets layer 3 interfaces. Do not include HiperSockets layer 2 interfaces and HiperSockets layer 3 interfaces in the same LAN.

z/VM VSWITCH or guest LAN
Linux must 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.

Procedure

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

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

where *<integer>* is
- 0 to turn off the `layer2` attribute; this results in the layer 3 discipline.
- 1 to turn on the `layer2` attribute; this results in the layer 2 discipline (default).

If the `layer2` attribute has a value of -1, the layer was not set. The default layer setting is used when the device is set online.

Results

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

For information about layer2, see:
- *OSA-Express Customer's Guide and Reference*, SA22-7935
- *OSA-Express Implementation Guide*, SG25-5848
- *Networking Overview for Linux on zSeries*, REDP-3901
- *z/VM Connectivity*, SC24-6174

Assigning a port name

You might need to set a port name for S/390 mainframes and z900 or z800 mainframes to identify the port for sharing by other operating system instances.

Before you begin

- You do not need to set a port name on the following mainframes:
  - HiperSockets and OSN CHPIDs
  - z9® and later mainframes
  - z890 and z990 mainframes
  - z900 and z800 mainframes with a microcode level of at least Driver 3G - EC stream J11204, MCL032 (OSA level 3.33)
- The device must be offline while you assign the port name.

About this task

For S/390 mainframes and z900 or z800 mainframes that are not exempted by the conditions that are listed under Before you begin, you must associate each OSA-Express CHPID in QDIO mode with a port name. The port name identifies the port for sharing by other operating system instances. The port name can be 1 - 8 characters long and must be uppercase. All operating system instances that share the port must use the same port name.

To assign a port name, set the `portname` device group attribute to the name of the port. Issue a command of the form:

```
# echo <PORTNAME> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/portname
```

Example

In this example (see Figure 55), two other mainframe operating systems share the OSA-Express CHPID in QDIO mode and use the port name “NETWORK1”.

Mainframe configuration:

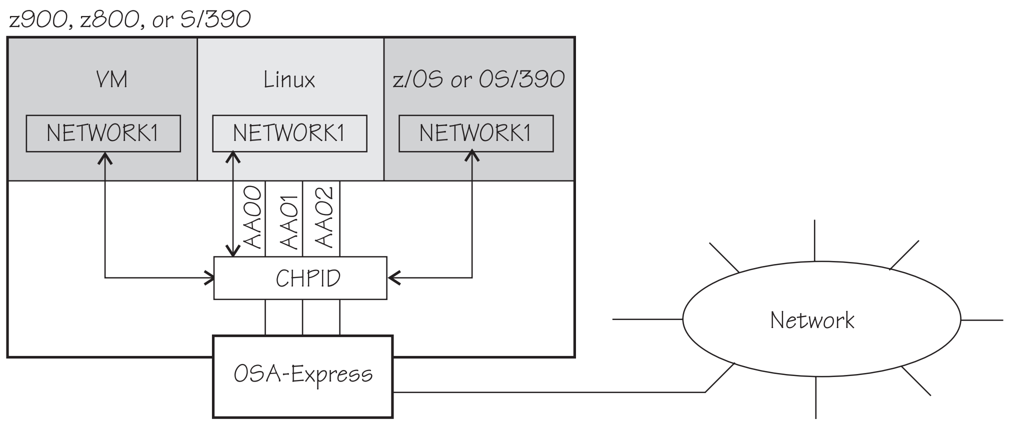


Figure 55. Mainframe configuration

Linux configuration:

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

Using priority queueing

An OSA-Express CHPID in QDIO mode has up to four output queues (queues 0 - 3). The priority queueing feature gives these queues different priorities (queue 0 having the highest priority). The four output queues are available only if multiple priority is enabled for queues on the OSA-Express CHPID in QDIO mode.

Before you begin

- Priority queueing applies to OSA-Express CHPIDs in QDIO mode only.
- If more than 160 TCP/IP stacks per OSA-Express CHPID are defined in the IOCDS, priority queueing is disabled.
- The device must be offline while you set the queueing options.

About this task

Queueing is relevant mainly in high traffic situations. When there is little traffic, queueing has no impact on processing. The qeth device driver can put data on one or more of the queues. By default, the driver uses queue 2 for all data.

You can determine how outgoing IP packages are assigned to queues by setting a value for the priority_queueing attribute of your qeth device. Issue a command of the form:

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

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

`prio_queueing_vlan`
: to base the queue assignment on the two most significant bits in the priority code point in the IEEE 802.1Q header as used in VLANs. This value affects only traffic with VLAN headers, and hence works only with qeth devices in layer 2 mode.

  You can set the priority code point in the IEEE 802.1Q headers of the traffic based on `skb->priority` by using a command of the form:

  ```
  ip link add link <link> name <name> type vlan id <vlan-id> egress-qos-map <mapping>
  ```

  Note: Enabling this option makes all traffic default to queue 3.

`prio_queueing_skb`
: to base the queue assignment on the priority flag of the skbs. An skb, or socket buffer, is a Linux kernel-internal structure that represents network data. The mapping to the priority queues is as follows:

Table 45. Mapping of flag value to priority queues

Priority flag of the skb	Priority queue
0-1	3
2-3	2
4-5	1
≥6	0

  You can use `prio_queueing_skb` for any network setups, including conventional LANs.

  Use either `sockopt SO_PRIORITY` or an appropriate **iptables** command to adjust the priority flag of the skb (`skb->priority`).

  Note: The priority flag of the skbs defaults to 0, hence enabling this option makes all traffic default to queue 3.

`prio_queueing_prec`
: to base the queue assignment on the two most significant bits of each packet's IP header precedence field. To set the precedence field, use `sockopt IP_TOS` (for IPv4) or `IPV6_TCLASS` (for IPv6).

  Note: Enabling this option makes all traffic default to queue 3.

`prio_queueing_tos`
: Deprecated; do not use for new setups.

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

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

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

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

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

Example

To read what is set for priority queueing for device 0.0.a110, issue:

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a110/priority_queuing
```

Possible results are:

by VLAN headers
if `prio_queueing_vlan` is set.

by skb-priority
if `prio_queueing_skb` is set.

by precedence
if `prio_queueing_prec` is set.

by type of service
if `prio_queuing_tos` is set.

always queue <*x*>
otherwise.

To configure queueing by `skb->priority` setting for device 0.0.a110 issue:

```
# echo prio_queueing_skb > /sys/bus/ccwgroup/drivers/qeth/0.0.a110/priority_queueing
```

Specifying the number of inbound buffers

Depending on the amount of available storage and the amount of traffic, you can assign 8 - 128 inbound buffers for each qeth group device.

Before you begin

The device must be offline while you specify the number of buffers for inbound traffic.

About this task

By default, the qeth device driver assigns 64 inbound buffers to OSA devices and 128 to HiperSockets devices.

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

The buffer size is equivalent to the frame size. See "Finding out the maximum frame size" on page 277 for details.

Procedure

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
```

Finding out the maximum frame size

The inbuf_size parameter returns the maximum frame size (MFS) in KB. To find out the MFS, read the inbuf_size attribute of the devices.

About this task

Issue a command of the form:

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

An OSA-Express CHPID in QDIO mode or an OSA-Express CHPID in OSN mode allows packing of data, and always run with an MFS of 64 KB.

HiperSockets CHPIDs do not pack data and run with a frame size that matches their definition in the hardware configuration (IOCP CHPARM specification). On HiperSockets, the MFS maps to corresponding maximum transmission unit (MTU) sizes, see Table 46.

Table 46. HiperSockets MFS and corresponding MTU sizes

inbuf_size value	MFS	MTU
16k	16 KB	8 KB
24k	24 KB	16 KB
40k	40 KB	32 KB
64k	64 KB	56 KB

Example

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

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a100/inbuf_size
64k
```

Specifying the relative port number

Use the portno sysfs attribute to specify the relative port number.

Before you begin

- This description applies to network adapters that, per CHPID, show more than one port to Linux.
- The device must be offline while you specify the relative port number.

Procedure

By default, the qeth group device uses port 0. To use a different port, issue a command of the form:

```
# 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

Use the card_type attribute to find out the type of the network adapter through which your device is connected.

Procedure

You can find out the type of the network adapter through which your device is connected. To find out the type, read the device's card_type attribute. 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 adapter and the type of network link (if applicable) available at the card's ports. See Table 47 for details.

Table 47. Possible values of card_type and what they mean

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

Table 47. Possible values of card_type and what they mean (continued)

Value of card_type	Adapter type	Link type
Unknown	Other	

Example

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

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

Setting a device online or offline

Use the online device group attribute to set a device online or offline.

Procedure

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

```
# 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”). When you set a device successfully online or offline, a change uevent is created.
Setting a device offline closes this network device. If IPv6 is active, you lose any IPv6 addresses set for this device. After you set the device online, you can restore lost IPv6 addresses only by issuing the **ip** or an equivalent command 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.

Procedure

To find the interface name of a qeth group device, either:

- Obtain a mapping for all qeth interfaces and devices by issuing the **lsqeth -p** command.
- Find out the interface name of a qeth group device for which you know the device bus-ID by reading the group device's if_name attribute. Issue a command of the form:

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

Example

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

Finding out the bus ID of a qeth interface

Use the **lsqeth -p** command to obtain a mapping for all qeth interfaces and devices. Alternatively, you can use sysfs.

Procedure

To find the device bus-ID that corresponds to an interface, either:

- Use the **lsqeth -p** command.
- Use the readlink command. For each network interface, there is a directory in sysfs under `/sys/class/net/`, for example, `/sys/class/net/eth0` for interface eth0. This directory contains a symbolic link “device” to the corresponding device in `/sys/devices`. Read this link to find the device bus-ID of the device that corresponds to the interface.

Example

To find out which device bus-ID corresponds to an interface eth0 issue, for example:

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

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

Activating an interface

Use the **ip** command or equivalent to activate an interface.

Before you begin

- You must know the interface name of the qeth group device (see “Finding out the interface name of a qeth group device” on page 279).
- You must know the IP address that you want to assign to the device.

About this task

The MTU size defaults to the correct settings for HiperSockets devices. For OSA-Express CHPIDs in QDIO mode, the default MTU size depends on the device mode, layer 2 or layer 3.

- For layer 2, the default MTU is 1500 bytes.
- For layer 3, the default MTU is 1492 bytes.

In most cases, these defaults are well suited for OSA-Express CHPIDs in QDIO mode. If your network is laid out for jumbo frames, increase the MTU size to a maximum of 9000 bytes for layer 2, or to 8992 bytes for layer 3. See *OSA-Express Customer's Guide and Reference*, SA22-7935 for more details about MTU size.

For HiperSockets, the maximum MTU size is restricted by the maximum frame size as announced by the Licensed Internal Code (LIC). The maximum MTU is equal to the frame size minus 8 KB. Hence, the possible frame sizes of 16 KB, 24 KB, 40 KB, or 64 KB result in maximum corresponding MTU sizes of 8 KB, 16 KB, 32 KB, or 56 KB.

On heavily loaded systems, MTU sizes that exceed 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 drop packets, possibly so frequently as to make the network interface unusable.

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 allow larger sizes.

Procedure

You activate or deactivate network devices with **ip** or an equivalent command. For details of the **ip** command, see the **ip** man page.

Examples

- This example activates a HiperSockets CHPID with broadcast address 192.168.100.255:

```
# ip addr add 192.168.100.10/24 dev hsi0
# ip link set dev hsi0 up
```

- This example activates an OSA-Express CHPID in QDIO mode with broadcast address 192.168.100.255:

```
# ip addr add 192.168.100.11/24 dev eth0
# ip link set dev eth0 up
```

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

```
# ip link set dev eth0 up
```

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

```
# ip link set dev osn0 up
```

Confirming that an IP address has been set under layer 3

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

About this task

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

There are usually failure messages about not being able to set the IP address or duplicate IP addresses in the kernel messages. You can find these messages in the output of the **dmesg** command. For most distributions, you can also find the messages in /var/log/messages.

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

Duplicate IP addresses

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

About this task

Several setups require duplicate addresses:

- To perform IP takeover you need to be able to set the IP address to be taken over. This address exists prior to the takeover. See "Taking over IP addresses" on page 293 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 297 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 298 for details.

You can use the **qethconf** command (see "qethconf - Configure qeth devices" on page 658) to maintain a list of IP addresses that your device can take over, a list of IP addresses for which your device can handle ARP, and a list of IP addresses that can be used as virtual IP addresses, regardless of any duplicates on the same OSA-Express adapter or in the LAN.

Deactivating an interface

You can deactivate an interface with **ip** or an equivalent command or by setting the network device offline.

About this task

Setting a device offline involves actions on the attached device, but deactivating a device only stops the interface logically within Linux.

Procedure

To deactivate an interface with **ip**, issue a command of the form:

```
# ip link set dev <interface_name> down
```

Example

To deactivate eth0 issue:

```
# ip link set dev eth0 down
```

Recovering a device

You can use the recover attribute of a qeth group device to recover it in case of failure.

About this task

For example, error messages in /var/log/messages from the qeth, qdio, or cio kernel modules might inform you of a malfunctioning device.

Setting the recover attribute schedules recovery synchronously, however the recovery itself might take some time.

Procedure

Issue a command of the form:

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

Example

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

Isolating data connections

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

About this task

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

QDIO data connection isolation is configured as a policy. The policy is implemented as a sysfs attribute called isolation. The attribute appears in sysfs regardless of whether the hardware supports the feature. The policy can take the following values:

none No isolation. This value is the default.

drop Specifies the ISOLATION_DROP policy. All packets from guests that share an OSA adapter to guests that have this policy configured are dropped automatically. The same holds for all packets that are sent by the guest with this policy configured to guests on the same OSA card. All packets to or from the isolated guest must have a target that is not hosted on the OSA card. You can accomplish this by a router hosted on a separate machine or a separate OSA adapter.

For example, assume that three Linux instances share an OSA adapter, but only one instance (Linux A) must be isolated. Then Linux A declares its OSA adapter (QDIO Data Connection to the OSA adapter) to be isolated. Any packet sent to or from Linux A must pass at least the physical switch to which the shared OSA adapter is connected. Linux A cannot communicate with other instances that share the OSA adapter, here B or C. The two other instances can still communicate directly through the OSA adapter without the external switch in the network path (see Figure 56 on page 284 page 284).

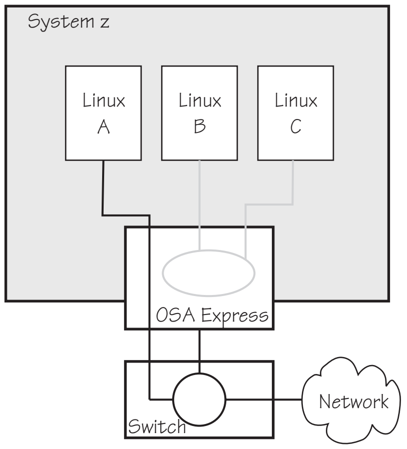


Figure 56. Linux instance A is isolated from instances B and C

forward

Specifies the ISOLATION_FORWARD policy. All packets are passed through a switch. The ISOLATION_FORWARD policy requires a network adapter in VEPA mode with an adjacent switch port configured for reflective relay mode.

Using a network adapter in VEPA mode achieves further isolation. VEPA mode forces traffic from the Linux guests to be handled by the external switch. For example, Figure 57 shows instances A and B with ISOLATION_FORWARD specified for the policy. All traffic between A and B goes through the external switch. The rule set of the switch now determines which connections are possible. The graphic assumes that A can communicate with B, but not with C.

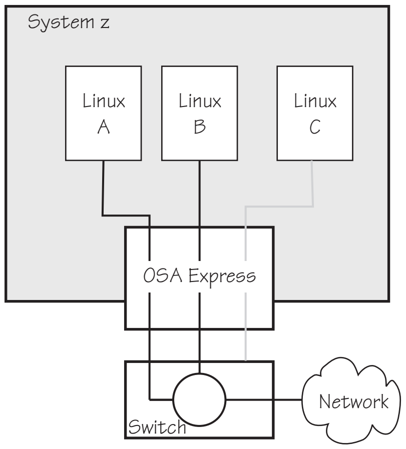


Figure 57. Traffic from Linux instance A and B is forced through an external switch

If the ISOLATION_FORWARD policy was enforced successfully, but the switch port later loses the reflective-relay capability, the device is set offline to prevent damage.

You can configure the policy regardless of whether the device is online. If the device is online, the policy is configured immediately. If the device is offline, the policy is configured when the device comes online.

Examples

- To check the current isolation policy:

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

- To set the isolation policy to ISOLATION_DROP:

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

- To set the isolation policy to ISOLATION_FORWARD:

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

  If the switch is not capable of VEPA support, or VEPA support is not configured on the switch, then you cannot set the isolation attribute value to 'forward' while the device is online. If the switch does not support VEPA and you set the isolation value 'forward' while the device is offline, then the device cannot be set online until the isolation value is set back to 'drop' or 'none'.
- To set the isolation policy to none:

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

When you use vNICs, VEPA mode must be enabled on the respective VSWITCH. See *z/VM Connectivity*, SC24-6174 for information about setting up data connection isolation on a VSWITCH.

Starting and stopping collection of QETH performance statistics

Use the performance_stats attribute to start and stop collection of QETH performance statistics.

About this task

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

This attribute is initially set to 0, that is, QETH performance data is not collected.

Procedure

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. See the **ethtool** man page for details.

Example

The following example shows statistic and device driver information:

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

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

Capturing a hardware trace

Hardware traces are intended for use by the IBM service organization. Hardware tracing is turned off by default. Turn on the hardware-tracing feature only when instructed to do so by IBM service.

Before you begin

- The OSA-Express adapter must support the hardware-tracing feature.
- The qeth device must be online to return valid values of the `hw_trap` attribute.

About this task

When errors occur on an OSA-Express adapter, both software and hardware traces must be collected. The hardware-tracing feature requests a hardware trace if an error is detected. This feature makes it possible to correlate the hardware trace with the device driver trace. If the hardware-tracing feature is activated, traces are captured automatically, but you can also start the capturing yourself.

Procedure

To activate or deactivate the hardware-tracing feature, issue a command of the form:

```
# echo <value> > /sys/devices/qeth/<device_bus_id>/hw_trap
```

Where *<value>* can be:

arm
If the hardware-tracing feature is supported, write `arm` to the `hw_trap` sysfs attribute to activate it. If the hardware-tracing feature is present and activated, the `hw_trap` sysfs attribute has the value `arm`.

disarm
Write `disarm` to the `hw_trap` sysfs attribute to turn off the hardware-tracing feature. If the hardware-tracing feature is not present or is turned off, the `hw_trap` sysfs attribute has the value `disarm`. This setting is the default.

trap
(Write only) Capture a hardware trace. Hardware traces are captured automatically, but if asked to do so by IBM service, you can start the capturing yourself by writing `trap` to the `hw_trap` sysfs attribute. The hardware trap function must be set to arm.

Examples

In this example the hardware-tracing feature is activated for qeth device 0.0.a000:

```
# echo arm > /sys/devices/qeth/0.0.a000/hw_trap
```

In this example a trace capture is started on qeth device 0.0.a000:

1. Check that the `hw_trap` sysfs attribute is set to arm:

```
# cat /sys/devices/qeth/0.0.a000/hw_trap
arm
```

2. Start the capture:

```
# echo trap > /sys/devices/qeth/0.0.a000/hw_trap
```

Working with qeth devices in layer 3 mode

Tasks you can perform on qeth devices in layer 3 mode include setting up a router, configuring offload operations, and taking over IP addresses.

Use the layer 2 attribute to set the mode. See “Setting the layer2 attribute” on page 272 about setting the mode. See “Layer 2 and layer 3” on page 258 for general information about the layer 2 and layer 3 disciplines.

Setting up a Linux 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.

Before you begin

- A suitable hardware setup must be in place that enables your Linux instance to act as a router.
- The Linux instance is set up as a router. To configure Linux running as a z/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, enable IP forwarding by issuing:

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

  For IPv6, enable IP forwarding by issuing:

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

Note:

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

About this task

You can set the route4 or route6 attribute dynamically, while the qeth device is online.

The same values are possible for route4 and route6 but depend on the type of CHPID, as shown in Table 48.

Table 48. Summary of router setup values

Router specification	OSA-Express CHPID in QDIO mode	HiperSockets CHPID
primary_router	Yes	No
secondary_router	Yes	No
primary_connector	No	Yes
secondary_connector	No	Yes
multicast_router	Yes	Yes
no_router	Yes	Yes

Both types of CHPIDs accept:

`multicast_router`
: causes the qeth driver to receive all multicast packets of the CHPID. For a unicast function for HiperSockets see "HiperSockets Network Concentrator" on page 309.

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

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

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

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

A HiperSockets CHPID accepts the following values, if the microcode level supports the feature:

`primary_connector`
to make your Linux instance the principal connection between a HiperSockets network and an external network (see "HiperSockets Network Concentrator" on page 309).

`secondary_connector`
to make your Linux instance a backup connection between a HiperSockets network and an external network (see "HiperSockets Network Concentrator" on page 309).

Example

In this example (see Figure 58), two Linux instances, "Linux P" and "Linux S", running on an IBM mainframe use OSA-Express to act as primary and secondary routers between two networks. IP forwarding must be enabled for Linux in an LPAR or as a z/VM guest to act as a router. IP forwarding is configured in procfs or in a configuration file; see your distribution manual for details.

Mainframe configuration:

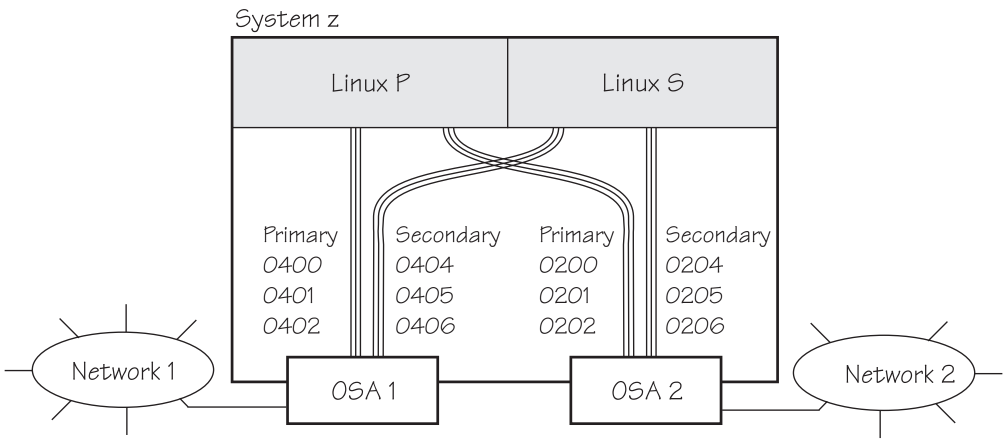


Figure 58. Mainframe configuration

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

Linux P configuration:

To create the qeth group devices:

```
# echo 0.0.0400,0.0.0401,0.0.0402 > /sys/bus/ccwgroup/drivers/qeth/group
# echo 0.0.0200,0.0.0201,0.0.0202 > /sys/bus/ccwgroup/drivers/qeth/group
```

To assign port names to the CHPIDs (For S/390 and certain microcode levels of z900 and z800 mainframes only, see “Assigning a port name” on page 273):

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

To make Linux P a primary router for IPv4:

```
# echo primary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0400/route4
# echo primary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0200/route4
```

Linux S configuration:

To create the qeth group devices:

```
# echo 0.0.0404,0.0.0405,0.0.0406 > /sys/bus/ccwgroup/drivers/qeth/group
# echo 0.0.0204,0.0.0205,0.0.0206 > /sys/bus/ccwgroup/drivers/qeth/group
```

To assign port names to the CHPIDs (For S/390 and certain microcode levels of z900 and z800 mainframes only, see “Assigning a port name” on page 273):

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

To make Linux S a secondary router for IPv4:

```
# echo secondary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0404/route4
# echo secondary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0204/route4
```

In this example, qeth device 0.0.1510 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 309 for further examples.

Configuring offload operations

Some operations can be offloaded to the OSA adapter, thus relieving the burden on the host CPU.

The qeth device driver supports offloading the following operations:

- Inbound (receive) checksum calculations
- Outbound (send) checksum calculations
- Large send (TCP segmentation offload)

Offload operations are supported for OSA connections on layer 3 only. VLAN interfaces inherit offload settings from their base interface.

The offload operations can be set with the Linux **ethtool** command, version 6 or later. See the **ethtool** man page for details. The following example shows the default offload settings:

```
# ethtool -k eth0
Offload parameters for eth0:
rx-checksumming: on
tx-checksumming: off
scatter-gather: off
tcp-segmentation-offload: off
udp-fragmentation-offload: off
generic-segmentation-offload: off
generic-receive-offload: on
large-receive-offload: off
```

Note: With kernel 2.6.39, the defaults for `rx-checksumming` and for `generic-receive-offload` changed from `off` to `on`.

Turning inbound checksum calculations on and off

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

Procedure

The qeth device driver supports offloading checksum calculations on inbound packets to the OSA feature. To enable or disable checksum calculations by the OSA feature, issue a command of this form:

```
# ethtool -K <interface_name> rx <value>
```

where *<value>* is on or off.

Examples

- To let the OSA feature calculate the inbound checksum for network device eth0, issue

  ```
  # ethtool -K eth0 rx on
  ```

- To let the host CPU calculate the inbound checksum for network device eth0, issue

  ```
  # ethtool -K eth0 rx off
  ```

Turning outbound checksum calculations on and off

The qeth device driver supports offloading outbound (send) checksum calculations to the OSA feature.

About this task

You can enable or disable the OSA feature calculating the outbound checksums by using the **`ethtool`** command.

Attention: When outbound checksum calculations are offloaded, the OSA feature performs the checksum calculations. Offloaded checksum calculations only applies to packets that go out to the LAN or come in from the LAN. Linux instances that share an OSA port exchange packages directly. The packages are forwarded by the OSA adapter but do not go out on the LAN and no checksum offload is performed. The qeth device driver cannot detect this, and so cannot issue any warning about it.

Procedure

Issue a command of the form:

```
# ethtool -K <interface_name> tx <value>
```

where *<value>* is on or off.

Example

- To let the OSA feature calculate the outbound checksum for network device eth0, issue

  ```
  # ethtool -K eth0 tx on
  ```

- To let the host CPU calculate the outbound checksum for network device eth0, issue

  ```
  # ethtool -K eth0 tx off
  ```

Enabling and disabling TCP segmentation offload

Offloading the TCP segmentation operation from the Linux network stack to the adapter can lead to enhanced performance for interfaces with predominately large outgoing packets.

Procedure

To support TCP segmentation offload (TSO), a network device must support outbound (TX) checksumming and scatter gather. For this reason, you must turn on scatter gather and outbound checksumming prior to configuring TSO. All three options can be turned on or off with a single **ethtool** command of the form:

```
# ethtool -K <interface_name> tx <value> sg <value> tso <value>
```

where *<value>* is either on or off.

Attention: When TCP segmentation is offloaded, the OSA feature performs the calculations. Offloaded calculations apply only to packets that go out to the LAN or come in from the LAN. Linux instances that share an OSA port exchange packages directly. The packages are forwarded by the OSA adapter but do not go out on the LAN and no TCP segmentation calculation is performed. The qeth device driver cannot detect this, and so cannot issue any warning about it.

Examples

- To enable TSO for a network device eth0 issue:

  ```
  # ethtool -K eth0 tx on sg on tso on
  ```

- To disable TSO for a network device eth0 issue:

  ```
  # ethtool -K eth0 tx off sg off tso off
  ```

Faking broadcast capability

It is possible to fake the broadcast capability for devices that do not support broadcasting.

Before you begin

- You can fake the broadcast capability only on devices that do not support broadcast.
- The device must be offline while you enable faking broadcasts.

About this task

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

To find out whether a device supports broadcasting, use the **ip** command. If the resulting list shows the BROADCAST flag, the device supports broadcast. This example shows that the device eth0 supports broadcast:

```
# ip -s link show dev eth0
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1492 qdisc pfifo_fast qlen 1000
    link/ether 00:11:25:bd:da:66 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    236350     2974     0       0       0       9
    TX: bytes  packets  errors  dropped carrier collsns
    374443     1791     0       0       0       0
```

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

Procedure

To set the broadcast capable flag for devices that do not support broadcast, set the `fake_broadcast` attribute of the qeth group device to 1. To reset the flag, set it to `0`. Issue a command of the form:

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

Example

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

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

Taking over IP addresses

You can configure IP takeover if the layer2 option is not enabled. If you enabled the layer2 option, you can configure for IP takeover as you would in a distributed server environment.

About this task

For information about the layer2 option, see “MAC headers in layer 2 mode” on page 261.

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

An OSA-Express CHPID in QDIO mode can take over IP addresses from any zSeries operating system. IP takeover for HiperSockets CHPIDs is restricted to taking over addresses from other Linux instances in the same Central Electronics Complex (CEC).

IP address takeover between multiple CHPIDs requires ARP for IPv4 and Neighbor Discovery for IPv6. OSA-Express handles ARP transparently, but not Neighbor Discovery.

There are three stages to taking over an IP address:

Stage 1: Ensure that your qeth group device is enabled for IP takeover

Stage 2: Activate the address to be taken over for IP takeover

Stage 3: Issue a command to take over the address

Stage 1: Enabling a qeth group device for IP takeover

For OSA-Express and HiperSockets CHPIDs, both the qeth group device that is to take over an IP address and the device that surrenders the address must be enabled for IP takeover.

Procedure

By default, qeth devices are not enabled for IP takeover. To enable a qeth group device for IP address takeover set the enable device group attribute to `1`. To switch off the takeover capability set the enable device group attribute to `0`. In sysfs, the enable attribute is located in a subdirectory `ipa_takeover`. Issue a command of the form:

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

Example

In this example, a device 0.0.a500 is enabled for IP takeover:

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.a500/ipa_takeover/enable
```

Stage 2: Activating and deactivating IP addresses for takeover

The qeth device driver maintains a list of IP addresses that qeth group devices can take over or surrender. To enable Linux to take over an IP-address or to surrender an address, the address must be added to this list.

Procedure

Use the **`qethconf`** command to add IP addresses to the list.

- To display the list of IP addresses that are activated for IP takeover issue:

```
# qethconf ipa list
```

- To activate an IP address for IP takeover, add it to the list. Issue a command of the form:

```
# qethconf ipa add <ip_address>/<mask_bits> <interface_name>
```

- To deactivate an IP address delete it from the list. Issue a command of the form:

```
# qethconf ipa del <ip_address>/<mask_bits> <interface_name>
```

In these commands, *<ip_address>/<mask_bits>* is the range of IP addresses to be activated or deactivated. See "qethconf - Configure qeth devices" on page 658 for more details about the **qethconf** command.

IPv4 example:

In this example, there is only one range of IP addresses (192.168.10.0 to 192.168.10.255) that can be taken over by device hsi0.

List the range of IP addresses (192.168.10.0 to 192.168.10.255) that can be taken over by device hsi0.

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

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

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

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

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

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

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

IPv6 example:

The following command adds one range of IPv6 addresses, fec0:0000:0000:0000:0000:0000:0000:0000 to fec0:0000:0000:0000:FFFF:FFFF:FFFF:FFFF, that can be taken over by device eth2.

Add a range of IP addresses:

```
qethconf ipa add fec0::/64 eth2
qethconf: Added fec0:0000:0000:0000:0000:0000:0000:0000/64 to
          sysfs entry /sys/class/net/eth2/device/ipa_takeover/add6.
qethconf: For verification please use "qethconf ipa list"
```

Listing the activated IP addresses now shows the range of addresses:

```
qethconf ipa list
...
ipa add fec0:0000:0000:0000:0000:0000:0000:0000/64 eth2
```

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

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

Stage 3: Issuing a command to take over the address

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

Before you begin

- Both the device that is to take over the IP address and the device that is to surrender the IP address must be enabled for IP takeover. This rule applies to the devices on both OSA-Express and HiperSockets CHPIDs. (See “Stage 1: Enabling a qeth group device for IP takeover” on page 294).
- 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 294).

About this task

Be aware of the information in “Confirming that an IP address has been set under layer 3” on page 281 when using IP takeover.

Examples

IPv4 example:

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

```
# ip addr add 192.168.10.22/24 dev hsi0
```

For IPv4, the IP address you are taking over must be different from the one that is already set for your device. If your device already has the IP address it is to take over, you must issue two commands: First remove the address to be taken over if it is already there. Then add the IP address to be taken over.

For example, to make a device hsi0 take over IP address 192.168.10.22 if hsi0 is already configured to have IP address 192.168.10.22 issue:

```
# ip addr del 192.168.10.22/24 dev hsi0
# ip addr add 192.168.10.22/24 dev hsi0
```

IPv6 example:

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

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

For IPv6, setting the **nodad** (no duplicate address detection) option ensures that the eth2 interface uses the IP address fec0::111:25ff:febd:d9da/64. Without the **nodad** option, the previous owner of the IP address might prevent the takeover by responding to a duplicate address detection test.

The IP address you are taking over must be different from the one that is already set for your device. If your device already has the IP address it is to take over you must issue two commands: First remove the address to be taken over if it is already there. Then add the IP address to be taken over.

For example, to make a device eth2 take over IP address fec0::111:25ff:febd:d9da/64 when eth2 is already configured to have that particular IP address issue:

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

Configuring a device for proxy ARP

You can configure a device for proxy ARP if the layer2 option is not enabled. If you enabled the layer2 option, you can configure for proxy ARP as you would in a distributed server environment.

Before you begin

Configure only qeth group devices that are set up as routers for proxy ARP.

About this task

For information about the layer2 option, see “MAC headers in layer 2 mode” on page 261.

The qeth device driver maintains a list of IP addresses for which a qeth group device handles ARP and issues gratuitous ARP packets. For more information about proxy ARP, see

`www.sjdjweis.com/linux/proxyarp`

Use the **`qethconf`** command to display this list or to change the list by adding and removing IP addresses (see “qethconf - Configure qeth devices” on page 658).

Be aware of the information in “Confirming that an IP address has been set under layer 3” on page 281 when you work with proxy ARP.

Example

Figure 59 on page 298 shows an environment where proxy ARP is used.

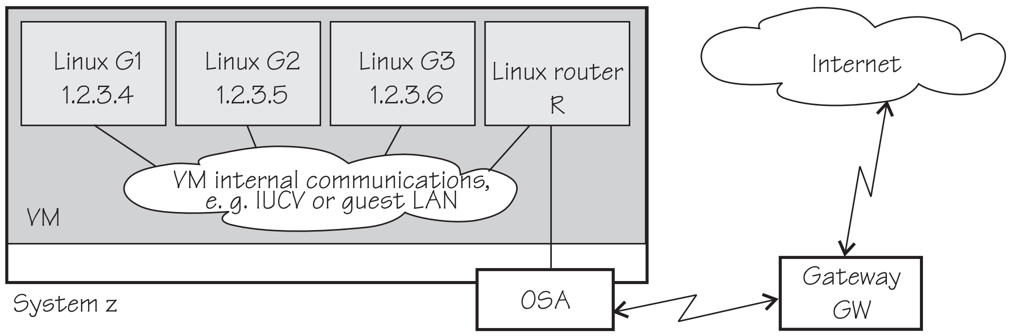


Figure 59. Example of proxy ARP usage

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

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

```
# qethconf parp add 1.2.3.4 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)

You can configure a device for VIPA if the layer2 option is not enabled. If you enabled the layer2 option, you can configure for VIPA as you would in a distributed server environment.

Before you begin

Virtual IP address (VIPA) can be configured only if the kernel was compiled with the common code configuration option CONFIG_DUMMY.

About this task

For information about the layer2 option, see “MAC headers in layer 2 mode” on page 261.

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 658).

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 281 when you work with VIPAs.

Configuring a HiperSockets device for AF_IUCV addressing

| Use the `hsuid` attribute of a HiperSockets device in layer 3 mode to identify it to the AF_IUCV addressing family support.

Before you begin

- Support for AF_IUCV based connections through real HiperSockets requires Completion Queue Support.
- The device must be set up for AF_IUCV addressing (see “Setting up HiperSockets devices for AF_IUCV addressing” on page 360).

Procedure

To set an identifier, issue a command of this form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/qeth/0.0.a007/hsuid
```

The identifier is case-sensitive and must adhere to these rules:

- It must be 1 - 8 characters.
- It must be unique across your environment.
- It must not match any z/VM user ID in your environment. The AF_IUCV addressing family support also supports z/VM IUCV connections.

Example

In this example, MYHOST01 is set as the identifier for a HiperSockets device with bus ID `0.0.a007`.

```
# echo MYHOST01 > /sys/bus/ccwgroup/drivers/qeth/0.0.a007/hsuid
```

Scenario: VIPA – minimize outage due to adapter failure

Using VIPA you can assign IP addresses that are not associated with a particular adapter. VIPA thus minimizes outage that is caused by adapter failure.

For VIPA you can use:

Standard VIPA
Standard VIPA is sufficient for applications, such as web servers, that do *not* open connections to other nodes.

Source VIPA (version 2.0.0 and later)
Source VIPA is used for applications that open connections to other nodes. Use Source VIPA Extensions to work with multiple VIPAs per destination in order to achieve multipath load balancing.

Note:

1. The VIPA functionality requires a kernel that is built with the CONFIG_DUMMY option.
2. See the information in “Confirming that an IP address has been set under layer 3” on page 281 concerning possible failure when you set IP addresses for OSA-Express features in QDIO mode (qeth device driver).
3. The configuration file layout for Source VIPA changed as of version 2.0.0. A policy is now included. For details, see the readme file and the man pages that are provided with the package.

Standard VIPA

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.

Setting up standard VIPA

To set up VIPA you must create a dummy device, ensure that your service listens to the IP address, and set up routing to it.

Procedure

Follow these main steps to set up VIPA in Linux:

1. Create a dummy device with a virtual IP address.
2. Ensure that your service (for example, the Apache web server) listens to the virtual IP address assigned in step 1.
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 60 on page 301).
   - Dynamic routing. For details of how to configure routes, you must see the documentation that is delivered with your routing daemon (for example, `zebra` or `gated`).

Adapter outage

If outage of an adapter occurs, you must switch adapters.

Procedure

- Under static routing:
  1. Delete the route that was set previously.
  2. Create an alternative route to the virtual IP address.
- Under dynamic routing, see the documentation that is delivered with your routing daemon for details.

Example of how to set up standard VIPA

This example shows you how to configure VIPA under static routing, and how to switch adapters when an adapter outage occurs.

About this task

Figure 60 on page 301 shows the network adapter configuration that is used in the example.

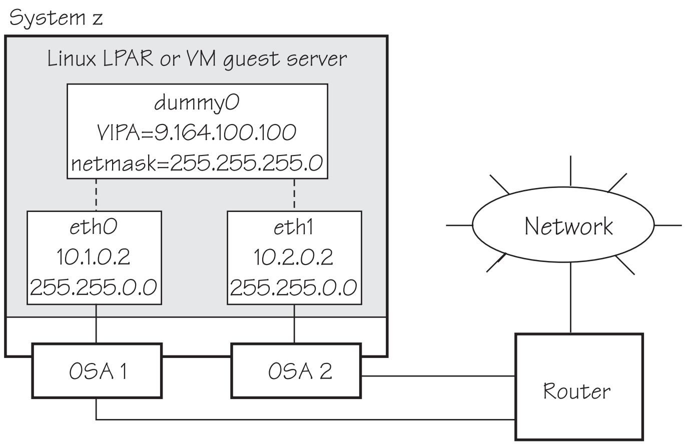


Figure 60. Example of using Virtual IP Address (VIPA)

Procedure

1. Define the real interfaces.

```
[server]# ip addr add 10.1.0.2/16 dev eth0
[server]# ip link set dev eth0 up
[server]# ip addr add 10.2.0.2/16 dev eth1
[server]# ip link set dev eth1 up
```

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

```
[server]# modprobe dummy
```

3. Create a dummy interface with a virtual IP address 9.164.100.100 and a netmask 255.255.255.0:

```
[server]# ip addr add 9.164.100.100/24 dev dummy0
[server]# ip link set dev dummy0 up
```

4. Enable the network devices for this VIPA so that it accepts packets for this IP address.
   - IPv4 example:

```
 [server]# qethconf vipa add 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 2002::1234:5678 eth0
qethconf: Added 2002:0000:0000:0000:0000:0000:1235:5678 to /sys/class/net/eth0/device/vipa/add6.
qethconf: Use "qethconf vipa list" to check for the result
[server]# qethconf vipa add 2002::1235:5678 eth1
qethconf: Added 2002:0000:0000:0000:0000:0000:1235:5678 to /sys/class/net/eth1/device/vipa/add6.
qethconf: Use "qethconf vipa list" to check for the result
```

5. Ensure that the addresses are 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 through the gateway with address `10.1.0.2`.

```
[router]# ip route add 9.164.100.100 via 10.1.0.2
```

What to do next

Now assume that an adapter outage occurs. You must then:

1. Delete the previously created route.

```
[router]# ip route del 9.164.100.100
```

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

```
[router]# ip route add 9.164.100.100 via 10.2.0.2
```

Source VIPA

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.

Some operating system kernels cannot do load balancing among several connections with the same source and destination address over several interfaces. The solution is to use several source addresses.

To achieve load balancing, a policy must be selected in the policy section of the configuration file of Source VIPA (`/etc/src_vipa.conf`). In this policy section, you can also specify several source addresses that are used for one destination. Source VIPA then applies the source address selection according to the rules of the policy that is selected in the configuration file.

This Source VIPA solution does not affect kernel stability. Source VIPA is controlled by a configuration file that contains flexible rules for when to use Source VIPA based on destination IP address ranges.

You can use IPv6 or IPv4 addresses for Source VIPA.

Setting up source VIPA

To set up source VIPA, define your address ranges in the configuration file.

Usage

You can obtain the src_vipa package at www.ibm.com/developerworks/linux/linux390/useful_add-ons_vipa.html. To install the package, issue:

```
make
make starter
make install
```

Paths can be changed in the makefile. The defaults are as follows:

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

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

Note:

If you upgrade from an earlier version of Source VIPA and do not need multiple VIPAs, use the `onevipa` policy that your VIPA follows (see "Policies" on page 304). Check your syslog (usually in `/var/log/messages`) for problems the first time you use the new version.

Configuration

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

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

```
# comment
D1.D2.D3.D4/MASK POLICY S1.S2.S3.S4 [T1.T2.T3.T4 [...]]
.INADDR_ANY P1-P2 POLICY S1.S2.S3.S4 [T1.T2.T3.T4 [...]]
.INADDR_ANY P POLICY S1.S2.S3.S4 [T1.T2.T3.T4 [...]]
```

`D1.D2.D3.D4/MASK` specifies a range of destination addresses and the number of bits set in the subnet mask (MASK). As soon as a socket is opened and connected to these destination addresses and the application does not do an explicit bind to a source address, Source VIPA does a bind to one of the source addresses specified (S, T, [...]). It uses the policy that is selected in the configuration file to distribute the source addresses. See "Policies" on page 304 for available load distribution policies. Instead of IP addresses in dotted notation, host names can also be used and are resolved using DNS.

You can use IPv6 or IPv4 IP addresses, but not both within a single rule in the configuration file. The following is an example of an IPv6 configuration file with a random policy:

```
# IPv6
2221:11c3:0123:d9d8:05d5:5a44:724c:783b/64 random ed27:120:da42:: 1112::33cc
```

`.INADDR_ANY P1-P2 POLICY S1.S2.S3.S4` or `.INADDR_ANY P POLICY S1.S2.S3.S4` causes bind calls with `.INADDR_ANY` as a local address to be intercepted if the port the socket is bound to is between P1 and P2 (inclusive). In this case, `.INADDR_ANY` is replaced by one of the source addresses specified (S, T, [...]), which can be 0.0.0.0.

All `.INADDR_ANY` statements are read and evaluated in order of appearance. This method 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" on page 304 for available load distribution policies.

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

Policies

With Source VIPA Extensions, you provide a range of dummy source addresses for replacing the source addresses of a socket. The policy that is 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 that is used is selected randomly from all the specified source addresses.

lrr (local round robin)
: The source address that is 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 that share a configuration file. All processes that use 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. Make this permission mask as restrictive as possible. A process that has 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> <parameters>
```

enables the Source VIPA function for the application. The configuration file is read when the application is started. It is also possible to change the starter script and run multiple applications with different Source VIPA settings in separate files. To do this, define and export a SRC_VIPA_CONFIG_FILE environment variable that points to the separate file before you start an application.

Note:

1. LD_PRELOAD security prevents `setuid` programs to be run under Source VIPA; programs of this kind can be run only when the real UID is 0. The ping utility is usually installed with `setuid` permissions.
2. The maximum number of VIPAs per destination is 8.

Example of how to set up source VIPA

This is an example of how to set up source VIPA.

About this task

Figure 61 shows a configuration where two applications with VIPA `9.164.100.100` and `9.164.100.200` are to be set up for source VIPA with a local round robin policy.

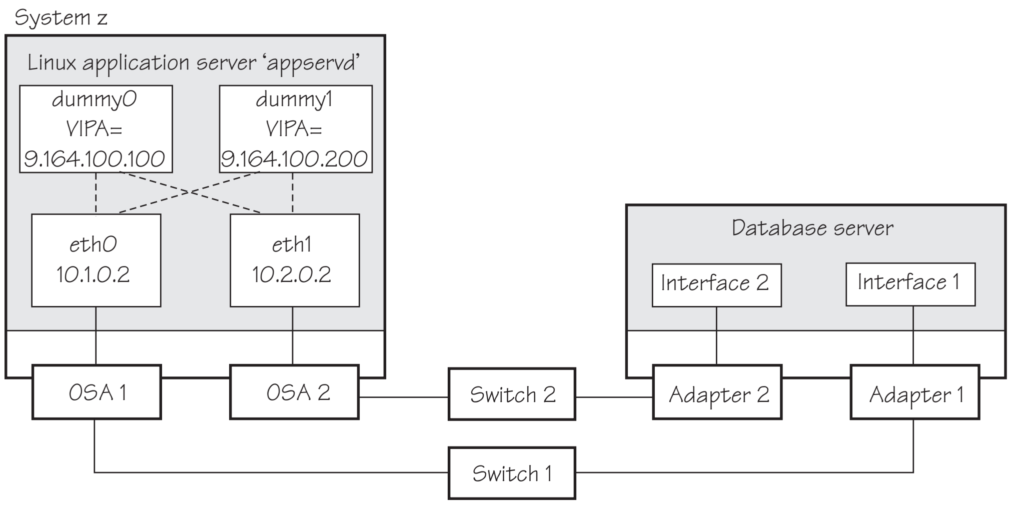


Figure 61. Example of using source VIPA

The required entry in the Source VIPA configuration file is:

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

Scenario: Virtual LAN (VLAN) support

VLAN technology works according to IEEE Standard 802.1Q by logically segmenting the network into different broadcast domains. Thus packets are switched only between ports that are designated for the same VLAN.

By containing traffic that originates on a particular LAN to other LANs within the same VLAN, switched virtual networks avoid wasting bandwidth. Wasted bandwidth is a drawback inherent in traditional bridged/switched networks where packets are often forwarded to LANs that do not require them.

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

The qeth device driver for OSA-Express (QDIO) and HiperSockets supports priority tags as specified by IEEE Standard 802.1Q for both layer2 and layer3.

Introduction to VLANs

Use VLANs to increase traffic flow and reduce latency. With VLANs, you can organize your network by traffic patterns rather than by physical location.

In a conventional network topology, such as that shown in Figure 62 on page 306, devices communicate across LAN segments in different broadcast domains by using routers. Although routers add latency by delaying transmission of data while they are using more of the data packet to determine destinations, they are preferable to building a single broadcast domain. A single domain can easily be flooded with traffic.

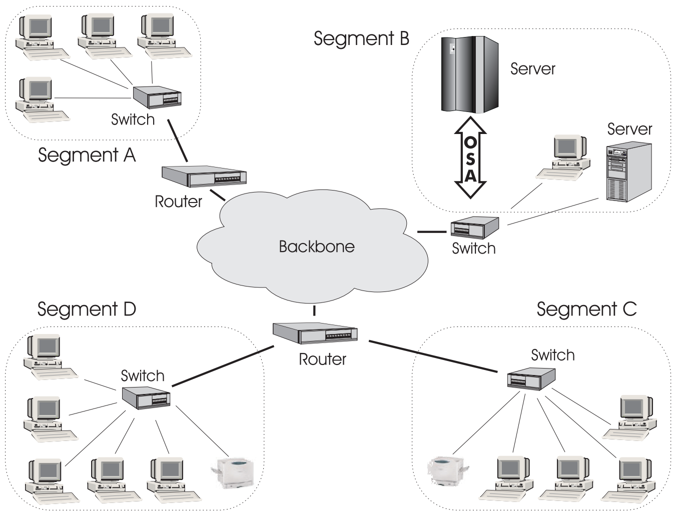


Figure 62. Conventional routed network

By organizing the network into VLANs by using Ethernet switches, distinct broadcast domains can be maintained without the latency that is introduced by multiple routers. As Figure 63 shows, a single router can provide the interfaces for all VLANs that appeared as separate LAN segments in the previous figure.

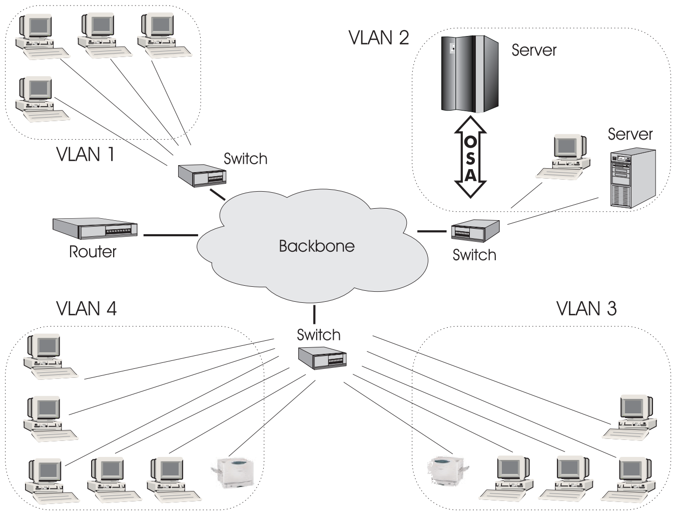


Figure 63. Switched VLAN network

Figure 64 on page 307 shows how VLANs can be organized logically, according to traffic flow, rather than being restricted by physical location. If workstations 1-3 communicate mainly with the small server, VLANs can be used to organize only these devices in a single broadcast domain that keeps broadcast traffic within the group. This setup reduces traffic both inside the domain and outside, on the rest of the network.

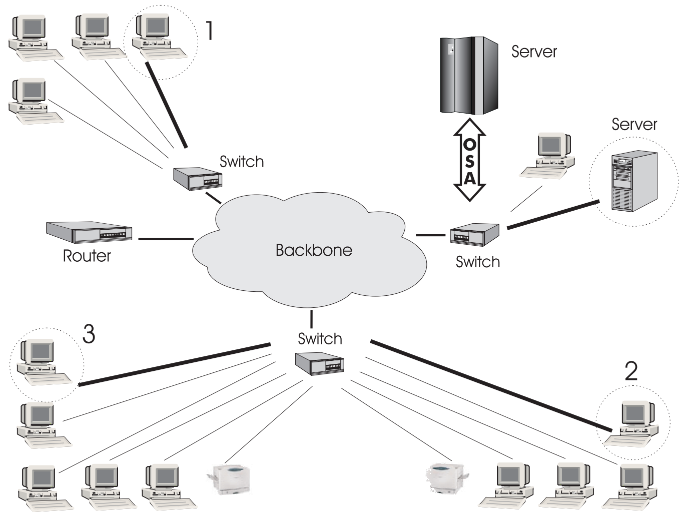


Figure 64. VLAN network organized for traffic flow

Configuring VLAN devices

Configure VLANs with the **ip link add** command. See the **ip-link** man page for details.

About this task

Information about the current VLAN configuration is available by listing the files in

```
/proc/net/vlan/*
```

with **cat** or more. For example:

```
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#
```

Example: Creating two VLANs

VLANs are allocated in an existing interface that represents a physical Ethernet LAN.

The following example creates two VLANs, one with ID 3 and one with ID 5.

```
ip addr add 9.164.160.23/19 dev eth1
ip link set dev eth1 up
ip link add dev eth1.3 link eth1 type vlan id 3
ip link add dev eth1.3 link eth1 type vlan id 5
```

The **ip link add** commands added interfaces “eth1.3” and “eth1.5”, which you can then configure:

```
ip addr add  1.2.3.4/24    dev eth1.3
ip link set dev eth1.3 up
ip addr add  10.100.2.3/16 dev eth1.5
ip link set dev eth1.5 up
```

The traffic that flows out of eth1.3 is in the VLAN with ID=3. This traffic is not received by other stacks that listen to VLANs with ID=4.

The internal routing table ensures that every packet to 1.2.3.x goes out through eth1.3, and everything to 10.100.x.x through eth1.5. Traffic to 9.164.1xx.x flows through eth1 (without a VLAN tag).

To remove one of the VLAN interfaces:

```
ip link set dev eth1.3 down
ip link delete eth1.3 type vlan
```

Example: Creating a VLAN with five Linux instances

An example of how to set up a VLAN with five Linux instances.

The following example illustrates the definition and connectivity test for a VLAN comprising five different Linux systems (two LPARs, two z/VM guest virtual machines, and one x86 system), each connected to a physical Ethernet LAN through eth1:

- LINUX1: LPAR

```
ip link add dev eth1.3 link eth1 type vlan id 5
ip addr add 10.100.100.1/24 dev eth1.5
ip link set dev eth1.5 up
```

- LINUX2: LPAR

```
ip link add dev eth1.3 link eth1 type vlan id 5
ip addr add 10.100.100.2/24 dev eth1.5
ip link set dev eth1.5 up
```

- LINUX3: z/VM guest

```
ip link add dev eth1.3 link eth1 type vlan id 5
ip addr add 10.100.100.3/24 dev eth1.5
ip link set dev eth1.5 up
```

- LINUX4: z/VM guest

```
ip link add dev eth1.3 link eth1 type vlan id 5
ip addr add 10.100.100.4/24 dev eth1.5
ip link set dev eth1.5 up
```

- LINUX5: x86

```
ip link add dev eth1.3 link eth1 type vlan id 5
ip addr add 10.100.100.5/24 dev eth1.5
ip link set dev eth1.5 up
```

Test the connections:

```
ping 10.100.100.1              // Unicast-PING
...
ping 10.100.100.5
ping -I eth1.5 224.0.0.1       // Multicast-PING
ping -b 10.100.100.255         // Broadcast-PING
```

HiperSockets Network Concentrator

You can configure a HiperSockets Network Concentrator on a QETH device in layer 3 mode.

Before you begin: The instructions that are given apply to IPv4 only. The HiperSockets Network Concentrator connector settings are available in layer 3 mode only.

The HiperSockets Network Concentrator connects systems to an external LAN within one IP subnet that uses HiperSockets. HiperSockets Network Concentrator connected systems look as if they were directly connected to the LAN. This simplification helps to reduce the complexity of network topologies that result from server consolidation.

Without changing the network setup, you can use HiperSockets Network Concentrator to port systems:

- From the LAN into a System z Server environment
- From systems that are connected by a different HiperSockets Network Concentrator into a System z Server environment

Thus, HiperSockets Network Concentrator helps to simplify network configuration and administration.

Design

A connector Linux system forwards traffic between the external OSA interface and one or more internal HiperSockets interfaces. The forwarding is done via IPv4 forwarding for unicast traffic and via a particular bridging code (xcec_bridge) for multicast traffic.

A script named ip_watcher.pl observes all IP addresses registered in the HiperSockets network and sets them as Proxy ARP entries (see “Configuring a device for proxy ARP” on page 297) on the OSA interfaces. The script also establishes routes for all internal systems to enable IP forwarding between the interfaces.

All unicast packets that cannot be delivered in the HiperSockets network are handed over to the connector by HiperSockets. The connector also receives all multicast packets to bridge them.

Setup

The setup principles for configuring the HiperSockets Network Concentrator on a mainframe Linux system are as follows:

leaf nodes
: The leaf nodes do not require a special setup. To attach them to the HiperSockets network, their setup should be as if they were directly attached to the LAN. They do not have to be Linux systems.

connector systems
: In the following, HiperSockets Network Concentrator IP refers to the subnet of the LAN that is extended into the HiperSockets net.

- If you want to support forwarding of all packet types, define the OSA interface for traffic into the LAN as a multicast router (see "Setting up a Linux router" on page 288).

  If only unicast packages are to be forwarded, there is also the possibility not to identify the OSA interface as multicast router: add the interface name to the `start_hsnc` script and only unicast packets are forwarded.
- All HiperSockets interfaces that are involved must be set up as connectors: set the route4 attributes of the corresponding devices to "primary_connector" or to "secondary_connector". Alternatively, you can add the OSA interface name to the start script as a parameter. This option results in HiperSockets Network Concentrator ignoring multicast packets, which are then not forwarded to the HiperSockets interfaces.
- IP forwarding must be enabled for the connector partition. Enable the forwarding either manually with the command

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

  Alternatively, you can use distribution-dependent configuration files to activate IP forwarding for the connector partition automatically after booting.
- The network routes for the HiperSockets interface must be removed. A network route for the HiperSockets Network Concentrator IP subnet must be established through the OSA interface. To establish a route, assign the IP address 0.0.0.0 to the HiperSockets interface. At the same time, assign an address that is used in the HiperSockets Network Concentrator IP subnet to the OSA interface. These assignments set up the network routes correctly for HiperSockets Network Concentrator.
- To start HiperSockets Network Concentrator, run the script `start_hsnc.sh`. You can specify an interface name as optional parameter. The interface name makes HiperSockets Network Concentrator use the specified interface to access the LAN. There is no multicast forwarding in that case.
- To stop HiperSockets Network Concentrator, use the command `killall ip_watcher.pl` to remove changes that are caused by running HiperSockets Network Concentrator.

Availability setups

If a connector system fails during operation, it can simply be restarted. If all the startup commands are run automatically, it will instantaneously be operational again after booting. Two common availability setups are mentioned here:

One connector partition and one monitoring system
: As soon as the monitoring system cannot reach the connector for a specific

timeout (for example, 5 seconds), it restarts the connector. The connector itself monitors the monitoring system. If it detects (with a longer timeout than the monitoring system, for example, 15 seconds) a monitor system failure, it restarts the monitoring system.

Two connector systems monitoring each other
In this setup, there is an active and a passive system. As soon as the passive system detects a failure of the active connector, it takes over operation. To take over operation, it must reset the other system to release all OSA resources for the multicast_router operation. The failed system can then be restarted manually or automatically, depending on the configuration. The passive backup HiperSockets interface can either switch into primary_connector mode during the failover, or it can be set up as secondary_connector. A secondary_connector takes over the connecting function, as soon as there is no active primary_connector. This setup has a faster failover time than the first one.

For further information about availability, consult the general documentation of Linux on System z on availability.

Hints

- The MTU of the OSA and HiperSockets link should be of the same size. Otherwise, multicast packets that do not fit in the link's MTU are discarded as there is no IP fragmentation for multicast bridging. Warnings are printed to `/var/log/messages` or a corresponding syslog destination.
- The script `ip_watcher.pl` prints error messages to the standard error descriptor of the process.
- `xcec-bridge` logs messages and errors to syslog. On most distributions, you can find these messages 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 function, issue `killall ip_watcher.pl`. This script removes all routing table and Proxy ARP entries added during the use of HiperSockets Network Concentrator.

Note:

1. Broadcast bridging is active only on OSA or HiperSockets hardware that can handle broadcast traffic without causing a bridge loop. If you see the message "`Setting up broadcast echo filtering for ... failed`" in the message log when you set the qeth device online, broadcast bridging is not available.
2. Unicast packets are routed by the common Linux IPv4 forwarding mechanisms. As bridging and forwarding are done at the IP Level, the IEEE 802.1q VLAN and the IPv6 protocol are not supported.
3. To use HiperSockets Network Concentrator, the s390-tools package from developerWorks is required.

Examples for setting up a network concentrator

An example of a network environment with a network concentrator.

Figure 65 on page 312 shows a network environment where a Linux instance C acts as a network concentrator that connects other operating system instances on a HiperSockets LAN to an external LAN.

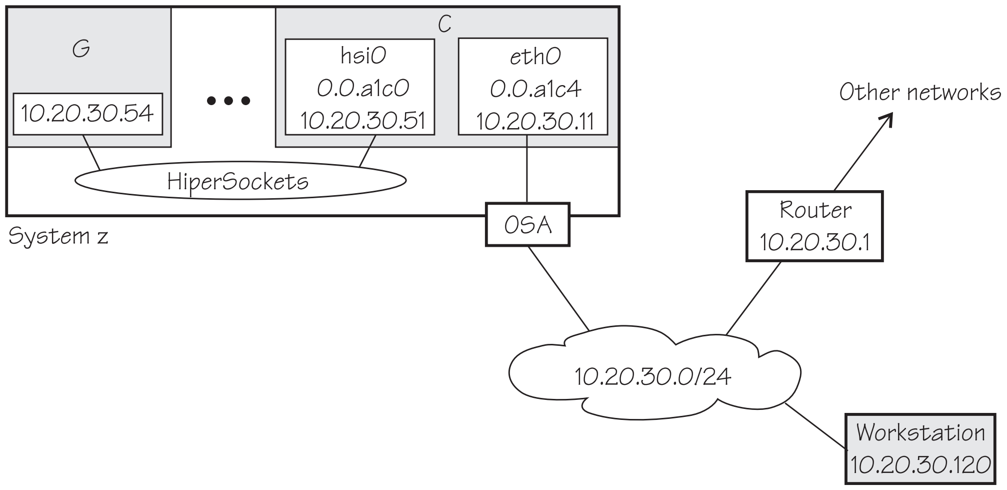


Figure 65. HiperSockets network concentrator setup

Setup for the network concentrator C:

The HiperSockets interface hsi0 (device bus-ID 0.0.a1c0) has IP address 10.20.30.51, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

Issue:

```
# echo primary_connector > /sys/bus/ccwgroup/drivers/qeth/0.0.a1c0/route4
```

The OSA-Express CHPID in QDIO mode interface eth0 (with device bus-ID 0.0.a1c4) has IP address 10.20.30.11, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

Issue:

```
# echo multicast_router > /sys/bus/ccwgroup/drivers/qeth/0.0.a1c4/route4
```

To enable IP forwarding issue:

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

Tip: See your distribution information about using configuration files to automatically enable IP forwarding when Linux boots.

To remove the network routes for the HiperSockets interface issue:

```
# ip route del 10.20.30/24
```

To start the HiperSockets network concentrator, run the script `start_hsnc.sh`. Issue:

```
# start_hsnc.sh &
```

Setup for G:

No special setup required. The HiperSockets interface has IP address 10.20.30.54, 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 66 shows the example of Figure 65 on page 312 with an additional mainframe. On the second mainframe a Linux instance D acts as a HiperSockets network concentrator.

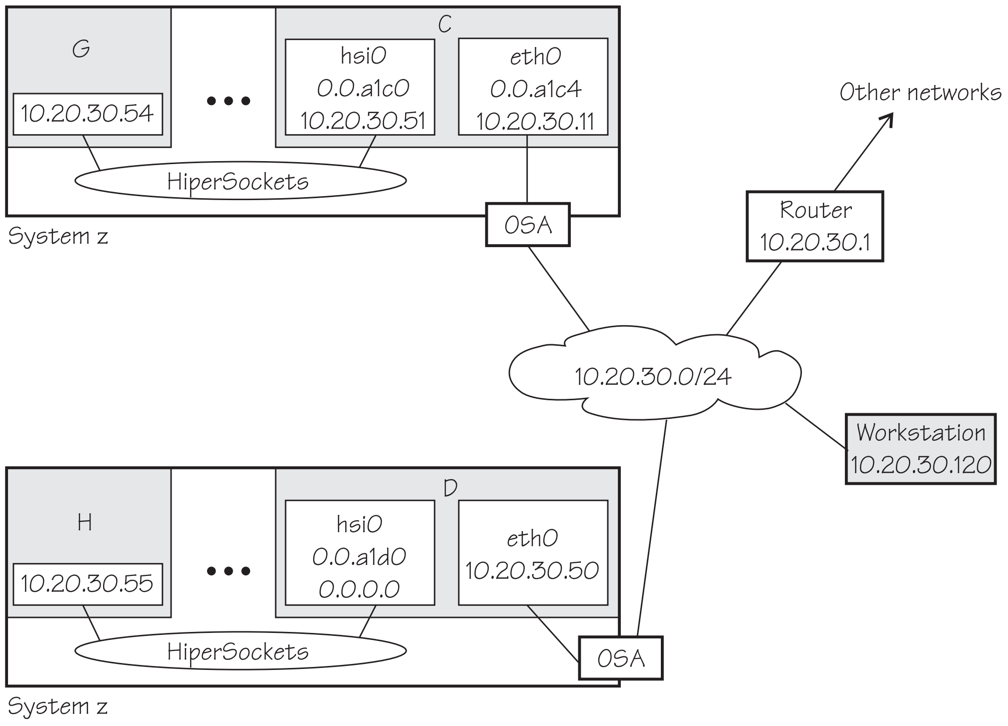


Figure 66. Expanded HiperSockets network concentrator setup

The configuration of C, G, and the workstation remain the same as for Figure 65 on page 312.

Setup for the network concentrator D:

The HiperSockets interface hsi0 has IP address 0.0.0.0.

Assuming that the device bus-ID of the HiperSockets interface is 0.0.a1d0, issue:

```
# echo primary_connector > /sys/bus/ccwgroup/drivers/qeth/0.0.a1d0/route4
```

The OSA-Express CHPID in QDIO mode interface eth0 has IP address 10.20.30.50, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

D is not configured as a multicast router, it therefore forwards only unicast packets.

To enable IP forwarding issue:

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

Tip: See your distribution information about using configuration files to automatically enable IP forwarding when Linux boots.

To start the HiperSockets network concentrator, run the script `start_hsnc.sh`. Issue:

```
# start_hsnc.sh &
```

Setup for H:
No special setup required. The HiperSockets interface has IP address 10.20.30.55, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

Setting up for DHCP with IPv4

For connections through an OSA-Express adapter in QDIO mode, the OSA-Express adapter offloads ARP, MAC header, and MAC address handling.

For information about MAC headers, see “MAC headers in layer 3 mode” on page 262.

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

`www.ietf.org`

Two types of DHCP environments must be taken into account:

- DHCP through OSA-Express adapters in QDIO mode
- DHCP in a z/VM VSWITCH or guest LAN

For information about setting up DHCP for Linux on System z in a z/VM VSWITCH or guest LAN environment, see Redpaper™ *Linux on IBM eServer zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP-3596 at

`www.ibm.com/redbooks`

The programs *dhclient* and *dhcp* are examples of a DHCP client and a DHCP server you can use. The distribution that you use might provide different DHCP client and server programs. For example, some distributions use the *dhcpcd* client program.

Required options for using dhcpcd with layer3

You must configure the DHCP client program `dhclient` to use it on Linux on System z with layer3.

- Run the DHCP client with an option that instructs the DHCP server to broadcast its response to the client.

Because the OSA-Express adapter in QDIO mode forwards packets to Linux based on IP addresses, a DHCP client that requests an IP address cannot receive the response from the DHCP server without this option.

- Run the DHCP client with an option that specifies the client identifier string.

  By default, the client uses the MAC address of the network interface. Hence, without this option, all Linux instances that share the OSA-Express adapter in QDIO mode would also have the same client identifier.

See the documentation for `dhcpcd` about selecting these options.

You need no special options for the DHCP server program, `dhcp`.

Setting up Linux as a LAN sniffer

You can set up a Linux instance to act as a LAN sniffer, for example, to make data on LAN traffic available to tools like **`tcpdump`** or Wireshark.

The LAN sniffer can be:
- A HiperSockets Network Traffic Analyzer for LAN traffic between LPARs
- A LAN sniffer for LAN traffic between z/VM guest virtual machines, for example, through a z/VM virtual switch (VSWITCH)

Setting up a HiperSockets network traffic analyzer

A HiperSockets network traffic analyzer (NTA) runs in an LPAR and monitors LAN traffic between LPARs.

Before you begin

- Your Linux instance must not be a z/VM guest.
- On the SE, the LPARs must be authorized for analyzing and being analyzed.

  Tip: Do any authorization changes before you configure the NTA device. If you must activate the NTA after SE authorization changes, set the qeth device offline, set the sniffer attribute to 1, and set the device online again.
- You need a traffic-dumping tool such as **`tcpdump`**.
- You need a mainframe system that supports HiperSockets network traffic analyzer. HiperSockets network traffic analyzer became available for System z10 in March 2010.

About this task

The HiperSockets NTA is available to trace both layer 3 and layer 2 network traffic, but the analyzing device itself must be configured as a layer 3 device. The analyzing device is a dedicated NTA device and cannot be used as a regular network interface.

Procedure

Perform the following steps:

Linux setup:

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

2. Configure a HiperSockets interface dedicated to analyzing with the `layer2` sysfs attribute set to `0` and the `sniffer` sysfs attribute set to `1`.

   For example, assuming the HiperSockets interface is hsi0 with device bus-ID 0.0.a1c0:

   ```
   # znetconf -a a1c0 -o layer2=0 -o sniffer=1
   ```

   The **`znetconf`** command also sets the device online. For more information about **`znetconf`**, see “znetconf - List and configure network devices” on page 699. The qeth device driver automatically sets the buffer_count attribute to 128 for the analyzing device.
3. Activate the device (no IP address is needed):

   ```
   # ip link set hsi0 up
   ```

4. Switch the interface into promiscuous mode:

   ```
   # tcpdump -i hsi0
   ```

Results

The device is now set up as a HiperSockets network traffic analyzer.

Hint: A HiperSockets network traffic analyzer with no free empty inbound buffers might have to drop packets. Dropped packets are reflected in the "dropped counter" of the HiperSockets network traffic analyzer interface and reported by **`tcpdump`**.

Example:

```
# ip -s link show dev hsi0
...
    RX: bytes  packets  errors  dropped overrun mcast
    223242     6789     0       5       0       176
...
# tcpdump -i hsi0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on hsi1, link-type EN10MB (Ethernet), capture size 96 bytes
...
5 packets dropped by kernel
```

Setting up a z/VM guest LAN sniffer

You can set up a guest LAN sniffer on a virtual NIC that is coupled to a z/VM VSWITCH or guest LAN.

Before you begin

- You need class B authorization on z/VM.
- The Linux instance to be set up as a guest LAN sniffer must run as a guest of the same z/VM system as the guest LAN you want to investigate.

About this task

If a virtual switch connects to a VLAN that includes nodes outside the z/VM system, these external nodes are beyond the scope of the sniffer.

For information about VLANs and z/VM VSWITCHes, see *z/VM Connectivity*, SC24-6174.

Procedure

- Set up Linux.

  Ensure that the qeth device driver has been compiled into the Linux kernel or that the qeth device driver has been loaded as a module.
- Set up z/VM.

  Ensure that the z/VM guest virtual machine on which you want to set up the guest LAN sniffer is authorized for the switch or guest LAN and for promiscuous mode.

  For example, if your virtual NIC is coupled to a z/VM VSWITCH, perform the following steps on your z/VM system:

  1. Check if the z/VM guest virtual machine already has the required authorizations. Enter a CP command of this form:

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

     where <*switchname*> is the name of the virtual switch. If the output lists the z/VM guest virtual machine as authorized for promiscuous mode, no further setup is required.
  2. If the output from step 1 does not list the guest virtual machine, check if the guest is authorized for the virtual switch. Enter a CP command of this form:

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

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

     If the output lists the z/VM guest virtual machine as authorized, you must temporarily revoke the authorization for the switch before you can grant authorization for promiscuous mode. Enter a CP command of this form:

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

     where <*switchname*> is the name of the virtual switch and <*userid*> identifies the z/VM guest virtual machine.
  3. Authorize the Linux instance for the switch and for promiscuous mode. Enter a CP command of this form:

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

     where <*switchname*> is the name of the virtual switch and <*userid*> identifies the z/VM guest virtual machine.

  For details about the CP commands used in this section and for commands you can use to check and assign authorizations for other types of guest LANs, see *z/VM CP Commands and Utilities Reference*, SC24-6175.

Chapter 16. OSA-Express SNMP subagent support

The OSA-Express Simple Network Management Protocol (SNMP) subagent (osasnmpd) supports management information bases (MIBs) for the OSA-Express features.

The subagent supports OSA-Express features as shown in Table 36 on page 254 and Table 37 on page 255.

This subagent capability through the OSA-Express features is also called *Direct SNMP* to distinguish it from another method of accessing OSA SNMP data through OSA/SF, a package for monitoring and managing OSA features that does not run on Linux.

See "osasnmpd – Start OSA-Express SNMP subagent" on page 654 for information about the **osasnmpd** command itself.

To use the osasnmpd subagent, you need:
- An OSA-Express feature that runs in QDIO mode with the latest textual MIB file for the appropriate LIC level (recommended)
- The qeth device driver for OSA-Express (QDIO)
- The osasnmpd subagent from s390-tools
- One of:
  - net-snmp package 5.1.x or higher
  - ucd-snmp package 4.2.x (if possible, use 4.2.3 or higher)

What you should know about osasnmpd

The osasnmpd subagent requires a master agent to be installed on a Linux system.

You get the master agent from either the net-snmp or the ucd-snmp package. The subagent uses the Agent eXtensibility (AgentX) protocol to communicate with the master agent.

net-snmp/ucd-snmp is an open source project that is owned by the Open Source Development Network, Inc. (OSDN). For more information on net-snmp/ucd-snmp visit

```
net-snmp.sourceforge.net
```

When the master agent (snmpd) is started on a Linux system, it binds to a port (default 161) and awaits requests from SNMP management software. Subagents can connect to the master agent to support MIBs of special interest (for example, OSA-Express MIB). When the osasnmpd subagent is started, it retrieves the MIB objects of the OSA-Express features currently present on the Linux system. It then registers with the master agent the object IDs (OIDs) for which it can provide information.

An OID is a unique sequence of dot-separated numbers (for example, .1.3.6.1.4.1.2) that represents a particular information. OIDs form a hierarchical structure. The longer the OID, that is the more numbers it is made up of, the more specific is the information that is represented by the OID. For example, .1.3.6.1.4.1.2 represents all IBM-related network information while ..1.3.6.1.4.1.2.6.188 represents all OSA-Express-related information.

A MIB corresponds to a number of OIDs. MIBs provide information on their OIDs including textual representations the OIDs. For example, the textual representation of .1.3.6.1.4.1.2 is .iso.org.dod.internet.private.enterprises.ibm.

The structure of the MIBs might change when updating the OSA-Express Licensed Internal Code (LIC) to a newer level. If MIB changes are introduced by a new LIC level, you must to download the appropriate MIB file for the LIC level (see "Downloading the IBM OSA-Express MIB" on page 321). You do not need to update the subagent. Place the updated MIB file in a directory that is searched by the master agent.

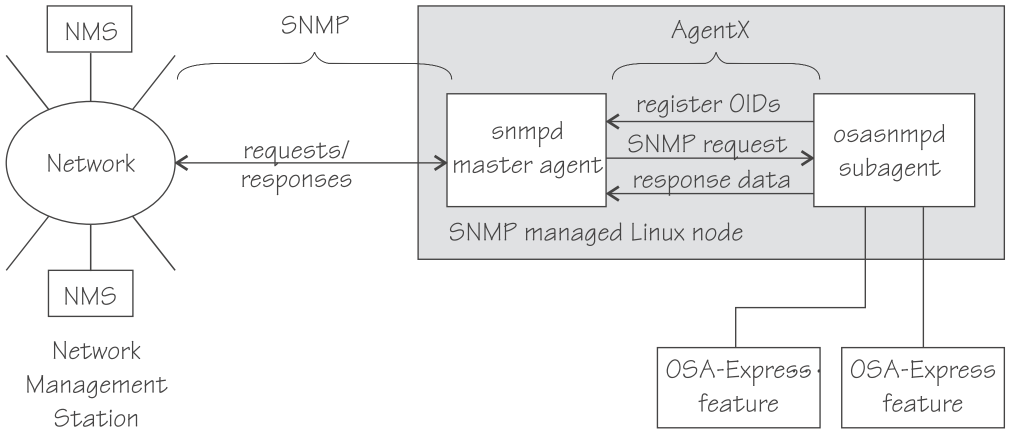


Figure 67. OSA-Express SNMP agent flow

Figure 67 illustrates the interaction between the snmpd master agent and the osasnmpd subagent.

Example: This example shows the processes that run after the snmpd master agent and the osasnmpd subagent are started. When you start osasnmpd, a daemon called osasnmpd starts. In the example, PID 687 is the SNMP master agent and PID 729 is the OSA-Express SNMP subagent process:

```
ps -ef | grep snmp

USER       PID
root       687     1  0 11:57 pts/1    00:00:00 snmpd
root       729   659  0 13:22 pts/1    00:00:00 osasnmpd
```

When the master agent receives an SNMP request for an OID that is registered by a subagent, the master agent uses the subagent to collect any requested information and to perform any requested operations. The subagent returns any requested information to the master agent. Finally, the master agent returns the information to the originator of the request.

Setting up osasnmpd

You must download the IBM OSA-Express MIB and configure access control before you use can the osasnmpd subagent.

Downloading the IBM OSA-Express MIB

Keep your MIB file up to date by downloading the latest version.

About this task

Perform the following steps to download the IBM OSA-Express MIB. The MIB file is valid only for hardware that supports the OSA-Express adapter.

Procedure

1. Go to www.ibm.com/servers/resourcelink

   A user ID and password are required. If you do not yet have one, you can apply for a user ID.
2. Sign in.
3. Select **Library** from the navigation area.
4. Under **Library shortcuts**, select **Open Systems Adapter (OSA) Library**.
5. Follow the link for **OSA-Express Direct SNMP MIB module**.
6. Select and download the MIB for your LIC level.
7. Rename the MIB file to the name specified in the MIBs definition line and use the extension `.txt`.

   Example: If the definition line in the MIB looks like this:
   ```
   ==>IBM-OSA-MIB DEFINITIONS ::= BEGIN
   ```
   Rename the MIB to `IBM-OSA-MIB.txt`.
8. Place the MIB into `/usr/share/snmp/mibs`. If you want to use a different directory, be sure to specify the directory in the snmp.conf configuration file (see step 10 on page 324).

Results

You can now make the OID information from the MIB file available to the master agent. You can then use textual OIDs instead of numeric OIDs when you use master agent commands.

See also the FAQ (How do I add a MIB to the tools?) for the master agent package at

```
net-snmp.sourceforge.net/FAQ.html
```

Configuring access control

To start successfully, the subagent requires at least read access to the standard MIB-II on the local node.

About this task

During subagent startup or when network interfaces are added or removed, the subagent must query OIDs from the interfaces group of the standard MIB-II.

Given here is an example of how to use the snmpd.conf and snmp.conf configuration files to assign access rights with the View-Based Access Control Mechanism (VACM). The following access rights are assigned on the local node:

- General read access for the scope of the standard MIB-II
- Write access for the scope of the OSA-Express MIB

- Public local read access for the scope of the interfaces MIB

The example is intended for illustration purposes only. Depending on the security requirements of your installation, you might need to define your access differently. See the snmpd man page for a more information about assigning access rights to snmpd.

Procedure

1. See your distribution documentation to find out where you can find a template for snmpd.conf and where you must place it.

   Some of the possible locations are:
   - `/usr/local/share/snmp`
   - `/etc/snmp`
   - `/usr/share/snmp`
2. Open snmpd.conf with your preferred text editor.
3. Find the security name section and include a line of this form to map a community name to a security name:

   ```
   com2sec <security-name> <source> <community-name>
   ```

   where:

   <security-name>
   : is given access rights through further specifications within snmpd.conf.

   <source>
   : is the IP address or DNS name of the accessing system, typically a Network Management Station.

   <community-name>
   : is the community string used for basic SNMP password protection.

   Example:

   ```
   #       sec.name    source      community
   com2sec osasec      default     osacom
   com2sec pubsec      localhost   public
   ```
4. Find the group section.

   Use the security name to define a group with different versions of the master agent for which you want to grant access rights. Include a line of this form for each master agent version:

   ```
   group <group-name> <security-model> <security-name>
   ```

   where:

   <group-name>
   : is a group name of your choice.

   <security-model>
   : is the security model of the SNMP version.

   <security-name>
   : is the same as in step 3.

   Example:

```
#        groupName    securityModel    securityName
group    osagroup     v1               osasec
group    osagroup     v2c              osasec
group    osagroup     usm              osasec
group    osasnmpd     v2c              pubsec
```

Group “osasnmpd” with community “public” is required by osasnmpd to determine the number of network interfaces.

5. Find the view section and define your views. A view is a subset of all OIDs. Include lines of this form:

   ```
   view  <view-name>  <included|excluded>   <scope>
   ```

   where:

   <view-name>
   : is a view name of your choice.

   <included|excluded>
   : indicates whether the following scope is an inclusion or an exclusion statement.

   <scope>
   : specifies a subtree in the OID tree.

   Example:

   ```
   #    name       incl/excl    subtree                 mask(optional)
   view allview    included     .1
   view osaview    included     .1.3.6.1.4.1.2
   view ifmibview  included     interfaces
   view ifmibview  included     system
   ```

   View `allview` encompasses all OIDs while `osaview` is limited to IBM OIDs. The numeric OID provided for the subtree is equivalent to the textual OID “.iso.org.dod.internet.private.enterprises.ibm”. View “ifmibview” is required by osasnmpd to determine the number of network interfaces.

   Tip: Specifying the subtree with a numeric OID leads to better performance than using the corresponding textual OID.

6. Find the access section and define access rights. Include lines of this form:

   ```
   access <group-name> "" any noauth exact <read-view> <write-view> none
   ```

   where:

   <group-name>
   : is the group that you defined in step 4 on page 322.

   <read-view>
   : is a view for which you want to assign read-only rights.

   <write-view>
   : is a view for which you want to assign read/write rights.

   Example:

   ```
   #      group    context sec.model sec.level prefix read      write   notif
   access osagroup ""      any       noauth    exact  allview   osaview none
   access osasnmpd ""      v2c       noauth    exact  ifmibview none    none
   ```

   The access line of the example gives read access to the `allview` view and write access to the `osaview`. The second access line gives read access to the `ifmibview`.

7. Also, include the following line to enable the AgentX support:

```
master agentx
```

By default, AgentX support is compiled into the net-snmp master agent 5.1.x and, as of version 4.2.2, also into the ucd-snmp master agent.

8. Save and close snmpd.conf.
9. Open snmp.conf with your preferred text editor.
10. Include a line of this form to specify the directory to be searched for MIBs:

    ```
    mibdirs +<mib-path>
    ```

    Example:

    ```
    mibdirs +/usr/share/snmp/mibs
    ```

11. Include a line of this form to make the OSA-Express MIB available to the master agent:

    ```
    mibs +<mib-name>
    ```

    where <*mib-name*> is the stem of the MIB file name you assigned in “Downloading the IBM OSA-Express MIB” on page 321.

    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 that you defined in step 3 on page 322.

    Example:

    ```
    defVersion   2c
    defCommunity osacom
    ```

    These default specifications simplify issuing master agent commands.

13. Save and close snmp.conf.

Working with the osasnmpd subagent

Working with the osasnmpd subagent includes starting it, checking the log file, issuing queries, and stopping the subagent.

Working with osasnmpd comprises the following tasks:

- “Starting the osasnmpd subagent”
- “Checking the log file” on page 325
- “Issuing queries” on page 325
- “Stopping osasnmpd” on page 326

Starting the osasnmpd subagent

Use the **osasnmpd** command to start the osasnmpd subagent.

Procedure

Start the osasnmpd subagent with the **osasnmpd** command:

```
# osasnmpd
```

The osasnmpd subagent starts a daemon that is called osasnmpd.
For command options see "osasnmpd – Start OSA-Express SNMP subagent" on page 654.
If you restart the master agent, you must also restart the subagent. When the master agent is started, it does not look for already running subagents. Any running subagents must also be restarted to be register with the master agent.

Checking the log file

Warnings and messages are written to the log file of either the master agent or the OSA-Express subagent. It is good practice to check these files at regular intervals.

Example

This example assumes that the default subagent log file is used. The lines in the log file show the messages after a successful OSA-Express subagent initialization.

```
# cat /var/log/osasnmpd.log
IBM OSA-E NET-SNMP 5.1.x subagent version  1.3.0
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.2.1.10.7.2.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.1.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.3.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.4.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.8.
OSA-E microcode level is 611 for interface eth0
Initialization of OSA-E subagent successful...
```

Issuing queries

You can issue queries against your SNMP setup.

About this task

Examples of what SNMP queries might look like are given here. For more comprehensive information about the master agent commands see the **snmpcmd** man page.

The commands can use either numeric or textual OIDs. While the numeric OIDs might provide better performance, the textual OIDs are more meaningful and give a hint about which information is requested.

Examples

The query examples assume an interface, eth0, for which the CHPID is 6B. You can use the **lsqeth** command to find the mapping of interface names to CHPIDs.

- To list the ifIndex and interface description relation (on one line):

```
# snmpget -v 2c -c osacom localhost interfaces.ifTable.ifEntry.ifDescr.6
interfaces.ifTable.ifEntry.ifDescr.6 = 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 is authorized as described in “Configuring access control” on page 321.

If you provided defaults for the SNMP version and the community (see step 12 on page 324), you can omit the `-v` and `-c` options:

```
# snmpwalk -OS localhost .1.3.6.1.4.1.2.6.188.1.1.1.1
IBM-OSA-MIB::ibmOSAExpChannelNumber.6 = Hex-STRING: 00 6B
IBM-OSA-MIB::ibmOSAExpChannelNumber.7 = Hex-STRING: 00 7A
IBM-OSA-MIB::ibmOSAExpChannelNumber.8 = Hex-STRING: 00 7D
```

You can obtain the same output by substituting the numeric OID .1.3.6.1.4.1.2.6.188.1.1.1.1 with its textual equivalent:

```
.iso.org.dod.internet.private.enterprises.ibm.ibmProd.ibmOSAMib.ibmOSAMibObjects.ibmOSAExpChannelTable.ibmOSAExpChannelEntry.ibmOSAExpChannelNumber
```

You can shorten this unwieldy OID to the last element, ibmOsaExpChannelNumber:

```
# snmpwalk -OS localhost ibmOsaExpChannelNumber
IBM-OSA-MIB::ibmOSAExpChannelNumber.6 = Hex-STRING: 00 6B
IBM-OSA-MIB::ibmOSAExpChannelNumber.7 = Hex-STRING: 00 7A
IBM-OSA-MIB::ibmOSAExpChannelNumber.8 = Hex-STRING: 00 7D
```

- To find the port type for the interface with index number 6:

```
# snmpwalk -OS localhost .1.3.6.1.4.1.2.6.188.1.4.1.2.6
IBM-OSA-MIB::ibmOsaExpEthPortType.6 = INTEGER: fastEthernet(81)
```

fastEthernet(81) corresponds to card type OSD_100.

Using the short form of the textual OID:

```
# snmpwalk -OS localhost ibmOsaExpEthPortType.6
IBM-OSA-MIB::ibmOsaExpEthPortType.6 = INTEGER: fastEthernet(81)
```

Specifying the index, 6 in the example, limits the output to the interface of interest.

Stopping osasnmpd

The subagent can be stopped by sending either a SIGINT or SIGTERM signal to the thread.

About this task

Avoid stopping the subagent with **`kill -9`** or with **`kill -SIGKILL`**. These commands do not allow the subagent to unregister the OSA-Express MIB objects from the SNMP master agent. This can cause problems when restarting the subagent.

If you saved the subagent PID to a file when you started it, you can consult this file for the PID (see “osasnmpd – Start OSA-Express SNMP subagent” on page 654). Otherwise, you can issue a **`ps`** command to find it out.

Example

The osasnmpd subagent starts a daemon that is called osasnmpd. To stop osasnmpd, issue the **`kill`** command for either the daemon or its PID:

```
# ps -ef | grep snmp

USER       PID
root       687     1  0 11:57 pts/1    00:00:00 snmpd
root       729   659  0 13:22 pts/1    00:00:00 osasnmpd
# killall osasnmpd
```

Chapter 17. LAN channel station device driver

The LAN channel station device driver (LCS device driver) supports Open Systems Adapters (OSA) features in non-QDIO mode.

Table 49 shows the supported OSA-Express features.

Table 49. The LCS device driver supported OSA features

Feature	zEC12 and zBC12	z196, z114, and System z10	System z9	eServer zSeries
OSA-Express4S	1000Base-T Ethernet	Not supported	Not supported	Not supported
OSA-Express3	1000Base-T Ethernet	1000Base-T Ethernet	Not supported	Not supported
OSA-Express2	Not supported	1000Base-T Ethernet	1000Base-T Ethernet	Not supported
OSA-Express	Not supported	Not supported	Fast Ethernet 1000Base-T Ethernet	Fast Ethernet 1000Base-T Ethernet (z890, z990)

The LCS device driver supports automatic detection of Ethernet connections. The LCS device driver can be used for Internet Protocol, version 4 (IPv4) only.

What you should know about LCS

Interface names are assigned to LCS group devices, which map to subchannels and their corresponding device numbers and device bus-IDs.

LCS group devices

The LCS device driver requires two I/O subchannels for each LCS interface, a read subchannel and a write subchannel. The corresponding bus IDs must be configured for control unit type 3088.

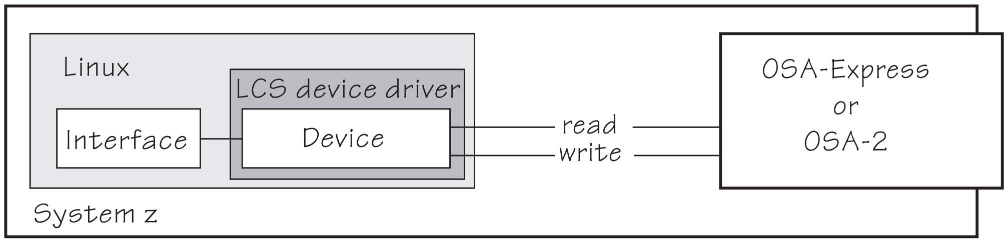


Figure 68. I/O subchannel interface

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

read must be even.

write must be the device bus-ID of the read subchannel plus one.

LCS interface names

When an LCS group device is set online, the LCS device driver automatically assigns an Ethernet interface name to it.

The naming scheme uses the base name eth<*n*>, 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", and the second "eth1".

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 260.

Building a kernel with the LCS device driver

If you want to build your own kernel, there are certain options that you must select in the Linux configuration menu to include the LCS device driver.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

You must select the option CONFIG_LCS if you want to work with LCS devices (see Figure 69).

```
Device Drivers --->
   ...
    Network device support --->          (common code options CONFIG_NETDEVICES)
      ...
      S/390 network device drivers (depends on NETDEVICES && S390) --->
         ...
         Lan Channel Station Interface                        (CONFIG_LCS)
```

Figure 69. LCS kernel configuration menu option

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

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

Setting up the LCS device driver

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

If you compiled the LCS component as a separate module, you must load it before you can work with LCS devices. Load the lcs module with the **modprobe** command to ensure that any other required modules are loaded in the correct order:

```
# modprobe lcs
```

Working with LCS devices

Working with LCS devices includes tasks such as creating an LCS group device, specifying a timeout, or activating an interface.

- "Creating an LCS group device"
- "Removing an LCS group device" on page 332
- "Specifying a timeout for LCS LAN commands" on page 332
- "Setting an LCS group device online or offline" on page 333
- "Activating and deactivating an interface" on page 334

Creating an LCS group device

Use the `group` attribute to create an LCS group device.

Before you begin

You must know the device bus-IDs that corresponds to the read and write subchannel of your OSA card. The subchannel is defined in the IOCDS of your mainframe.

Procedure

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

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

Results

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

`/sys/bus/ccwgroup/drivers/lcs/<read_device_bus_id>`

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

Note: When the device subchannels are added, device types 3088/08 and 3088/1f can be assigned to either the CTCM or the LCS device driver.

To check which devices are assigned to which device driver, issue the following commands:

```
# ls -l /sys/bus/ccw/drivers/ctcm
# ls -l /sys/bus/ccw/drivers/lcs
```

To change a faulty assignment, use the unbind and bind attributes of the device. For example, to change the assignment for device bus-IDs `0.0.2000` and `0.0.2001` issue the following commands:

```
# echo 0.0.2000 > /sys/bus/ccw/drivers/ctcm/unbind
# echo 0.0.2000 > /sys/bus/ccw/drivers/lcs/bind
# echo 0.0.2001 > /sys/bus/ccw/drivers/ctcm/unbind
# echo 0.0.2001 > /sys/bus/ccw/drivers/lcs/bind
```

Example

Assuming that 0.0.d000 is the device bus-ID that corresponds to a read subchannel:

```
# echo 0.0.d000,0.0.d001 > /sys/bus/ccwgroup/drivers/lcs/group
```

This command results in the creation of the following directories in sysfs:

- `/sys/bus/ccwgroup/drivers/lcs/0.0.d000`
- `/sys/bus/ccwgroup/devices/0.0.d000`
- `/sys/devices/lcs/0.0.d000`

Removing an LCS group device

Use the `ungroup` attribute to remove an LCS group device.

Before you begin

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

Procedure

To remove an LCS group device, write 1 to the `ungroup` attribute. Issue a command of the form:

```
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/<device_bus_id>/ungroup
```

Example

This command removes device 0.0.d000:

```
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/ungroup
```

Specifying a timeout for LCS LAN commands

Use the `lancmd_timeout` attribute to set a timeout for an LCS LAN command.

About this task

You can specify a timeout for the interval that the LCS device driver waits for a reply after issuing a LAN command to the LAN adapter. For older hardware, the replies can take a longer time. The default is 5 s.

Procedure

To set a timeout, issue a command of this form:

```
# echo <timeout> > /sys/bus/ccwgroup/drivers/lcs/<device_bus_id>/lancmd_timeout
```

where <*timeout*> is the timeout interval in seconds in the range 1 - 60.

Example

In this example, the timeout for a device 0.0.d000 is set to 10 s.

```
# echo 10 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/lancmd_timeout
```

Setting an LCS group device online or offline

Use the `online` device group attribute to set an LCS device online or offline.

About this task

Setting a device online associates it with an interface name. Setting the device offline preserves the interface name.

You must know the interface name to activate the network interface. To determine the assigned interface name, use the **`znetconf -c`** command. For each online interface, the interface name is shown in the Name column. Alternatively, to determine the assigned interface name issue a command of the form:

```
# ls /sys/devices/lcs/<device_bus_id>/net/
```

Procedure

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

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

Example

To set an LCS device with bus ID 0.0.d000 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/online
```

To determine the interface name issue:

```
# znetconf -c
Device IDs                 Type    Card Type      CHPID Drv. Name       State
--------------------------------------------------------------------------------
0.0.d000,0.0.d001          3088/60 OSA LCS card         lcs  eth0       online
```

or

```
# ls /sys/devices/lcs/0.0.d000/net/
eth0
...
```

The interface name that was assigned to the LCS group device in the example is eth0.

For each online interface, there is a symbolic link of the form /sys/class/net/<interface_name>/device in sysfs. You can confirm that you found the correct interface name by reading the link:

```
# readlink /sys/class/net/eth0/device
../../../0.0.d000
```

To set the device offline issue:

```
# echo 0 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/online
```

Activating and deactivating an interface

Use the **ip** command or equivalent to activate or deactivate an interface.

About this task

Before you can activate an interface, you must set the group device online and found out the interface name that is assigned by the LCS device driver. See “Setting an LCS group device online or offline” on page 333.

You activate or deactivate network devices with **ip** or an equivalent command. For details of the **ip** command, see the **ip** man page.

Examples

- This example activates an Ethernet interface:

```
# ip addr add 192.168.100.10/24 dev eth0
# ip link set dev eth0 up
```

- This example deactivates the Ethernet interface:

```
# ip link set dev eth0 down
```

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

```
# ip link set dev eth0 up
```

Recovering an LCS group device

You can use the recover attribute of an LCS group device to recover it in case of failure. For example, error messages in /var/log/messages might inform you of a malfunctioning device.

Procedure

Issue a command of the form:

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

Example

```
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/0.0.d100/recover
```

Chapter 18. 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 Linux on System z to VTAM® on traditional mainframe operating systems.

Features

The CTCM device driver provides different kinds of CTC connections between mainframes, z/VM guests, and LPARs.

The CTCM device driver provides:

- MPC connections to VTAM on traditional mainframe operating systems.
- ESCON or FICON CTC connections (standard CTC and basic CTC) between mainframes in basic mode, LPARs or z/VM guests.

  For more information about FICON, see Redpaper *FICON CTC Implementation*, REDP-0158.
- Virtual CTCA connections between guests of the same z/VM system.
- CTC connections to other Linux instances or other mainframe operating systems.

What you should know about CTCM

The CTCM device driver assigns network interface names to CTCM group devices.

CTCM group devices

The CTCM device driver requires two I/O subchannels for each interface, a read subchannel and a write subchannel.

Figure 70 on page 336 illustrates the I/O subchannel interface. The device bus-IDs that correspond to the two subchannels must be configured for control unit type 3088.

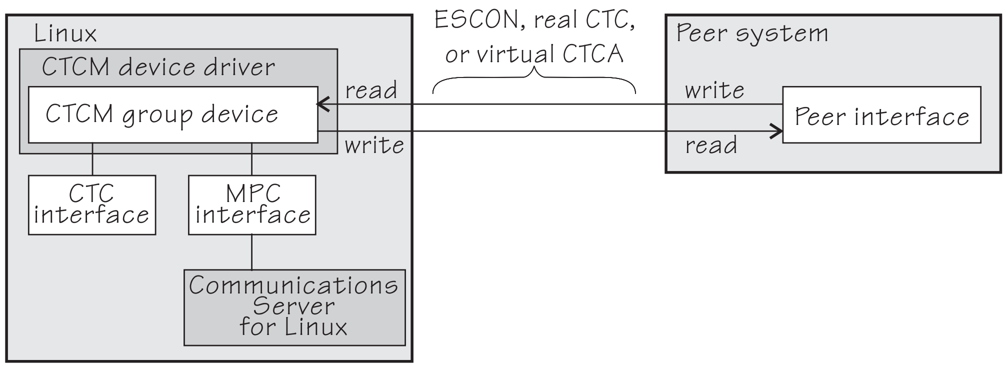


Figure 70. I/O subchannel interface

The device bus-IDs that correspond to the subchannel pair are grouped as one CTCM group device. There are no constraints on the device bus-IDs of read subchannel and write subchannel. In particular, it is possible to group non-consecutive device bus-IDs.

On the communication-peer operating system instance, read and write subchannels are reversed. That is, the write subchannel of the local interface is connected to the read subchannel of the remote interface and vice versa.

Depending on the protocol, the interfaces can be CTC interfaces or MPC interfaces. MPC interfaces are used by Communications Server for Linux and connect to peer interfaces that run under VTAM. For more information about Communications Server for Linux and on using MPC connections, go to

`www.ibm.com/software/network/commserver/linux`

Interface names assigned by the CTCM device driver

When a CTCM group device is set online, the CTCM device driver automatically assigns an interface name to it. The interface name depends on the protocol.

If the protocol is set to `4`, you get an MPC connection and the interface names are of the form `mpc<`*n*`>`.

If the protocol is set to `0`, `1`, or `3`, you get a CTC connection and the interface name is of the form `ctc<`*n*`>`.

<*n*> is an integer that identifies the device. When the first device is set online it is assigned 0, the second is assigned 1, the third 2, and so on. The devices are counted separately for CTC and MPC.

Network connections

If your CTC connection is to a router or z/VM TCP/IP service machine, you can connect CTC interfaces to an external network.

This section applies to CTC interfaces only.

Figure 71 on page 337 shows a CTC interface that is connected to a network.

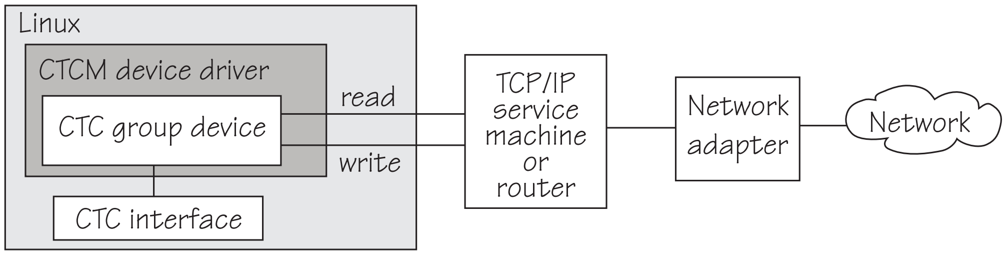


Figure 71. Network connection

Building a kernel with the CTCM device driver

Control the build options for the CTCM device driver through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

You must select the kernel configuration option CONFIG_CTCM to use CTCM connections (see Figure 72).

```
Device Drivers --->
    ...
     Network device support --->          (common code option CONFIG_NETDEVICES)
       ...
       S/390 network device drivers (depends on NETDEVICES && S390) --->
          ...
          CTC and MPC SNA device support                       (CONFIG_CTCM)
```

Figure 72. CTCM kernel configuration menu option

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

Setting up the CTCM device driver

You do not need to specify kernel or module parameters for the CTCM device driver, but you might need to load the ctcm module.

If the CTCM device driver was compiled as a separate module, load it with the **modprobe** command to ensure that any other required modules are loaded:

```
# modprobe ctcm
```

Working with CTCM devices

When you work with CTCM devices you might create a CTCM group device, set the protocol, and activate an interface.

The following sections describe typical tasks that you need when you work with CTCM devices.

- “Creating a CTCM group device”
- “Removing a CTCM group device” on page 339
- “Displaying the channel type” on page 339
- “Setting the protocol” on page 340
- “Setting a device online or offline” on page 340
- “Setting the maximum buffer size” on page 341 (CTC only)
- “Activating and deactivating a CTC interface” on page 342 (CTC only)
- “Recovering a lost CTC connection” on page 343 (CTC only)

See the Communications Server for Linux documentation for information about configuring and activating MPC interfaces.

Creating a CTCM group device

Use the `group` attribute to create a CTCM group device.

Before you begin

You must know the device bus-IDs that correspond to the local read and write subchannel of your CTCM connection as defined in your IOCDS.

Procedure

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

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

Results

The CTCM device driver uses the device bus-ID of the read subchannel to create a directory for a group device:
`/sys/bus/ccwgroup/drivers/ctcm/<read_device_bus_id>`

This directory contains a number of attributes that determine the settings of the CTCM group device.

Note: When the device subchannels are added, device types 3088/08 and 3088/1f can be assigned to either the CTCM or the LCS device driver.

To check which devices are assigned to which device driver, issue the following commands:

```
# ls -l /sys/bus/ccw/drivers/ctcm
# ls -l /sys/bus/ccw/drivers/lcs
```

To change a faulty assignment, use the unbind and bind attributes of the device. For example, to change the assignment for device bus-IDs `0.0.2000` and `0.0.2001` issue the following commands:

```
# echo 0.0.2000 > /sys/bus/ccw/drivers/lcs/unbind
# echo 0.0.2000 > /sys/bus/ccw/drivers/ctcm/bind
# echo 0.0.2001 > /sys/bus/ccw/drivers/lcs/unbind
# echo 0.0.2001 > /sys/bus/ccw/drivers/ctcm/bind
```

Example

Assuming that device bus-ID 0.0.2000 corresponds to a read subchannel:

```
# echo 0.0.2000,0.0.2001 > /sys/bus/ccwgroup/drivers/ctcm/group
```

This command results in the creation of the following directories in sysfs:
- `/sys/bus/ccwgroup/drivers/ctcm/0.0.2000`
- `/sys/bus/ccwgroup/devices/0.0.2000`
- `/sys/devices/ctcm/0.0.2000`

Removing a CTCM group device

Use the `ungroup` attribute to remove a CTCM group device.

Before you begin

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

Procedure

To remove a CTCM group device, write 1 to the `ungroup` attribute. Issue a command of the form:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/ungroup
```

Example

This command removes device 0.0.2000:

```
echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/ungroup
```

Displaying the channel type

Use the `type` attribute to display the channel type of a CTCM group device.

Procedure

Issue a command of this form to display the channel type of a CTCM group device:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/type
```

where <*device_bus_id*> is the device bus-ID that corresponds to the CTCM read channel. Possible values are: CTC/A, ESCON, and FICON.

Example

In this example, the channel type is displayed for a CTCM group device with device bus-ID 0.0.f000:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f000/type
ESCON
```

Setting the protocol

Use the `protocol` attribute to set the protocol.

Before you begin

The device must be offline while you set the protocol.

About this task

The type of interface depends on the protocol. Protocol `4` results in MPC interfaces with interface names `mpc<n>`. Protocols `0`, `1`, or `3` result in CTC interfaces with interface names of the form `ctc<n>`.

To choose a protocol, set the protocol attribute to one of the following values:

0 This protocol provides compatibility with peers other than OS/390®, or z/OS, for example, a z/VM TCP service machine. This value is the default.

1 This protocol provides enhanced package checking for Linux peers.

3 This protocol provides for compatibility with OS/390 or z/OS peers.

4 This protocol provides for MPC connections to VTAM on traditional mainframe operating systems.

Procedure

Issue a command of this form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/protocol
```

Example

In this example, the protocol is set for a CTCM group device 0.0.2000:

```
# echo 4 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/protocol
```

Setting a device online or offline

Use the `online` device group attribute to set a CTCM device online or offline.

About this task

Setting a group device online associates it with an interface name. Setting the group device offline and back online with the same protocol preserves the association with the interface name. If you change the protocol before you set the group device back online, the interface name can change as described in "Interface names assigned by the CTCM device driver" on page 336.

You must know the interface name to access the CTCM group device. To determine the assigned interface name, use the **znetconf -c** command. For each online interface, the interface name is shown in the Name column. Alternatively, to determine the assigned interface name issue a command of the form:

```
# ls /sys/devices/ctcm/<device_bus_id>/net/
```

For each online interface, there is a symbolic link of the form `/sys/class/net/<interface_name>/device` in sysfs. You can confirm that you found the correct interface name by reading the link.

Procedure

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

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

Example

To set a CTCM device with bus ID 0.0.2000 online issue:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/online
```

To determine the interface name issue:

```
# znetconf -c
Device IDs                 Type    Card Type      CHPID Drv. Name        State
--------------------------------------------------------------------------------
0.0.2000,0.0.2001          3088/08 CTC/A                ctcm ctc0        online
```

or

```
# ls /sys/devices/ctcm/0.0.2000/net/
ctc0
```

The interface name that was assigned to the CTCM group device in the example is mpc0. To confirm that this name is the correct one for the group device issue:

```
# readlink /sys/class/net/mpc0/device
../../../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

Use the `buffer` device group attribute to set a maximum buffer size for a CTCM group device.

Before you begin

- Set the maximum buffer size for CTC interfaces only. MPC interfaces automatically use the highest possible maximum buffer size.
- The device must be online when you set the buffer size.

About this task

You can set the maximum buffer size for a CTC interface. The permissible range of values depends on the MTU settings. It must be in the range <*minimum MTU* +

header size> to <*maximum MTU* + *header size*>. The header space is typically 8 bytes. The default for the maximum buffer size is 32768 byte (32 KB).

Changing the buffer size is accompanied by an MTU size change to the value <*buffer size - header size*>.

Procedure

To set the maximum buffer size, issue a command of this form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/buffer
```

where <*value*> is the number of bytes you want to set. If you specify a value outside the valid range, the command is ignored.

Example

In this example, the maximum buffer size of a CTCM group device 0.0.f000 is set to 16384 byte.

```
# echo 16384 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f000/buffer
```

Activating and deactivating a CTC interface

Use **ip** or an equivalent command to activate or deactivate an interface.

Before you begin

- Activate and deactivate a CTC interfaces only. For information about activating MPC interfaces, see the Communications Server for Linux documentation.
- You must know the interface name. See “Setting a device online or offline” on page 340.

About this task

Syntax for setting an IP address for a CTC interface with the ip command

```
►►──ip address── add <ip_address>── dev <interface>──────────────────────────►

►─┬──────────────────────────┬────────────────────────────────────────────────►◄
  └─ peer <peer_ip_address>──┘
```

Syntax for activating a CTC interface with the ip command

```
                                             ┌─ mtu 32760──────────────────┐
►►──ip link set── dev <interface>── up───────┼─────────────────────────────┼──────►◄
                                             └─ mtu <max_transfer_unit>────┘
```

Where:

<interface>
: is the interface name that was assigned when the CTCM group device was set online.

<ip_address>
: is the IP address that you want to assign to the interface.

<peer_ip_address>
: is the IP address of the remote side.

<max_transfer_unit>
: is the size of the largest IP packet that might be transmitted. Be sure to use the same MTU size on both sides of the connection. The MTU must be in the range of 576 byte to 65,536 byte (64 KB).

Syntax for deactivating a CTC interface with the ip command

```
►►──ip link set── dev <interface>── down────────────────────────────────────►◄
```

Where:

<interface>
: is the interface name that was assigned when the CTCM group device was set online.

Procedure

- Use **ip** or an equivalent command to activate the interface:
- To deactivate an interface, issue a command of this form:

```
# ip link set dev <interface> down
```

Examples

- This example activates a CTC interface ctc0 with an IP address 10.0.51.3 for a peer with address 10.0.50.1 and an MTU of 32760.

```
# ip addr add 10.0.51.3 dev ctc0 peer 10.0.50.1
# ip link set dev ctc0 up mtu 32760
```

- This example deactivates ctc0:

```
# ip link set dev ctc0 down
```

Recovering a lost CTC connection

If one side of a CTC connection crashes, you cannot simply reconnect after a reboot. You must also deactivate the interface of the peer of the crashed side.

Before you begin

These instructions apply to CTC interfaces only.

Procedure

Proceed as follows to recover a lost CTC connection:

1. Reboot the crashed side.
2. Deactivate the interface on the peer. See "Activating and deactivating a CTC interface" on page 342.
3. Activate the interface on the crashed side and on the peer. For details, see "Activating and deactivating a CTC interface" on page 342.

   If the connection is between a Linux instance and a non-Linux instance, activate the interface on the Linux instance first. Otherwise, you can activate the interfaces in any order.

Results

If the CTC connection is uncoupled, you must couple it again and reconfigure the interface of both peers with the **ip** command. See "Activating and deactivating a CTC interface" on page 342.

Scenarios

Typical use cases of CTC connections include connecting to a peer in a different LPAR and connecting Linux instances that run as z/VM guests to each other.

- "Connecting to a peer in a different LPAR"
- "Connecting Linux on z/VM to another guest of the same z/VM system" on page 346

Connecting to a peer in a different LPAR

A Linux instance and a peer both run in LPAR mode on the same or on different mainframes. They are to be connected with a CTC FICON or CTC ESCON network interface.

Assumptions:

- Locally, the read and write channels are 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 73 on page 345 illustrates a CTC setup with a peer in a different LPAR.

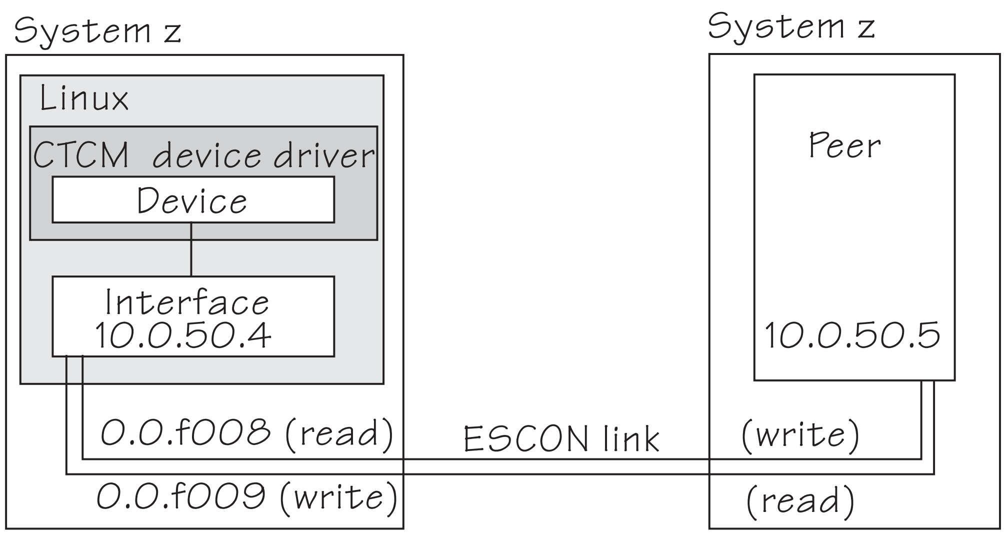


Figure 73. CTC scenario with peer in a different LPAR

Procedure

1. Create a CTCM group device. Issue:

```
# echo 0.0.f008,0.0.f009 > /sys/bus/ccwgroup/drivers/ctcm/group
```

2. Confirm that the device uses CTC FICON or CTC ESCON:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/type
ESCON
```

   In this example, ESCON is used. You would proceed the same for FICON.

3. Select a protocol. The choice depends on the peer.

If the peer is ...	Choose ...
Linux	1
z/OS or OS/390	3
Any other operating system	0

   Assuming that the peer is Linux:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/protocol
```

4. Set the CTCM group device online and find out the assigned interface name:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/online
# ls /sys/devices/ctcm/0.0.f008/net/
ctc0
```

   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:

```
# ip addr add 10.0.50.4 dev ctc0 peer 10.0.50.5
# ip link set dev ctc0 up
```

Connecting Linux on z/VM to another guest of the same z/VM system

A virtual CTCA connection is to be set up between an instance of Linux on z/VM and another guest of the same z/VM system.

Assumptions:

- The guest ID of the peer is “guestp”.
- A separate subnet was obtained from the TCP/IP network administrator. The Linux instance uses IP address 10.0.100.100 and the peer uses IP address 10.0.100.101.

Figure 74 illustrates a CTC setup with a peer in the same z/VM.

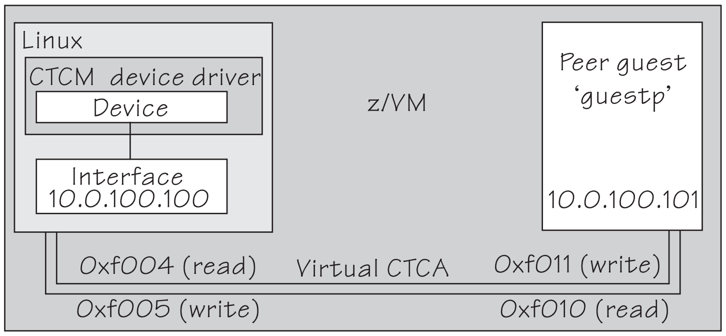


Figure 74. CTC scenario with peer in the same z/VM

Procedure

1. Define two virtual channels to your user ID. The channels can be defined in the z/VM user directory with directory control `SPECIAL` statements, for example:

   ```
   special f004 ctca
   special f005 ctca
   ```

   Alternatively, you can use the CP commands:

   ```
   define ctca as f004
   define ctca as f005
   ```

2. Assure that the peer interface is configured.
3. Connect the virtual channels. Assuming that the read channel on the peer corresponds to device number 0xf010 and the write channel to 0xf011 issue:

   ```
   couple f004 to guestp f011
   couple f005 to guestp f010
   ```

   Be sure that you couple the read channel to the peers write channel and vice versa.
4. From your booted Linux instance, create a CTCM group device. Issue:

   ```
   # echo 0.0.f004,0.0.f005 > /sys/bus/ccwgroup/drivers/ctcm/group
   ```

5. Confirm that the group device is a virtual CTCA device:

```
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/type
CTC/A
```

6. Select a protocol. The choice depends on the peer.

If the peer is ...	Choose ...
Linux	1
z/OS or OS/390	3
Any other operating system	0

   Assuming that the peer is Linux:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/protocol
```

7. Set the CTCM group device online and find out the assigned interface name:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/online
# ls /sys/devices/ctcm/0.0.f004/net/
ctc1
```

   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:

```
# ip addr add 10.0.100.100 dev ctc1 peer 10.0.100.101
# ip link set dev ctc1 up
```

   Be sure that the MTU on both sides of the connection is the same. If necessary, change the default MTU (see "Activating and deactivating a CTC interface" on page 342).
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 341 if the peer is not Linux, see the operating system documentation of the peer.

Chapter 19. NETIUCV device driver

NETIUCV connections are only supported for compatibility with earlier versions. Do not use for new network setups.

The Inter-User Communication Vehicle (IUCV) is a z/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 instances of Linux on z/VM, or to connect an instance of Linux on z/VM 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 Linux on z/VM
- Multiple input paths to Linux on z/VM
- Multiple paths between Linux on z/VM and the same peer z/VM 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

The NETIUCV device driver assigns IUCV interface names and creates IUCV devices in sysfs.

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, see Figure 75.

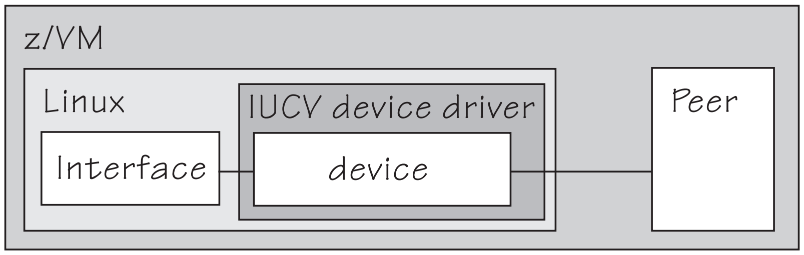


Figure 75. Direct IUCV connection

If your IUCV connection is to a router, the peer can be remote and connected through an external network, see Figure 76 on page 350.

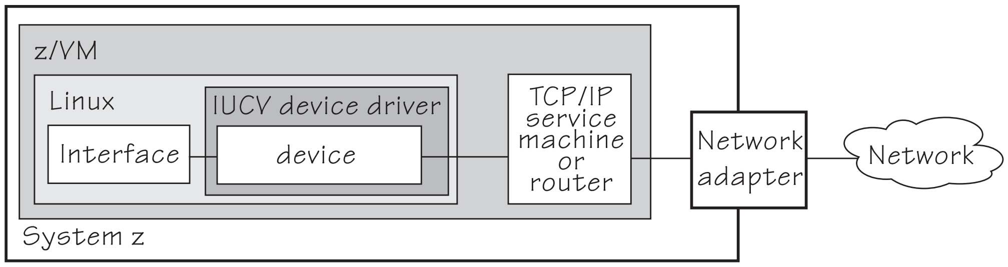


Figure 76. Routed IUCV connection

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*, SC24-6238.

IUCV interfaces and devices

The NETIUCV device driver assigns names to its 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 352) 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.

Building a kernel with the NETIUCV device driver

Control the build options for the NETIUCV device driver through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

This section describes the options you must select in the Linux configuration menu to include the NETIUCV device driver.

Figure 77 on page 351 summarizes the kernel configuration menu options that are relevant to the NETIUCV device driver:

```
Networking support --->                   (common code option CONFIG_NET)
   ...
   Networking options --->
      ...
      IUCV support (S390 - z/VM only)                          (CONFIG_IUCV)
Device Drivers --->
   ...
   Network device support --->            (common code option CONFIG_NETDEVICES)
      ...
      S/390 network device drivers (depends on NETDEVICES && S390) --->
         ...
         IUCV network device support (VM only)                (CONFIG_NETIUCV)*
```

Figure 77. IUCV kernel configuration menu option

CONFIG_IUCV
: This option is required if you want to use IUCV to connect to other z/VM guests. It can be compiled into the kernel or as a separate module, iucv.

CONFIG_NETIUCV
: This option is required if you want to use NETIUCV device driver to connect to other z/VM guests. It can be compiled into the kernel or as a separate module, netiucv.

Setting up the NETIUCV device driver

There are no kernel or module parameters for the NETIUCV device driver, but you might need to load the netiucv module, if it has been compiled as a separate module.

You also need to set up a TCP/IP service machine as a peer for IUCV connections from Linux.

Loading the IUCV modules

If netiucv has been compiled as a separate module, you need to load it 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 IUCV devices

Typical tasks that you need to perform when working with IUCV devices include creating an IUCV device, setting the mazimum buffer size, and activating an interface.

About this task

This section describes typical tasks that you need to perform when working with IUCV devices.
- “Creating an IUCV device”
- “Changing the peer” on page 353
- “Setting the maximum buffer size” on page 354
- “Activating an interface” on page 354
- “Deactivating and removing an interface” on page 355

Creating an IUCV device

Use the `connection` attribute to create an IUCV device.

About this task

To define an IUCV device, write the user ID of the peer z/VM guest and, optionally, a path name to `/sys/bus/iucv/drivers/netiucv/connection`.

Procedure

Issue a command of this form:

```
# echo <peer_id>.<path_name> > /sys/bus/iucv/drivers/netiucv/connection
```

Where:

<peer_id>
: is the user ID of the z/VM guest you want to connect to.

.<path_name>
: identifies an individual path to a peer z/VM guest. This specification is required for setting up multiple paths to the same peer z/VM guest. For setting up a single path to a particular peer z/VM guest, this specification is optional and can be omitted. The path name can be up to 16 characters long. The peer must use the same path name when setting up the peer interface.

The NETIUCV device driver interprets the specification as uppercase.

Results

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 find the index numbers that corresponds to a given user ID, scan the name attributes of all NETIUCV devices. Issue a command of this form:

```
# grep <peer_id> /sys/bus/iucv/drivers/netiucv/*/user
```

Example

To create an IUCV device to connect to a z/VM guest with a z/VM user ID LINUXP issue:

```
# echo linuxp > /sys/bus/iucv/drivers/netiucv/connection
```

To create another two IUCV devices to connect to the same z/VM guest LINUXP issue:

```
# echo linuxp.alt1 > /sys/bus/iucv/drivers/netiucv/connection
# echo linuxp.alt2 > /sys/bus/iucv/drivers/netiucv/connection
```

To find the device and interface that connect to “LINUXP” issue:

```
# grep -Hxi linuxp /sys/bus/iucv/devices/*/user
/sys/bus/iucv/devices/netiucv0/user:LINUXP
/sys/bus/iucv/devices/netiucv1/user:LINUXP.ALT1
/sys/bus/iucv/devices/netiucv2/user:LINUXP.ALT2
```

In the sample output, the device for the first path is `netiucv0` and, therefore, the interface is `iucv0`. For the second and third paths, the devices are `netiucv1` and `netiucv2` and the corresponding interfaces are `iucv1` and `iucv2`.

Changing the peer

You can change the z/VM guest that an interface connects to, or the path name used for the connection, or both.

Before you begin

The interface must not be active when changing the peer.

About this task

Issue a command of this form:

```
# echo <peer_id>.<path_name> > /sys/bus/iucv/drivers/netiucv/netiucv<n>/user
```

where:

<peer_ID>
: is the z/VM user ID of the new communication peer. The value must be a valid z/VM user ID.

.<path_name>
: identifies an individual path to a peer z/VM guest. This specification is required for setting up multiple paths to the same peer z/VM guest. For setting up a single path to a particular peer z/VM guest, this specification is optional and can be omitted. The path name can be up to 16 characters long. The peer must use the same path name when setting up the peer interface.

<n>
: is an index that identifies the IUCV device and the corresponding interface.

The NETIUCV device driver interprets the specification as uppercase.

Examples

- In this example, LINUX22 is set as the new peer z/VM guest for interface `iucv0`.

```
# echo linux22 > /sys/bus/iucv/drivers/netiucv/netiucv0/user
```

- In this example, LINUX22 is set as the new peer z/VM guest for `iucv1` and PATH1 is set as the path name.

```
# echo linux22.path1 > /sys/bus/iucv/drivers/netiucv/netiucv1/user
```

Setting the maximum buffer size

Use the `buffer` attribute to set the maximum buffer size of an IUCV device.

About this task

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 **`ip`** or an equivalent command to activate the interface.

About this task

ip syntax for setting an IP address for an IUCV connection

```
►►──ip address── add <ip_address>── dev <interface>──────────────────────────►

►─┬──────────────────────────┬───────────────────────────────────────────►◄
  └─ peer <peer_ip_address>──┘
```

ip syntax for activating an IUCV interface

```
                                          ┌─ mtu 9216──────────────────┐
►►──ip link set── dev <interface>── up────┤                            ├────────►◄
                                          └─ mtu <max_transfer_unit>───┘
```

where:

<interface>
: is the interface name.

<ip_address>
: is the IP address of your Linux instance.

<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.

For more details, see the **ip** man page.

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.

```
# ip addr add 1.2.3.100 dev iucv1 peer 1.2.3.200
# ip link set dev iucv1 up mtu 32764
```

Deactivating and removing an interface

Use **ip** or an equivalent command to deactivate an interface.

About this task

Issue a command of this form:

```
# ip link set dev <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:

```
# ip link set dev iucv0 down
# echo iucv0 > /sys/bus/iucv/drivers/netiucv/remove
```

Scenario: Setting up an IUCV connection to a TCP/IP service machine

Two Linux instances with z/VM user IDs LNX1 and LNX2 are to be connected through a TCP/IP service machine with user ID VMTCPIP.

About this task

Both Linux instances and the service machine run as guests of the same z/VM system. 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, see Figure 78 on page 357.

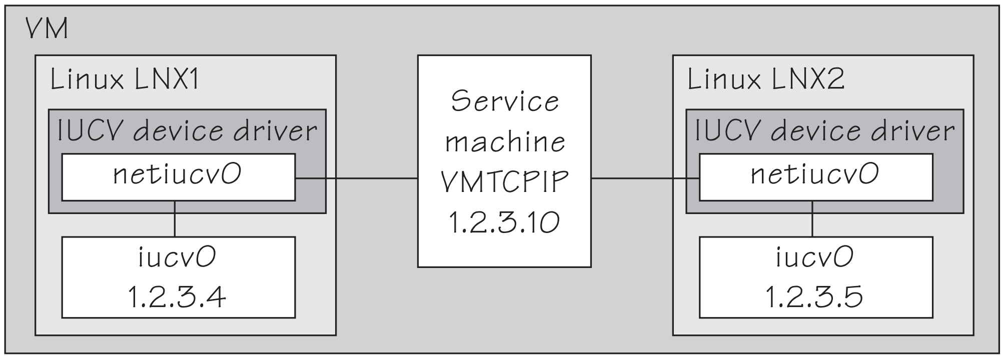


Figure 78. IUCV connection scenario

Setting up the service machine

Setting up the service machine entails editing the `PROFILE TCPIP` file of the service machine.

Procedure

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_name1> IUCV 0 0 <guest_ID1> A
Link <link_name1> IUCV 0  <device_name1>

Device <device_name2> IUCV 0 0 <guest_ID2> A
Link <link_name2> IUCV 0  <device_name2>

...

Start <device_name1>
Start <device_name2>
...
```

   where

   <ip_address1>, <ip_address2>
   are the IP address the Linux instances.

   <link_name1>, <link_name2>, ...
   are variables that associate the link statements with the respective home statements.

   <device_name1>, <device_name2>, ...
   are variables that associate the device statements with the respective link statements and start commands.

   <guest_ID1>, <guest_ID1>, ...
   identify the z/VM guest virtual machines on which the connected Linux instances run.

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 instance.
3. Bring these updates online by using `OBEYFILE` or by recycling `TCPIP` and/or `ROUTED` as needed.

Setting up Linux instance LNX1

Setting up the Linux instance entails setting up the NETIUCV device driver and creating an IUCV interface.

Procedure

Proceed like this to set up the IUCV connection on the Linux instance:

1. Set up the NETIUCV device driver as described in "Setting up the NETIUCV device driver" on page 351.
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.

```
# ip addr add 1.2.3.4 dev iucv0 peer 1.2.3.10
# ip link set dev iucv1 up mtu 32764
```

Chapter 20. AF_IUCV address family support

The AF_IUCV address family provides an addressing mode for communications between applications that run on System z mainframes.

This addressing mode can be used for connections through real HiperSockets and through the z/VM Inter-User Communication Vehicle (IUCV).

Support for AF_IUCV based connections through real HiperSockets requires Completion Queue Support.

HiperSockets devices facilitate connections between applications across LPARs within a System z mainframe. In particular, an application that runs on an instance of Linux on System z can communicate with:

- Itself
- Other applications that run on the same Linux instance
- An application on an instance of Linux on System z in another LPAR

IUCV facilitates connections between applications across z/VM guest virtual machines within a z/VM system. In particular, an application that runs on Linux on z/VM can communicate with:

- Itself
- Other applications that run on the same Linux instance
- Applications running on other instances of Linux on z/VM, within the same z/VM system
- Applications running on a z/VM guest other than Linux, within the same z/VM system
- The z/VM control program (CP)

The AF_IUCV address family supports stream-oriented sockets (SOCK_STREAM) and connection-oriented datagram sockets (SOCK_SEQPACKET). Stream-oriented sockets can fragment data over several packets. Sockets of type SOCK_SEQPACKET always map a particular socket write or read operation to a single packet.

Features

The AF_IUCV address family provides socket connections for HiperSockets and IUCV.

For all instances of Linux on System z, the AF_IUCV address family provides the following features:

- Multiple outgoing socket connections for real HiperSockets
- Multiple incoming socket connections for real HiperSockets

For instances of Linux on z/VM, the AF_IUCV address family also provides the following features:

- Multiple outgoing socket connections for IUCV
- Multiple incoming socket connections for IUCV

- Socket communication with applications that use the CMS AF_IUCV support

Building a kernel with AF_IUCV support

Control the build options for AF_IUCV support through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Figure 79 summarizes the kernel configuration menu options that you must select in the Linux configuration menu to include AF_IUCV support.

```
Networking support --->                                    (common code option CONFIG_NET)
   ...
   Networking options --->
      ...
      IUCV support (S390 - z/VM only)                                         (CONFIG_IUCV)
      AF_IUCV Socket support (S390 - z/VM and HiperSockets transport)         (CONFIG_AFIUCV)*
```

Figure 79. IUCV kernel configuration menu option

CONFIG_IUCV
: This option ensures that the base IUCV-services are usable. It is required if you want to use the z/VM Inter User Communication Vehicle. It can be compiled into the kernel or as a separate module, iucv.

CONFIG_AFIUCV
: This option provides AF_IUCV address family support. It can be compiled into the kernel or as a separate module, af_iucv.

Setting up the AF_IUCV address family support

You must authorize your LPAR or z/VM guest virtual machine and load those components that were compiled as separate modules.

There are no kernel or module parameters for the AF_IUCV address family support.

Setting up HiperSockets devices for AF_IUCV addressing

| In AF_IUCV addressing mode, HiperSockets devices in layer 3 mode are identified through their `hsuid` sysfs attribute.

You set up a HiperSockets device for AF_IUCV by assigning a value to this attribute (see "Configuring a HiperSockets device for AF_IUCV addressing" on page 299).

Setting up your z/VM guest virtual machine for IUCV

You must specify suitable IUCV statements for your z/VM guest virtual machine.

For details and for general IUCV setup information for z/VM guest virtual machines, see *z/VM CP Programming Services*, SC24-6179 and *z/VM CP Planning and Administration*, SC24-6178.

Granting IUCV authorizations

Use the IUCV statement to grant the necessary authorizations.

`IUCV ALLOW`
: allows any other z/VM virtual machine to establish a communication path with this z/VM virtual machine. With this statement, no further authorization is required in the z/VM virtual machine that initiates the communication.

`IUCV ANY`
: allows this z/VM guest virtual machine to establish a communication path with any other z/VM guest virtual machine.

`IUCV` *`<user ID>`*
: allows this z/VM guest virtual machine to establish a communication path to the z/VM guest virtual machine with the z/VM user ID <*user ID*>.

You can specify multiple IUCV statements. To any of these IUCV statements you can append the MSGLIMIT <*limit*> parameter. <*limit*> specifies the maximum number of outstanding messages that are allowed for each connection that is authorized by the statement. If no value is specified for MSGLIMIT, AF_IUCV requests 65 535, which is the maximum that is supported by IUCV.

Setting a connection limit

Use the OPTION statement to limit the number of concurrent connections.

`OPTION MAXCONN` *`<maxno>`*
: <*maxno*> specifies the maximum number of IUCV connections that are allowed for this virtual machine. The default is 64. The maximum is 65 535.

Example

These sample statements allow any z/VM guest virtual machine to connect to your z/VM guest virtual machine with a maximum of 10 000 outstanding messages for each incoming connection. Your z/VM guest virtual machine is permitted to connect to all other z/VM guest virtual machines. The total number of connections for your z/VM guest virtual machine cannot exceed 100.

```
IUCV ALLOW MSGLIMIT 10000
IUCV ANY
OPTION MAXCONN 100
```

Loading the IUCV modules

If af_iucv was compiled as a separate module, you must load it before you can use it.

Use **`modprobe`** to load the AF_IUCV address family support module af_iucv, which loads in addition to the required iucv module.

```
# modprobe af_iucv
```

Addressing AF_IUCV sockets in applications

To use AF_IUCV sockets in applications, you must code a special AF_IUCV sockaddr structure.

Application programmers: This information is intended for programmers who want to use connections that are based on AF_IUCV addressing in their applications.

The primary difference between AF_IUCV sockets and TCP/IP sockets is how communication partners are identified (for example, how they are named). To use the AF_IUCV support in an application, code a sockaddr structure with AF_IUCV as the socket address family and with AF_IUCV address information. For more information, see the af_iucv man page.

Chapter 21. CLAW device driver

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 Linux on System z instance to a communication peer, for example, on a RS/6000® or on a Cisco Channel Interface Processor (CIP).

The CLAW driver supports up to 256 devices.

What you should know about the CLAW device driver

Interface names are assigned to CLAW group devices, which map to subchannels and their corresponding device numbers and device bus-IDs.

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.

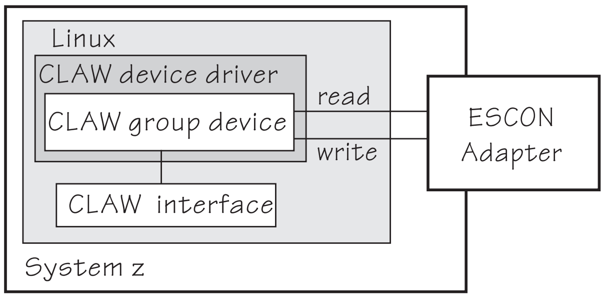


Figure 80. I/O subchannel interface

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 System p 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.

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 366) and 4 kilobyte otherwise.
- The maximum MTU you can set is 4096 byte.

Kernel builders: If you are a kernel builder, you can increase the maximum MTU above 4096 byte by changing the CLAW device driver code and recompiling. Be aware that recompiling the kernel is likely to affect any existing service contracts you may have for your kernel.

Building a kernel with the CLAW device driver

You must select options in the Linux configuration menu to include the CLAW device driver.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

You must select the CONFIG_CLAW option if you want to use CLAW connections (see Figure 81).

```
Device Drivers --->
   ...
    Network device support --->          (common code option CONFIG_NETDEVICES)
      ...
      S/390 network device drivers (depends on NETDEVICES && S390) --->
         ...
         CLAW device support                                 (CONFIG_CLAW)
```

Figure 81. CLAW kernel configuration menu option

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

Setting up the CLAW device driver

If you compiled the CLAW component as a separate module, you must load it before you can work with CLAW group devices.

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

Load the claw module with the **modprobe** command to ensure that any other required modules are loaded:

```
# modprobe claw
```

Working with CLAW devices

Typical tasks that you must perform when you work with CLAW devices include creating a CLAW group device, setting the connection type, and activating a group device.

- “Creating a CLAW group device”
- “Setting the host and adapter name” on page 366
- “Setting the connection type” on page 366
- “Setting the number of read and write buffers” on page 367
- “Setting a CLAW device online or offline” on page 368
- “Activating a CLAW group device” on page 369

Creating a CLAW group device

Use the `group` attribute to create a CLAW group device.

Before you begin

You must know the device bus-IDs that correspond to the local read and write subchannel of your CLAW connection as defined in your IOCDS.

Procedure

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
```

Results

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.

Example

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/claw/0.0.2d00`

Setting the host and adapter name

Use the `host_name` and `adapter_name` attributes to set the host and adapter for a CLAW group device.

About this task

Host and adapter names identify the communication peers to one another. The local host name must match the remote adapter name and vise 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.

Procedure

- 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

Use the `api_type` attribute to set the connection type for a CLAW group device.

About this task

The connection type determines the packing method that is used for outgoing packets. The connection type must match the connection type on the connected System p 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.

Procedure

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.

TCPIP 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

Use the `read_buffer` and `write_buffer` attributes to set the number of buffers for a CLAW group device.

About this task

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.

Procedure

- 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 366):
  - For connection type PACKED, you can allocate 2 - 64 buffers of 32 KB.
  - For the other connection types, you can allocate 2 - 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 device online or offline

Use the `online` device group attribute to set a CLAW group device online or offline.

Procedure

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. You must know the interface name to access the CLAW group device. To determine the assigned interface name, use the **znetconf -c** command. For each online interface, the interface name is shown in the Name column. Alternatively, to determine the assigned interface name issue a command of the form:

```
# ls /sys/devices/claw/<device_bus_id>/net/
```

For each online interface, there is a symbolic link of the form `/sys/class/net/<interface_name>/device` in sysfs. You can confirm that you found the correct interface name by reading the link.

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
```

To determine the interface name issue:

```
# znetconf -c
Device IDs                 Type    Card Type      CHPID Drv. Name       State
--------------------------------------------------------------------------------
0.0.2d00,0.0.2d01          3088/61                      claw claw0      online
```

or

```
# ls /sys/devices/claw/0.0.d200/net/
claw0
...
```

The interface name that was assigned to the CLAW group device in the example is claw0. To confirm that this name is the correct one for the group device, issue:

```
# readlink /sys/class/net/claw0/device
../../../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

Use **ip** or an equivalent command to activate a CLAW group device.

Procedure

You can activate a CLAW group device with **ip** or an equivalent command. See "MTU size" on page 364 for information on possible MTU settings.

Issue a command like the following example:

```
# ip addr add 10.22.34.5 dev claw0 peer 10.22.34.6
```

Chapter 22. RDMA over Converged Ethernet

Linux on System z supports RDMA over Converged Ethernet (RoCE) in the form of 10GbE RoCE Express features.

A 10GbE RoCE Express feature physically consists of a Mellanox ConnectX-3 EN adapter. This adapter is a two-port Ethernet adapter.

The RoCE support requires PCI Express support, see "PCI Express support" on page 21.

Building a kernel with RoCE

Control the build options for RoCE through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Figure 82 summarizes the kernel configuration menu options that are relevant to the RoCE support.

```
Device Drivers --->
    ...
    Infiniband support --->                                  (CONFIG_INFINIBAND)
      ...
    InfiniBand userspace access (verbs and CM)    (CONFIG_INFINIBAND_USER_ACCESS)
    Mellanox ConnectX HCA support --->                  (CONFIG_MLX4_INFINIBAND)
    ...
    Network device support --->                              (CONFIG_NETDEVICES)
      ...
      Ethernet driver support                                  (CONFIG_ETHERNET)
         ...
         Mellanox devices                           (CONFIG_NET_VENDOR_MELLANOX)
         └─ Mellanox Technologies 1/10/40Gbit Ethernet support   (CONFIG_MLX4_EN)
            └─ VXLAN offloads Support                      (CONFIG_MLX4_EN_VXLAN)

Networking support --->                                             (CONFIG_NET)
 ....
     Networking options
       ....
       The RDS Protocol                                             (CONFIG_RDS)
       ├─ RDS over Infiniband and iWARP                        (CONFIG_RDS_RDMA)
       ├─ RDS over TCP                                          (CONFIG_RDS_TCP)
       └─ RDS debugging messages                              (CONFIG_RDS_DEBUG)
```

Figure 82. RoCE kernel configuration menu options

All the options are common code. Deselect the common code option CONFIG_USB_SUPPORT.

Working with the RoCE support

Because the 10 GBE RoCE Express feature hardware physically consists of a Mellanox adapter, you must ensure that the following prerequisites are fulfilled before you can work with it.

Procedure

1. Ensure that PCIe support is enabled and the PCI card is active on your system. See “Setting up the PCIe support” on page 22 and “Using PCIe hotplug” on page 22.
2. Use the appropriate Mellanox device driver:
   - If you want to use TCP/IP, you need the `mlx4_en` module. If it is not compiled into kernel or already loaded, load it using for example, **`modprobe`**.
   - If you also want to use RDMA with InfiniBand (that is, using reliable datagram sockets, RDS), you need the `mlx4_ib` module. If it is not compiled into kernel or already loaded, load it using for example, **`modprobe`**. To use RDS, you also need the `rds` module and the `rds_rdma` module, see https://www.openfabrics.org/index.php/ofed-for-linux-ofed-for-windows/installing-ofed.html.
3. Activate the network interface. You need to know the network interface name, which you can find under `/sys/bus/pci/drivers/mlx4_core/<`*`PCI slot`*`>/net/<`*`interface`*`>`. Use the **`ip`** command or equivalent to activate the interface. See the `dev_port` sysfs attribute of the interface name to ensure that you are working with the correct port.

What to do next

For further information about Mellanox, see:

- http://www.mellanox.com/page/products_dyn?product_family=27&mtag=linux_driver
- http://www.mellanox.com/page/products_dyn?product_family=79&mtag=roce

Enabling debugging

The Mellanox mlx4 device driver can be configured with a kernel configuration option for debugging.

About this task

Debugging for the Mellanox mlx4 device driver is only available if the device driver is compiled with the kernel-configuration menu option CONFIG_MLX4_DEBUG.

Procedure

1. Check that the device driver has the CONFIG_MLX4_DEBUG option enabled.
2. Load the mlx4 modules with the sysfs parameter `debug_level=1` to write debug messages to the syslog. Check the value of the `debug_level` parameter . If the parameter is set to 0, you can set it to 1 with the following command:

   ```
   echo 1 > /sys/module/mlx4_core/parameters/debug_level
   ```

Part 5. System resources

These device drivers and features help you to manage the resources of your real or virtual hardware.

Newest version

You can find the newest version of this publication at www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Restrictions

For prerequisites and restrictions see
www.ibm.com/developerworks/linux/linux390/development_technical.html
www.ibm.com/developerworks/linux/linux390/development_restrictions.html

Chapter 23. Managing CPUs

You can read CPU capability, activate standby CPUs, and examine the CPU topology by using the CPU attributes in sysfs.

Some attributes that govern CPUs are available in sysfs under:

```
/sys/devices/system/cpu/cpu<N>
```

where *<N>* is the number of the CPU.

In addition to the sysfs interface described in this information, you can also use **lscpu** and **chcpu** to manage CPUs. These commands are part of the util-linux package. For details, see the man pages for these commands.

CPU capability change

When the CPUs of a mainframe heat or cool, the Linux kernel generates a uevent for all affected online CPUs.

You can read the CPU capability from the Capability and, if present, Secondary Capability fields in /proc/sysinfo.

The capability values are unsigned integers as defined in the system information block (SYSIB) 1.2.2 (see *z/Architecture Principles of Operation*, SA22-7832). A smaller value indicates a proportionally greater CPU capacity. Beyond that, there is no formal description of the algorithm that is used to generate this value. The value is used as an indication of the capability of the CPU relative to the capability of other CPU models.

Activating standby CPUs and deactivating operating CPUs

A CPU on an LPAR can be in a configured, standby, or reserved state. You can change the state of standby CPUs to configured state and vice versa.

Before you begin

- To put a CPU into standby state, the underlying hypervisor must support this operation.
- Daemon processes like **cpuplugd** can change the state of any CPU at any time. Such changes can interfere with manual changes.

About this task

Under Linux, on IPL only CPUs that are in a configured state are brought online and used. The kernel operates only with configured CPUs.

Reserved CPUs cannot be used without manual intervention and therefore are not recognized.

To configure or deconfigure a CPU, its physical address must be known. Because the sysfs interface is used to configure a CPU by its sysfs entry, a static mapping of physical to logical CPU numbers is needed. 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
```

Procedure

- 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 by using the `rescan` attribute. To rescan, write any string to the `rescan` attribute, for example:

     ```
     echo 1 > /sys/devices/system/cpu/rescan
     ```

     When new CPUs are found, new sysfs entries are created. The new CPUs 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
     ```

     When Linux is running in an LPAR, setting a CPU offline can result in the LPAR status "Exceptions" in the HMC or SE. With one or more CPUs offline, this status does not necessarily indicate a problem.
  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.

About this task

Use this information, for example, to optimize the Linux scheduler, which bases its decisions on which process gets scheduled to which CPU. Depending on the workload, this optimization might increase cache hits and therefore overall performance.

Note: By default, CPU topology support is enabled in the Linux kernel. If it is not suitable for your workload, disable the support by specifying the kernel parameter `topology=off` in your parmfile or `zipl.conf`. See “Specifying kernel parameters” on page 25 for information about specifying kernel parameters.

The common code attributes `core_siblings` and `core_id` are visible for all online CPUs:

```
/sys/devices/system/cpu/cpu<N>/topology/core_siblings
/sys/devices/system/cpu/cpu<N>/topology/core_id
```

The `core_siblings` attribute contains a CPU mask that indicates which CPUs, including the current one, are close to each other. If a machine reconfiguration causes the CPU topology to change, `change` uevents are created for each online CPU. All CPUs that have the same `core_siblings` CPU mask have the same `core_id`.

If the kernel also supports standby CPU activation and deactivation (see “Activating standby CPUs and deactivating operating CPUs” on page 375), the `core_siblings` CPU mask also contains the CPUs that are in a configured, but offline state. Updating the mask after a reconfiguration might take some time.

With zEnterprise, the book topology level was added above the core level. The `book_siblings` and `book_id` attributes describe which CPUs on different cores belong to the same book.

```
# cat /sys/devices/system/cpu/cpu1/topology/book_siblings
00000000,0000001f
# cat /sys/devices/system/cpu/cpu1/topology/book_id
2
```

The CPU masks contained in the `book_siblings` attribute are always a superset of the `core_siblings` attribute. All CPUs that have the same `book_siblings` CPU mask have the same `book_id`. If there are several books present in a configuration, the `core_ids` are only unique per book.

CPU polarization

You can optimize the operation of a vertical SMP environment by adjusting the SMP factor based on the workload demands.

About this task

During peak workloads, the operating system might operate on a large n-way, with all CPUs busy, whereas at other times it might fall back to a single processor. This limits the performance effects of context switches, TLB flushes, cache poisoning, and dispatcher workload balancing and the like, by delivering better processor affinity for particular workloads.

Horizontal CPU polarization means that the underlying hypervisor dispatches each virtual CPU of all z/VM guest virtual machines for the same amount of time.

If vertical CPU polarization is active, the hypervisor dispatches certain CPUs for a longer time than others for maximum performance. For example, if a guest has three virtual CPUs, each of them with a share of 33%, then in case of vertical CPU polarization all of the processing time would be combined to a single CPU, which would run all the time, while the other two CPUs would get nearly no CPU time.

There are three types of vertical CPUs: high, medium, and low. Low CPUs hardly get any real CPU time, while high CPUs get a full real CPU. Medium CPUs get something in between.

Note: Running a system with different types of vertical CPUs can result in significant performance regressions. If possible, use only one type of vertical CPUs. Set all other CPUs offline and deconfigure them.

Procedure

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`

Results

When switching polarization, the polarization attribute might contain the value `unknown` until the configuration change is done and the kernel has established the new polarization of each CPU.

Chapter 24. Managing hotplug memory

You can dynamically increase or decrease the memory for your running Linux instance.

To make memory available as hotplug memory, you must define it to your LPAR or z/VM. Hotplug memory is supported by z/VM 5.4 with the PTF for APAR VM64524 and by later z/VM versions.

For more information about memory hotplug, see `Documentation/memory-hotplug.txt` in the Linux source tree.

What you should know about memory hotplug

Hotplug memory is represented in sysfs. After rebooting Linux, all hotplug memory is offline.

How memory is represented in sysfs

Both the core memory of a Linux instance and the available hotplug memory are represented by directories in sysfs.

The memory with which Linux is started is the *core memory*. On the running Linux system, additional memory can be added as *hotplug memory*. The Linux kernel requires core memory to allocate its own data structures.

In sysfs, both the core memory of a Linux instance and the available hotplug memory are represented in form of memory blocks of equal size. Each block is represented as a directory of the form `/sys/devices/system/memory/memory<n>`, where <*n*> is an integer. You can find out the block size by reading the `/sys/devices/system/memory/block_size_bytes` attribute.

In the naming scheme, the memory blocks with the lowest address ranges are assigned the lowest integer numbers. The core memory always begins with memory0. The hotplug memory blocks follow the core memory blocks.

You can calculate where the hotplug memory begins. To find the number of core memory blocks, divide the base memory by the block size.

Example:

- With a core memory of 512 MB and a block size of 128 MB, the core memory is represented by four blocks, memory0 through memory3. Therefore, first hotplug memory block on this Linux instance is memory4.
- Another Linux instance with a core memory of 1024 MB and access to the same hotplug memory, represents this first hotplug memory block as memory8.

The hotplug memory is available to all operating system instances within the z/VM system or LPAR to which it was defined. The `state` sysfs attribute of a memory block indicates whether the block is in use by your own Linux system. The `state` attribute does not indicate whether a block is in use by another operating system instance. Attempts to add memory blocks that are already in use fail.

Hotplug memory and reboot

The original core memory is preserved as core memory and hotplug memory is freed when rebooting a Linux instance.

When you perform an IPL after shutting down Linux, always use **ipl clear** to preserve the original memory configuration.

Building a kernel with memory hotplug support

Control the build options for the memory hotplug support through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Figure 83 summarizes the common source options that you must select in the Linux configuration menu to include the memory hotplug functions.

```
Memory setup -->
   ...
   Allow for memory hot-add                      (CONFIG_MEMORY_HOTPLUG)
   └─ Allow for memory hot remove                (CONFIG_MEMORY_HOTREMOVE)
   Page migration                                (CONFIG_MIGRATION)
```

Figure 83. Kernel configuration menu options

CONFIG_MEMORY_HOTPLUG
is required for dynamically attaching memory to the Linux instance.

CONFIG_MEMORY_HOTREMOVE and CONFIG_MIGRATION
are required for dynamically detaching memory from the Linux instance.

Setting up hotplug memory

Before you can use hotplug memory on your Linux instance, you must define this memory as hotplug memory on your physical or virtual hardware.

Defining hotplug memory to an LPAR

You use the Hardware Management Console (HMC) to define hotplug memory as *reserved storage* on an LPAR.

For information about defining reserved storage for your LPAR, see the *Processor Resource/Systems Manager Planning Guide*, SB10-7041 for your mainframe.

Defining hotplug memory to z/VM

In z/VM, you define hotplug memory as *standby storage*.

There is also *reserved storage* in z/VM, but other than reserved memory defined for an LPAR, reserved storage that is defined in z/VM is not available as hotplug memory.

For information about defining standby memory for z/VM guests see the "DEFINE STORAGE" section in *z/VM CP Commands and Utilities Reference*, SC24-6175.

Performing memory management tasks

Typical memory management tasks include finding out the memory block size, adding memory, and removing memory.

- "Finding out the memory block size"
- "Listing the available memory blocks"
- "Adding memory" on page 382
- "Removing memory" on page 383

Finding out the memory block size

On a System z mainframe, memory is provided to Linux as memory blocks of equal size.

Procedure

- Use the **lsmem** command to find out the size of your memory blocks (see "lsmem - Show online status information about memory blocks" on page 625).

  Example:

```
# lsmem
Address range                          Size (MB)  State    Removable  Device
===============================================================================
0x0000000000000000-0x000000000fffffff        256  online   no         0
0x0000000010000000-0x000000002fffffff        512  online   yes        1-2
0x0000000030000000-0x000000003fffffff        256  online   no         3
0x0000000040000000-0x000000006fffffff        768  online   yes        4-6
0x0000000070000000-0x00000000ffffffff       2304  offline  -          7-15

Memory device size  : 256 MB
Memory block size   : 256 MB
Total online memory : 1792 MB
Total offline memory: 2304 MB
```

  In the example, the block size is 256 MB.
- Alternatively, you can read `/sys/devices/system/memory/block_size_bytes`. This sysfs attribute contains the block size in byte in hexadecimal notation.

  Example:

```
# cat /sys/devices/system/memory/block_size_bytes
10000000
```

  This hexadecimal value corresponds to 256 MB.

Listing the available memory blocks

List the available memory to find out how much memory is available and which memory blocks are online.

Procedure

- Use the **lsmem** command to list your memory blocks.

  Example:

```
# lsmem -a
Address range                          Size (MB)  State    Removable  Device
===============================================================================
0x0000000000000000-0x000000000fffffff        256  online   no         0
0x0000000010000000-0x000000001fffffff        256  online   no         1
0x0000000020000000-0x000000002fffffff        256  online   no         2
0x0000000030000000-0x000000003fffffff        256  online   yes        3
0x0000000040000000-0x000000004fffffff        256  online   yes        4
0x0000000050000000-0x000000005fffffff        256  offline  -          5
0x0000000060000000-0x000000006fffffff        256  offline  -          6
0x0000000070000000-0x000000007fffffff        256  offline  -          7

Memory device size  : 256 MB
Memory block size   : 256 MB
Total online memory : 1280 MB
Total offline memory: 786 MB
```

For more information about the **lsmem** command, see "lsmem - Show online status information about memory blocks" on page 625.

- Alternatively, you can list the available memory blocks by listing the contents of /sys/devices/system/memory. Read the state attributes of each memory block to find out whether it is online or offline.

  Example: The following command results in an overview for all available memory blocks.

```
# grep -r --include="state" "line" /sys/devices/system/memory/
/sys/devices/system/memory/memory0/state:online
/sys/devices/system/memory/memory1/state:online
/sys/devices/system/memory/memory2/state:online
/sys/devices/system/memory/memory3/state:online
/sys/devices/system/memory/memory4/state:online
/sys/devices/system/memory/memory5/state:offline
/sys/devices/system/memory/memory6/state:offline
/sys/devices/system/memory/memory7/state:offline
```

Note

Online blocks are in use by your Linux instance. An offline block can be free to be added to your Linux instance but it might also be in use by another Linux instance.

Adding memory

You can add memory to your Linux instance by setting unused memory blocks online.

Suspend and resume:

Do not add hotplug memory if you intend to suspend the Linux instance before the next IPL. Any changes to the original memory configuration prevent suspension, even if you restore the original memory configuration by removing memory blocks that were added. See Chapter 7, "Suspending and resuming Linux," on page 111 for more information about suspending and resuming Linux.

Procedure

- Use the **chmem** command with the **-e** parameter to set memory online. You can specify the amount of memory you want to add with the command without specifying particular memory blocks. If there are enough eligible memory blocks

to satisfy your request, the tool finds them for you and sets the most suitable blocks online. For information about the **chmem** command, see "chmem - Set memory online or offline" on page 548.
- Alternatively, you can write `online` to the sysfs `state` attribute of an unused memory block. Issue a command of the form:

```
# echo online > /sys/devices/system/memory/memory<n>/state
```

where <*n*> is an integer that identifies the memory unit.

Results

Adding the memory block fails if the memory block is already in use. The `state` attribute changes to `online` when the memory block has been added successfully.

Removing memory

You can remove memory from your Linux instance by setting memory blocks offline.

About this task

Avoid removing core memory. The Linux kernel requires core memory to allocate its own data structures.

Procedure

- Use the **chmem** command with the **-d** parameter to set memory offline. You can specify the amount of memory you want to remove with the command without specifying particular memory blocks. The tool finds eligible memory blocks for you and sets the most suitable blocks offline. For information about the **chmem** command, see "chmem - Set memory online or offline" on page 548.
- Alternatively, you can write `offline` to the sysfs `state` attribute of an unused memory block. Issue a command of the form:

```
# echo offline > /sys/devices/system/memory/memory<n>/state
```

where <*n*> is an integer that identifies the memory unit.

Results

The hotplug memory functions first relocate memory pages to free the memory block and then remove it. The `state` attribute changes to offline when the memory block has been removed successfully.

The memory block is not removed if it cannot be freed completely.

Chapter 25. Large page support

Large page support entails support for the Linux hugetlbfs file system.

This virtual file system is backed by larger memory pages than the usual 4 K pages; for System z the hardware page size is 1 MB.

Applications that use large page memory save a considerable amount of page table memory. Another benefit from the support might be an acceleration in the address translation and overall memory access speed.

The Linux kernel also supports transparent hugepages. For more information, see `Documentation/vm/transhuge.txt` in the Linux source tree.

Building a kernel with large page support

Control the build options for the large page support through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Figure 84 shows the kernel configuration menu options that are relevant to large page support.

Note: The configuration options for large page support are available only for 64-bit kernels.

```
Memory setup --->
   ...
   Transparent large page support                  (CONFIG_TRANSPARENT_HUGEPAGE)
   ...
File systems --->
   ...
   Pseudo filesystems --->
      ...
      HugeTLB file system support                     (CONFIG_HUGETLBFS)
```

Figure 84. Large page support kernel configuration menu options

CONFIG_TRANSPARENT_HUGEPAGE
: Allows transparent access to large page memory for any application that is using anonymous memory mappings. For details, see `Documentation/vm/transhuge.txt` in the Linux source tree.

CONFIG_HUGETLBFS
: The hugetlbfs is a file system backing for HugeTLB pages, based on ramfs. For details, see `Documentation/vm/hugetlbpage.txt` in the Linux source tree.

Setting up hugetlbfs large page support

You configure hugetlbfs large page support by adding the `hugepages=` parameter to the kernel parameter line.

Large page support kernel parameter syntax

```
►►──hugepages=<number>──────────────────────────────────────────────►◄
```

where:

`number`
is the number of large pages to be allocated at boot time.

Note: If you specify more pages than available, Linux reserves as many as possible. As a likely result, too few general pages remain for the boot process, and your system stops with an out-of-memory error.

Large pages and hotplug memory

Hotplug memory that is added to a running Linux instance is movable and can be allocated to movable resources only.

By default, large pages are not movable and cannot be allocated from movable memory. You can enable allocation from movable memory with the sysctl setting `hugepages_treat_as_movable`.

To enable allocation of large pages from movable hotplug memory, issue:

```
# echo 1 > /proc/sys/vm/hugepages_treat_as_movable
```

Although this setting makes large pages eligible for allocation through movable memory, it does not make large pages movable. As a result, the allocated hotplug memory cannot be set offline until all large pages are released from that memory.

To disable allocation of large pages from movable hotplug memory, issue:

```
# echo 0 > /proc/sys/vm/hugepages_treat_as_movable
```

Working with hugetlbfs large page support

Typical tasks for working with hugetlbfs large page support include reading the current number of large pages, changing the number of large pages, and display information about available large pages.

About this task

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.

You can configure Oracle Database 11*g* Release 2 to use large page memory. For more information, see the Redbooks® publication *Experiences with Oracle 11gR2 on Linux on System z*, SG24-8104.

Depending on your version of Java, you might require specific options to make a Java™ program use the large page feature. For IBM SDK, Java Technology Edition 7, specify the **`-Xlp`** option. If you use the SysV shared memory interface, which includes **`java -Xlp`**, you must adjust the shared memory allocation limits to match the workload requirements. Use the following sysctl attributes:

`/proc/sys/kernel/shmall`
Defines the global maximum amount of shared memory for all processes, specified in number of 4 KB pages.

`/proc/sys/kernel/shmmax`
Defines the maximum amount of shared memory per process, specified in number of Bytes.

For example, the following commands would set both limits to 20 GB:

```
# echo 5242880 > /proc/sys/kernel/shmall
# echo 21474836480 > /proc/sys/kernel/shmmax
```

Procedure

- Specify the `hugepages=` kernel parameter with the number of 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 runtime, write to procfs:

  ```
  # echo 12 > /proc/sys/vm/nr_hugepages
  ```

  If there is not enough contiguous memory available to fulfill the request, the maximum possible number of large pages are reserved.
- To obtain information about the number of large pages currently available and the large page size, issue:

  ```
  # cat /proc/meminfo

  ...
  HugePages_Total: 20
  HugePages_Free: 14
  HugePages_Rsvd: 0
  HugePages_Surp: 0
  ...
  Hugepagesize: 1024 KB
  ...
  ```

- To see whether hardware large page support is enabled, issue this command.

```
# grep edat /proc/cpuinfo
features        : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te
```

An output line that lists `edat` as a feature indicates large page support.

Chapter 26. S/390 hypervisor file system

The S/390 hypervisor file system (hypfs) provides a mechanism to access LPAR and z/VM hypervisor data.

Building a kernel with the S/390 hypervisor file system

There is an option in the kernel configuration menu that you must select to compile a kernel with the S/390 hypervisor file system.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

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

```
Virtualization --->
   ...
   s390 hypervisor file system support        (CONFIG_S390_HYPFS_FS)
```

Directory structure

When the hypfs file system is mounted, the accounting information is retrieved and a file system tree is created. The tree contains a full set of attribute files with the hypervisor information.

By convention, the mount point for the hypervisor file system is `/sys/hypervisor/s390`.

LPAR directories and attributes

There are hypfs directories and attributes with hypervisor information for Linux in LPAR mode.

Figure 85 on page 390 illustrates the file system tree that is created for LPAR.

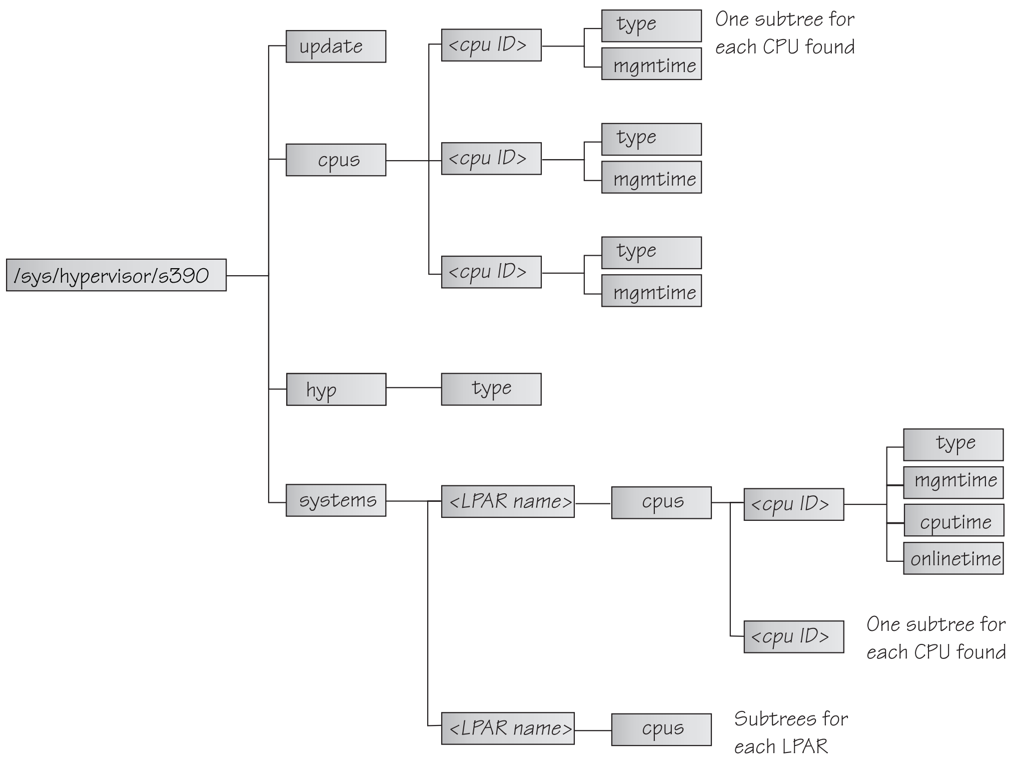


Figure 85. The hypervisor file system for LPAR

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. Generally, it is advantageous for applications to tolerate missing attributes or new attributes that are added to the file system. To check the content of the files, you can use tools such as **`cat`** or **`less`**.

z/VM directories and attributes

There are hypfs directories and attributes with hypervisor information for Linux on z/VM.

`update` Write-only file to trigger an update of all attributes.

`cpus/` Directory for all physical CPUs.

`cpus/count`
: Total current CPUs.

`hyp/` Directory for hypervisor information.

`hyp/type`
: Type of hypervisor (z/VM hypervisor).

`systems/`
: Directory for all z/VM guest virtual machines.

`systems/`*`<guest name>`***`/`**
: Directory for one guest virtual machine.

`systems/`*`<guest name>`***`/onlinetime_us`**
: Time in microseconds that the guest virtual machine has been logged on.

`systems/`*`<guest name>`***`/cpus/`**
: Directory for the virtual CPUs for one guest virtual machine.

  `capped` Flag that shows whether CPU capping is on for the guest virtual machine (0 = off, 1 = soft, 2 = hard).

  `count` Total current virtual CPUs in the guest virtual machine.

  `cputime_us`
  : Number of microseconds where the guest virtual CPU was running on a physical CPU.

  `dedicated`
  : Flag that shows if the guest virtual machine has at least one dedicated CPU (0 = no, 1 = yes).

  `weight_cur`
  : Current share of guest virtual machine (1-10000); 0 for ABSOLUTE SHARE guests.

weight_max
Maximum share of guest virtual machine (1-10000); 0 for ABSOLUTE SHARE guests.

weight_min
Number of operating CPUs. Do not be confused by the attribute name, which suggests a different meaning.

systems/<*guest name*>**/samples/**
Directory for sample information for one guest virtual machine.

cpu_delay
Number of CPU delay samples that are attributed to the guest virtual machine.

cpu_using
Number of CPU using samples attributed to the guest virtual machine.

idle Number of idle samples attributed to the guest virtual machine.

mem_delay
Number of memory delay samples that are attributed to the guest virtual machine.

other Number of other samples attributed to the guest virtual machine.

total Number of total samples attributed to the guest virtual machine.

systems/<*guest name*>**/mem/**
Directory for memory information for one guest virtual machine.

max_KiB
Maximum memory in KiB (1024 bytes).

min_KiB
Minimum memory in KiB (1024 bytes).

share_KiB
Guest estimated core working set size in KiB (1024 bytes).

used_KiB
Resident memory in KiB (1024 bytes).

To check the content of the files, you can use tools such as **cat** or **less**.

Setting up the S/390 hypervisor file system

To use the file system, it must be mounted. You can mount the file system with the mount command or with an entry in /etc/fstab.

To mount the file system manually, issue the following command:

```
# mount none -t s390_hypfs <mount point>
```

where <*mount point*> is where you want the file system mounted. Preferably, use /sys/hypervisor/s390.

To mount hypfs by using /etc/fstab, add the following line:

```
none <mount point> s390_hypfs defaults 0 0
```

If your z/VM system does not support DIAG 2fc, the s390_hypfs is not activated and it is not possible to mount the file system. Instead, an error message like this is issued:

```
mount: unknown filesystem type 's390_hypfs'
```

To get data for all z/VM guests, privilege class B is required for the guest where hypfs is mounted. For non-class B guests, data is provided only for the local guest.

To get data for all LPARs, select the **Global performance data control** check box in the HMC or SE security menu of the LPAR activation profile. Otherwise, data is provided only for the local LPAR.

Working with the S/390 hypervisor file system

Typical tasks that you must perform when working with the S/390 hypervisor file system include defining access permissions and updating hypfs information.

- "Defining access permissions"
- "Updating hypfs information"

Defining access permissions

The root user usually has access to the hypfs file system. It is possible to explicitly define access permissions.

About this task

If no mount options are specified, the files and directories of the file system get the uid and gid of the user who mounted the file system (usually root). You can explicitly define uid and gid using the mount options `uid=<number>` and `gid=<number>`.

Example

You can define uid=1000 and gid=2000 with the following mount command:

```
# mount none -t s390_hypfs -o "uid=1000,gid=2000" <mount point>
```

Alternatively, you can add the following line to the `/etc/fstab` file:

```
none <mount point> s390_hypfs uid=1000,gid=2000 0 0
```

The first mount defines uid and gid. Subsequent mounts automatically have the same uid and gid setting as the first one.

The permissions for directories and files are as follows:

- Update file: 0220 (`--w--w----`)
- Regular files: 0440 (`-r--r-----`)
- Directories: 0550 (`dr-xr-x---`)

Updating hypfs information

You trigger the update process by writing something into the `update` attribute at the top-level hypfs directory.

Procedure

With hypfs mounted at /sys/hypervisor/s390, you can trigger the update process by issuing the following command:

```
# echo 1 > /sys/hypervisor/s390/update
```

During the update, the entire directory structure is deleted and rebuilt. If a file was open before the update, subsequent reads return the old data until the file is opened again. Within 1 second only one update can be done. If multiple updates are triggered within a second, only the first update is performed and subsequent write system calls return -1 and errno is set to EBUSY.
Applications can use the following procedure to ensure consistent data:

1. Read the modification time through stat(2) from the update attribute.
2. If the data is too old, write to the update attribute and start again with step 1.
3. Read data from the file system.
4. Read the modification time of the update attribute again and compare it with first timestamp. If the timestamps do not match, return to step 2.

Chapter 27. CHSC subchannel device driver

The CHSC subchannel device driver provides a means for user space programs to obtain information about the machine setup and to perform asynchronous channel subsystem calls that modify the machine setup.

From a designated LPAR with special privileges, you can issue asynchronous channel subsystem calls. With these calls, you can dynamically change the I/O configuration of the machine.

The CHSC subchannel device driver implements an interface to issue synchronous and asynchronous CHSC calls. Control blocks for the calls must be built in user space.

What you should know about the CHSC subchannel device driver

The CHSC subchannel device driver binds to any CHSC subchannels that are found on the css bus. No special attributes are created; and, unlike other subchannel types, no second-level device is registered.

CHSC subchannels are automatically enabled when the CHSC subchannel device driver binds to them. There is no user interface for enabling or disabling CHSC subchannels.

Building a kernel with the CHSC subchannel device driver

Use a kernel configuration option to build a kernel with the CHSC subchannel device driver.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Select the CONFIG_CHSC_SCH kernel configuration option to include the CHSC subchannel device driver (see Figure 86).

```
I/O subsystem --->
   ...
   Support for CHSC subchannels                           (CONFIG_CHSC_SCH)
```

Figure 86. Kernel configuration menu option for the CHSC subchannel device driver

The CHSC subchannel device driver can be compiled into the kernel or as a separate module, chsc_sch.

Setting up the CHSC subchannel device driver

If you have compiled the CHSC subchannel device driver as a separate module, you must load it before you can work with it.

Load the chsc_sch module with the modprobe command to ensure that any other required modules are loaded in the correct order:

```
# modprobe chsc_sch
```

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

Assuring that a device node exists

Normally, your distribution creates a device node for you. Alternatively, use the **mknod** command to create one.

The CHSC subchannel device driver provides a misc character device: /dev/chsc. If your distribution does not create the device node for you (for example, with udev), you must create a node.

If your distribution provides the CHSC subchannel device driver as a separate module, be sure to load the module before you check for the node.

To check whether there is already a node issue this command:

```
# find /dev -name chsc
```

If there is no node, use major number 10 and a dynamic minor number to create one. To find the minor number, issue:

```
# grep chsc /proc/misc
 57 chsc
```

Issue:

```
# mknod /dev/chsc c 10 57
```

An application can issue ioctls on this device to obtain information about the current I/O configuration and to dynamically change the I/O configuration. See “External programming interfaces” for a summary of ioctls.

External programming interfaces

Applications can use the CHSC subchannel device driver through ioctls.

Programmers: This information is intended for programmers who want to program additional functions for the CHSC subchannel device driver.

Issue ioctls on the misc character device /dev/chsc to obtain information on the current I/O configuration and to dynamically change the I/O configuration. The ioctls and the structures that are passed are defined in the header file arch/s390/include/chsc.h.

To use an ioctl, for example CHSC_START, issue a call of the form:

```
rc = ioctl(fd, CHSC_START, <pointer to chsc_async_area>);
```

Table 50 on page 397 lists the ioctls that are defined.

Table 50. The CHSC ioctls

Name	Structure passed	Description
CHSC_START	struct chsc_async_area	Use to start an asynchronous chsc request that changes the I/O configuration.
CHSC_INFO_CHANNEL_PATH	struct chsc_chp_cd	Use to obtain information about a specific channel path.
CHSC_INFO_CU	struct chsc_cu_cd	Use to obtain information about a specific control unit.
CHSC_INFO_SCH_CU	struct chsc_sch_cud	Use CHSC_INFO_SCH_CU to obtain information about control units on a subchannel.
CHSC_INFO_CI	struct chsc_conf_info	CHSC_INFO_CI can be used to obtain the configuration information as needed by the dynamic I/O configuration chscs.
CHSC_INFO_CCL	struct chsc_comp_list	CHSC_INFO_CCL can be used to obtain information about various configuration components.
CHSC_INFO_CPD	struct chsc_cpd_info	Use CHSC_INFO_CPD to obtain a description of a specified channel path.
CHSC_INFO_DCAL	struct chsc_dcal	Use CHSC_INFO_DCAL to obtain domain attributes of the configuration.
CHSC_START_SYNC	struct chsc_sync_area	Use CHSC_START_SYNC to start a synchronous chsc request.
CHSC_ON_CLOSE_SET	struct chsc_async_area	Install the asynchronous “on close” chsc request. This request is executed when the device node is closed.
CHSC_ON_CLOSE_REMOVE	none	Remove the asynchronous “on close” chsc request.

Chapter 28. ETR- and STP-based clock synchronization

Your Linux instance might be part of an extended remote copy (XRC) setup that requires synchronization of the Linux time-of-day (TOD) clock with a timing network.

Linux on System z supports external time reference (ETR) and system time protocol (STP) based TOD synchronization. ETR and STP work independently of one another. If both ETR and STP are enabled, Linux might use either to synchronize the clock.

For more information about ETR, see the IBM Redbooks technote at `www.ibm.com/redbooks/abstracts/tips0217.html`

For information about STP, see `www.ibm.com/systems/z/advantages/pso/stp.html`

Both ETR and STP support are included in the Linux kernel. No special build options are required.

ETR requires at least one ETR unit that is connected to an external time source. For availability reasons, many installations use a second ETR unit. The ETR units correspond to two ETR ports on Linux. Always set both ports online if two ETR units are available.

Attention: Be sure that a reliable timing signal is available before enabling clock synchronization. With enabled clock synchronization, Linux expects regular timing signals and might stop indefinitely to wait for such signals if it does not receive them.

Enabling clock synchronization when booting

Use kernel parameters to enable clock synchronization when booting.

You can use kernel parameters to set up synchronization for your Linux TOD clock. These kernel parameters specify the initial synchronization settings. On a running Linux instance, you can change these settings through attributes in sysfs (see “Enabling and disabling clock synchronization” on page 401).

Enabling ETR-based clock synchronization

Use the `etr=` kernel parameter to set ETR ports online when Linux is booted.

ETR-based clock synchronization is enabled if at least one ETR port is online.

etr syntax

```
   ┌─etr=off───┐
►►─┼───────────┼──────────────────────────────────────────────────────────►◄
   ├─etr=on────┤
   ├─etr=port0─┤
   └─etr=port1─┘
```

The values have the following effect:

`on` sets both ports online.

`port0`
sets `port0` online and `port1` offline.

`port1`
sets `port1` online and `port0` offline.

`off`
sets both ports offline. With both ports offline, ETR-based clock synchronization is not enabled. This is the default.

Example

To enable ETR-based clock synchronization with both ETR ports online, specify:

```
etr=on
```

Enabling STP-based clock synchronization

Use the `stp=` kernel parameter to enable STP-based clock synchronization when Linux is booted.

stp syntax

```
      ┌─stp=off─┐
►►────┤         ├──────────────────────────────────────────────────────►◄
      └─stp=on──┘
```

By default, STP-based clock synchronization is not enabled.

Example

To enable STP-based clock synchronization, specify:

```
stp=on
```

Enabling and disabling clock synchronization

You can use the sysfs interfaces of ETR and STP to enable and disable clock synchronization on a running Linux instance.

Enabling and disabling ETR-based clock synchronization

Use the ETR sysfs attribute `online` to set an ETR port online or offline.

About this task

ETR-based clock synchronization is enabled if at least one of the two ETR ports is online. ETR-based clock synchronization is switched off if both ETR ports are offline.

Procedure

To set an ETR port online, set its sysfs `online` attribute to `1`. To set an ETR port offline, set its sysfs `online` attribute to `0`. Enter a command of this form:

```
# echo <flag> > /sys/devices/system/etr/etr<n>/online
```

where *<n>* identifies the port and is either `0` or `1`.

Example

To set ETR port `etr1` offline, enter:

```
# echo 0 > /sys/devices/system/etr/etr1/online
```

Enabling and disabling STP-based clock synchronization

Use the STP sysfs attribute `online` to enable or disable STP-based clock synchronization.

Procedure

To enable STP-based clock synchronization, set `/sys/devices/system/stp/online` to 1. To disable STP-based clock synchronization, set this attribute to `0`.

Example

To disable STP-based clock synchronization, enter:

```
# echo 0 > /sys/devices/system/stp/online
```

Chapter 29. Identifying the System z hardware

In installations with several System z mainframes, you might need to identify the particular hardware system on which a Linux instance is running.

Two attributes in `/sys/firmware/ocf` can help you to identify the hardware.

cpc_name
: contains the name that is assigned to the central processor complex (CPC). This name identifies the mainframe system on a Hardware Management Console (HMC).

hmc_network
: contains the name of the HMC network to which the mainframe system is connected.

The two attributes contain the empty string if the Linux instance runs as a guest of a hypervisor that does not support the operations command facility (OCF) communication parameters interface.

Use the **`cat`** command to read these attributes.

Example:

```
# cat /sys/firmware/ocf/cpc_name
Z05
# cat /sys/firmware/ocf/hmc_network
SNA00
```

Chapter 30. The diag288 watchdog device driver

The diag288 watchdog device driver provides Linux watchdog applications with access to the watchdog timer on System z.

You can use the diag288 watchdog in these environments:
- Linux on z/VM
- Linux in LPAR mode if the hardware supports it.

The diag288 watchdog device driver provides the following features:
- Access to the watchdog timer on System z.
- An API for watchdog applications (see "External programming interfaces" on page 410).

Watchdog applications can be used to set up automated restart mechanisms for Linux on System z. Watchdog-based restart mechanisms are an alternative to a networked heartbeat with STONITH (see "STONITH support (snipl for STONITH)" on page 139).

Watchdog applications that communicate directly with the System z firmware or with the z/VM control program (CP) do not require a third operating system to monitor a heartbeat.

What you should know about the diag288 watchdog device driver

The watchdog function comprises two components: a watchdog application on the Linux instance being controlled and a watchdog timer outside the Linux instance.

For Linux in LPAR mode, the timer runs in the System z firmware. For Linux on z/VM the timer is provided by z/VM CP.

While the Linux instance operates satisfactorily, the watchdog application reports a positive status to the watchdog timer at regular intervals. The watchdog application uses a device node to pass these status reports to the timer (Figure 87).

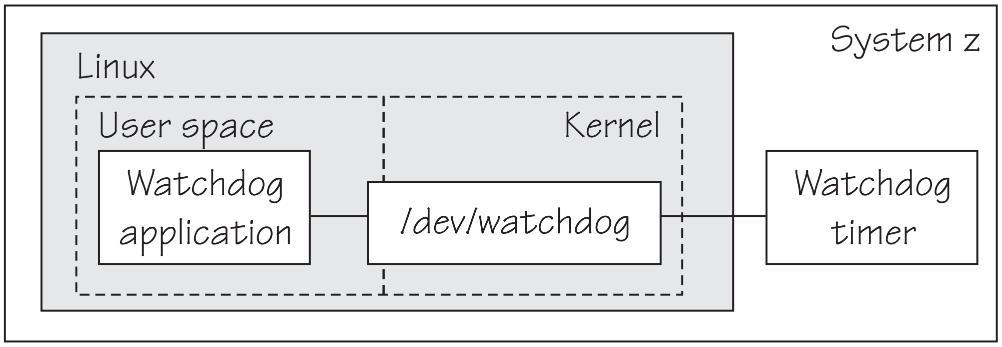


Figure 87. Watchdog application and timer for Linux in LPAR mode

The watchdog application typically derives its status by monitoring critical network connections, file systems, and processes on the Linux instance. If a specified time elapses without a positive report being received by the watchdog timer, the watchdog timer assumes that the Linux instance is in an error state. The watchdog timer then triggers a predefined action against the Linux instance. For example, Linux might be shut down or rebooted, or a system dump might be initiated. For information about setting the default timer and performing other actions, see "External programming interfaces" on page 410.

Linux on z/VM only: Loading or saving a DCSS can take a long time during which the virtual machine does not respond, depending on the size of the DCSS. As a result, a watchdog might time out and restart the guest. You are advised not to use the watchdog in combination with loading or saving DCSSs.

Your distribution might contain a watchdog application. You can also obtain a watchdog application from:

`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`.

Building a kernel with the diag288 watchdog device driver

You must select the kernel configuration option CONFIG_DIAG288_WATCHDOG to be able to use the watchdog device driver.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

```
Device Drivers --->
   ...
   Watchdog Timer Support --->                      (CONFIG_WATCHDOG)
      ...
      --- Watchdog Device Drivers ---
      ...
      System z diag288 Watchdog                     (CONFIG_DIAG288_WATCHDOG)
```

Figure 88. Watchdog kernel configuration option

The watchdog device driver can be compiled into the kernel or as a separate module, diag288_wdt.

CONFIG_DIAG288_WATCHDOG depends on the common code option CONFIG_WATCHDOG.

Setting up the diag288 watchdog device driver

You configure the diag288 watchdog device driver through module or kernel parameters. You also might have to create a device node.

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.

Setting the timeout action

The timeout action for the diag288 watchdog device driver is defined by the `restart` shutdown trigger.

The default action is a **PSW restart** for Linux in LPAR mode and the CP **`system restart`** command for Linux on z/VM. You can change this default by changing the shutdown action for the `restart` shutdown trigger (see Chapter 8, “Shutdown actions,” on page 117).

For Linux on z/VM, you can use the `diag288.cmd=` kernel parameter or the `cmd=` module parameter to directly specify a z/VM CP command to be issued, independent of the `restart` shutdown trigger.

Watchdog kernel parameters

If the watchdog device driver has been compiled into the kernel, you configure the device driver though kernel parameters.

watchdog kernel parameter syntax

```
        ┌─ diag288.cmd="SYSTEM RESTART"─┐
►►──────┼───────────────────────────────┼──┬───────────────────────┬──────────►
        └─ diag288.cmd=<command>────────┘  └─ diag288.conceal=1────┘

                                                 (1)
►──┬────────────────────────────────────┬──────────────────────────────────►◄
   └─ diag288.nowayout=<nowayout_flag>──┘
```

Notes:

1 `diag288.cmd=` and `diag288.conceal=` apply only to Linux on z/VM and are ignored for Linux in LPAR mode.

where:

<command>
: configures the shutdown action to be taken if Linux on z/VM fails.

The default, “SYSTEM RESTART”, configures the shutdown action that is specified for the `restart` shutdown trigger (see Chapter 8, “Shutdown actions,” on page 117).

Any other specification dissociates the timeout action from the `restart` shutdown trigger. Instead, the specification is issued by CP and must adhere to these rules:

- It must be a single valid CP command
- It must not exceed 230 characters
- It must be enclosed by quotation marks if it contains any blanks or newline characters

The specification is converted from ASCII to uppercase EBCDIC.

For details about CP commands, see *z/VM CP Commands and Utilities Reference*, SC24-6175.

On an running instance of Linux on z/VM, you can write to `/sys/module/diag288_wdt/parameters/cmd` to replace the command you specify when loading the module. Through this sysfs interface, you can also specify multiple commands to be issued, see “Example for Linux on z/VM” for more details.

The preferred method for configuring a timeout action other than a system restart is to configure a different shutdown action for the `restart` shutdown trigger.

`diag288.conceal=1`
enables the protected application environment where the guest is protected from unexpectedly entering CP READ. Do not enable the protected environment for guests with multiprocessor configurations. The protected application facility supports only virtual uniprocessor systems.

For details, see the “SET CONCEAL” section of *z/VM CP Commands and Utilities Reference*, SC24-6175.

`<nowayout_flag>`
determines what happens when the watchdog device node is closed by the watchdog application.

If the flag is set to `1`, the watchdog timer keeps running and triggers an action if no positive status report is received within the specified time interval. If the character "V" is written to the device and the flag is set to `0`, the watchdog timer is stopped and the Linux instance continues without the watchdog support.

The default is determined by the common code kernel configuration option CONFIG_WATCHDOG_NOWAYOUT.

Example for Linux on z/VM

The following kernel parameters determine that, on failure, the Linux instance is to be IPLed from a device with devno 0xb1a0. The protected application environment is not enabled. The watchdog application can close the watchdog device node after writing "V" to it. As a result the watchdog timer becomes ineffective and does not IPL the guest.

```
vmwatchdog.cmd="ipl b1a0" vmwatchdog.nowayout=0
```

The following example shows how to specify multiple commands to be issued. This example applies to both the built-in and module version, after booting the kernel or loading the module.

```
# /usr/bin/printf "<cmd1>\n<cmd2>\n<cmd3>" > /sys/module/diag288_wdt/parameters/cmd
```

where *<cmd1>*, *<cmd2>*, and *<cmd3>*are z/VM commands.

Use the **`printf`** version at `/usr/bin/printf`. The built-in **`printf`** command from bash might not process the newline characters as intended.

To verify that your commands have been accepted, issue:

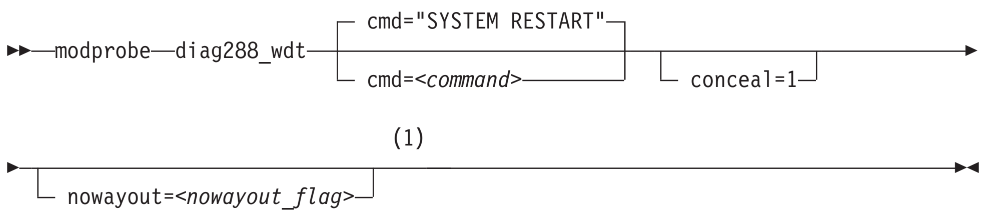
```
# cat /sys/module/diag288_wdt/parameters/cmd
<cmd1>
<cmd2>
<cmd3>
```

Note: You cannot specify multiple commands as kernel parameters during boot time or module parameters while loading the module.

Module parameters

If the watchdog device driver has been built as a separate module, the device driver is configured through module parameters.

watchdog module parameter syntax



Notes:

1 `cmd=` and `conceal=` apply only to Linux on z/VM and are ignored for Linux in LPAR mode.

The variables have the same meaning as the corresponding kernel parameters with prefix `diag288.` in "Watchdog kernel parameters" on page 407.

Example for Linux on z/VM

The following command loads the watchdog module and determines that, on failure, the Linux instance is to be IPLed from a device with devno 0xb1a0. The protected application environment is not enabled. The watchdog application can close the watchdog device node after writing "V" to it. As a result the watchdog timer becomes ineffective and does not IPL the guest.

```
# modprobe diag288_wdt cmd="ipl b1a0" nowayout=0
```

Assuring that a device node exists

The watchdog application on Linux needs a misc character device to communicate with the watchdog timer.

About this task

The watchdog device node is typically called `/dev/watchdog`. If your distribution does not create the device node for you, you must create a node.

Procedure

1. Check whether there is already a device node by issuing the following command:

```
# find /dev -name watchdog
```

If your distribution provides the watchdog device driver as a separate module, be sure to load the module before you check for the node.

2. If there is no node, use major number 10 and minor number 130 to create one. Issue

```
# mknod /dev/watchdog c 10 130
```

External programming interfaces

There is an API for applications that work with the watchdog device driver.

Application programmers: This information is intended for programmers who want to write watchdog applications that work with the watchdog device driver.

For information about the API, see the following files in the Linux source tree:

- `Documentation/watchdog/watchdog-api.txt`
- `include/linux/watchdog.h`

The default watchdog timeout is 60 seconds, the minimum timeout that can be set through the IOCTL SETTIMEOUT is 15 seconds.

The following IOCTLs are supported:

- WDIOC_SETTIMEOUT
- WDIOC_KEEPALIVE

Part 6. z/VM virtual server integration

These device drivers and features help you to effectively run and manage a z/VM-based virtual Linux server farm.

Newest version

You can find the newest version of this publication at www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Restrictions

For prerequisites and restrictions see www.ibm.com/developerworks/linux/linux390/development_technical.html

www.ibm.com/developerworks/linux/linux390/development_restrictions.html

Chapter 31. z/VM concepts

The z/VM performance monitoring and cooperative memory management concepts help you to understand how the different components interact with Linux.

Timed page pool chart description

This chart shows a fixed amount of memory that is allocated to Linux. In a succession of time intervals, the timed page pool takes different amounts of memory from this fixed amount. Only the remaining amount of memory is available to Linux. The amount of memory that is taken away from Linux during an individual interval is not constant but declines at a constant rate during the interval.

Performance monitoring for z/VM guest virtual machines

You can monitor the performance of z/VM guest virtual machines and their guest operating systems with performance monitoring tools on z/VM or on Linux.

These tools can be your own, IBM tools such as the Performance Toolkit for VM, or third-party tools. The guests being monitored require agents that write monitor data.

Monitoring on z/VM

z/VM monitoring tools must read performance data. For monitoring Linux instances, this data is APPLDATA monitor records.

Linux instances must write these records for the tool to read, as shown in Figure 89 on page 414.

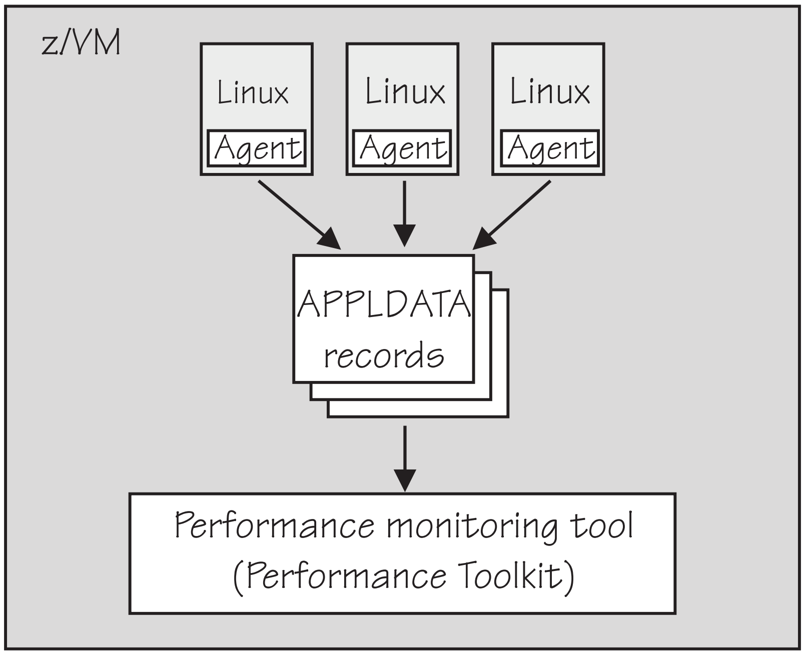


Figure 89. Linux instances write APPLDATA records for performance monitoring tools

Both user space applications and the Linux kernel can write performance data to APPLDATA records. Applications use the monwriter device driver to write APPLDATA records. The Linux kernel can be configured to collect system level data such as memory, CPU usage, and network-related data, and write it to data records.

For file system size data, there is a command, **mon_fsstatd**. This user space tool uses the monwriter device driver to write file system size information as defined records.

For process data, there is a command, **mon_procd**. This user space tool uses the monwriter device driver to write system information as defined records.

In summary, Linux on System z supports writing and collecting performance data as follows:
- The Linux kernel can write z/VM monitor data for Linux instances, see Chapter 32, "Writing kernel APPLDATA records," on page 419.
- Linux applications that are running on z/VM guests can write z/VM monitor data, see Chapter 33, "Writing z/VM monitor records," on page 427.
- You can collect monitor file system size information, see "mon_fsstatd – Monitor z/VM guest file system size" on page 642.
- You can collect system information about up to 100 concurrently running processes, see "mon_procd – Monitor Linux on z/VM" on page 647.

Monitoring on Linux

A Linux instance can read the monitor data by using the monreader device driver.

Figure 90 on page 415 illustrates a Linux instance that is set up to read the monitor data. You can use an existing monitoring tool or write your own software.

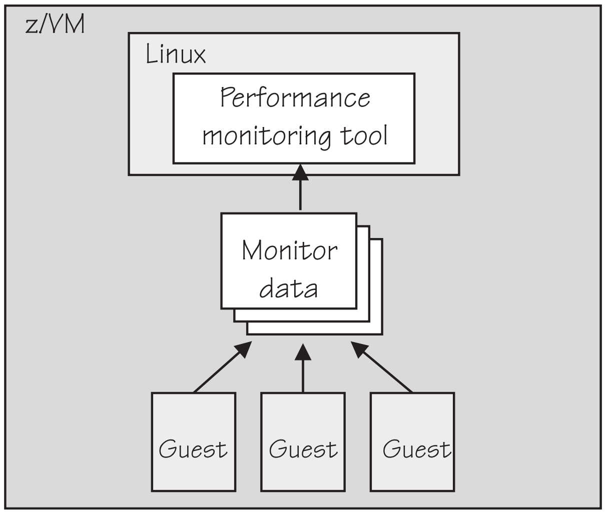


Figure 90. Performance monitoring using monitor DCSS data

In summary, Linux on System z supports reading performance data in the form of read access to z/VM monitor data for Linux instances. See Chapter 34, "Reading z/VM monitor records," on page 431 for more details.

Further information

Several z/VM publications include information about monitoring.

- See *z/VM Getting Started with Linux on System z*, SC24-6194, the chapter on monitoring performance for information about using the CP Monitor and the Performance Toolkit for VM.
- See *z/VM Saved Segments Planning and Administration*, SC24-6229 for general information about DCSSs (z/VM keeps monitor records in a DCSS).
- See *z/VM Performance*, SC24-6208 for information about creating a monitor DCSS.
- See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about the CP commands that are used in the context of DCSSs and for controlling the z/VM monitor system service.
- For the layout of the monitor records, visit www.ibm.com/vm/pubs/ctlblk.html and see Chapter 32, "Writing kernel APPLDATA records," on page 419.
- For more information about performance monitoring on z/VM, visit `www.ibm.com/vm/perf`

Cooperative memory management background

Cooperative memory management (CMM, or "cmm1") dynamically adjusts the memory available to Linux.

For information about setting up CMM, see Chapter 41, "Cooperative memory management," on page 475.

In a virtualized environment it is common practice to give the virtual machines more memory than is actually available to the hypervisor. Linux tends to use all of its available memory. As a result, the hypervisor (z/VM) might start swapping.

To avoid excessive z/VM swapping, the memory available to Linux can be reduced. CMM allocates pages to page pools that make the pages unusable to

Linux. There are two such page pools as shown in Figure 91.

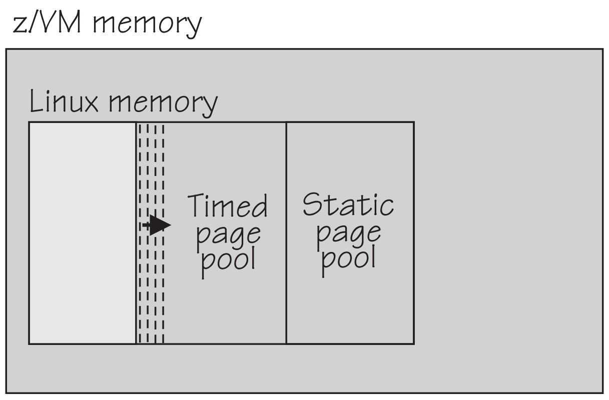


Figure 91. Page pools

There are two page pools:

A static page pool
The page pool is controlled by a resource manager that changes the pool size at intervals according to guest activity as well as overall memory usage on z/VM (see Figure 92).

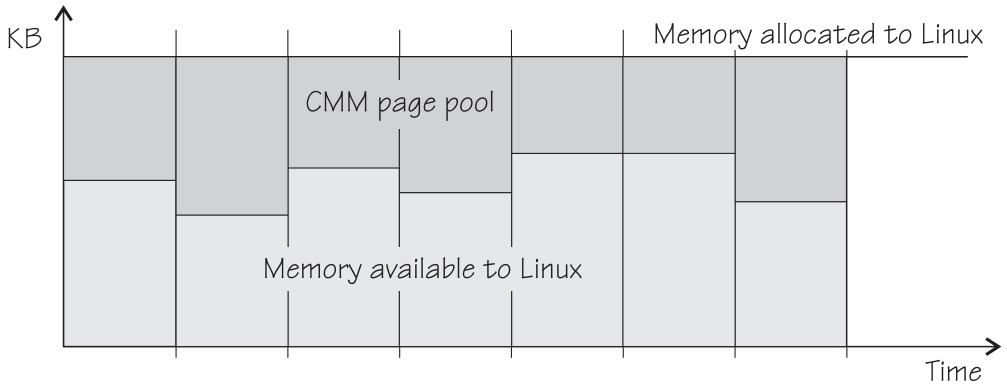


Figure 92. Static page pool. The size of the pool is static during an interval.

A timed page pool
Pages are released from this pool at a speed that is set in the *release rate* (see Figure 93 on page 417). According to guest activity and overall memory usage on z/VM, a resource manager adds pages at intervals. If no pages are added and the release rate is not zero, the pool empties.

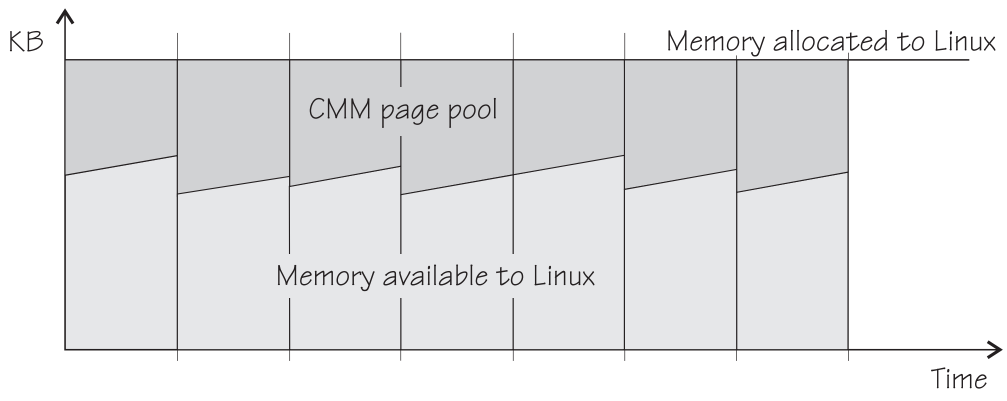


Figure 93. Timed page pool. Pages are freed at a set release rate.

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 information. For more details, see the chapter about VMRM in *z/VM Performance*, SC24-6208.

Third-party tools can provide a Linux deamon that receives commands for the memory allocation through TCP/IP. The deamon, in turn, uses the procfs-based interface. You can use the procfs interface to read the pool sizes. These values are useful diagnostic data.

Linux guest relocation

Information about guest relocations is stored in the s390 debug feature (s390dbf).

You can access this information in a kernel dump or from a running Linux instance. For more information, see *Using the Dump Tools*, SC33-8412.

Chapter 32. Writing kernel APPLDATA records

z/VM is a convenient point for collecting z/VM guest performance data and statistics for an entire server farm. Linux instances can export such data to z/VM by using APPLDATA monitor records.

z/VM regularly collects these records. The records are then available to z/VM performance monitoring tools.

A virtual CPU timer on the Linux instance to be monitored controls when data is collected. The timer accounts for only busy time to avoid unnecessarily waking up an idle guest. The APPLDATA record support comprises several modules. A base module provides an intra-kernel interface and the timer function. The intra-kernel interface is used by *data gathering modules* that collect actual data and determine the layout of a corresponding APPLDATA monitor record (see “APPLDATA monitor record layout” on page 422).

For an overview of performance monitoring support, see “Performance monitoring for z/VM guest virtual machines” on page 413.

Building a kernel that is enabled for monitoring

Control the build options for the APPLDATA record support through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

```
Virtualization --->
   ...
   Linux - VM Monitor Stream, base infrastructure     (CONFIG_APPLDATA_BASE)*
   ├─ Monitor memory management statistics            (CONFIG_APPLDATA_MEM)*
   ├─ Monitor OS statistics                           (CONFIG_APPLDATA_OS)
   └─ Monitor overall network statistics              (CONFIG_APPLDATA_NET_SUM)
```

Figure 94. Linux monitor stream kernel configuration menu options

CONFIG_APPLDATA_BASE
: This option provides the base component for the APPLDATA record support.

CONFIG_APPLDATA_MEM
: This option provides monitoring for memory-related data. It can be compiled into the kernel or as a separate module, appldata_mem.

CONFIG_APPLDATA_OS
: This option provides monitoring for operating system-related data, for example, CPU usage. It can be compiled into the kernel or as a separate module, appldata_os.

CONFIG_APPLDATA_NET_SUM
: This option provides monitoring for network-related data. It can be compiled into the kernel or as a separate module, appldata_net_sum.

Setting up the APPLDATA record support

You must enable your z/VM guest virtual machine for data gathering. You must also load any components of the APPLDATA record support that were compiled as separate modules.

Procedure

1. On z/VM, ensure that the user directory of the guest virtual machine includes the option APPLMON.
2. On Linux, use the **modprobe** command to load any required modules that were not compiled into the kernel.

APPLDATA record support module parameter syntax

```
►►──modprobe──┬─ appldata_mem──────┬──────────────────────────────────────►◄
              ├─ appldata_os───────┤
              └─ appldata_net_sum──┘
```

where appldata_mem, appldata_os, and appldata_net_sum are the modules for gathering memory-related data, operating system-related data, and network-related data.

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

Generating APPLDATA monitor records

You can set the timer interval and enable or disable data collection.

You control the monitor stream support through the procfs. APPLDATA monitor records are produced if both a particular data-gathering module and the monitoring support in general are enabled.

Enabling or disabling the support

Use the procfs `timer` attribute to enable or disable the monitoring support.

Procedure

To read the current setting, issue:

```
# cat /proc/sys/appldata/timer
```

To enable the monitoring support, issue:

```
# echo 1 > /proc/sys/appldata/timer
```

To disable the monitoring support, issue:

```
# echo 0 > /proc/sys/appldata/timer
```

Activating or deactivating individual data-gathering modules

Each data-gathering module has a procfs entry that contains a value `1` if the module is active and `0` if the module is inactive.

About this task

The following procfs entries control the data-gathering modules:

`/proc/sys/appldata/mem` for the memory data-gathering module

`/proc/sys/appldata/os` for the CPU data-gathering module

`/proc/sys/appldata/net_sum` for the net data-gathering module

To check whether a module is active, look at the content of the corresponding procfs entry.

Procedure

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 enabled (see "Enabling or disabling the support" on page 420).

Example

To find out whether memory data-gathering is active, issue:

```
# cat /proc/sys/appldata/mem
0
```

In the example, memory data-gathering is off. To activate memory data-gathering, issue:

```
# echo 1 > /proc/sys/appldata/mem
```

To deactivate the memory data-gathering module, issue:

```
# echo 0 > /proc/sys/appldata/mem
```

Setting the sampling interval

You can set the time that lapses between consecutive data samples.

About this task

The time that you set is measured by the virtual CPU timer. Because the virtual timer slows down as the guest idles, the sampling interval in real time can be considerably longer than the value you set.

The value in `/proc/sys/appldata/interval` is the sample interval in milliseconds. The default sample interval is 10 000 ms.

Procedure

To read the current value, issue:

```
# cat /proc/sys/appldata/interval
```

To set the sample interval to a different value, write the new value (in milliseconds) to `/proc/sys/appldata/interval`. Issue a command of this form:

```
# echo <interval> > /proc/sys/appldata/interval
```

where <*interval*> is the new sample interval in milliseconds. Valid input must be greater than 0 and less than $2^{31} - 1$. Input values greater than $2^{31} - 1$ produce unpredictable results.

Example

To set the sampling interval to 20 s (20000 ms), issue:

```
# echo 20000 > /proc/sys/appldata/interval
```

APPLDATA monitor record layout

Each of the data-gathering modules writes a different type of record.
- Memory data (see Table 51 on page 423)
- Processor data (see Table 52 on page 423)
- Networking (see Table 53 on page 424)

z/VM can identify the records by their unique product ID. The product ID is an EBCDIC string of this form: “LINUXKRNL<*record ID*>260100”. The <*record ID*> is treated as a byte value, not a string.

The records contain data of the following types:

u32	unsigned 4-byte integer.
u64	unsigned 8-byte integer.

Important: On 31-bit Linux systems, the u64 values are actually only 32-bit values. That is, the lower 32-bit wrap around like 32-bit counters and the upper 32 bit are always zero.

Table 51. APPLDATA_MEM_DATA record (Record ID 0x01)

Offset (Decimal)	Offset (Hex)	Type	Name	Description
0	0x0	u64	timestamp	TOD time stamp that is generated on the Linux side after record update
8	0x8	u32	sync_count_1	After z/VM collected the record data, sync_count_1 and sync_count_2 must be the same. Otherwise, the record was updated on the Linux side while z/VM was collecting the data. As a result, the data might be inconsistent.
12	0xC	u32	sync_count_2	See sync_count_1.
16	0x10	u64	pgpgin	Data that was read from disk (in KB)
24	0x18	u64	pgpgout	Data that was written to disk (in KB)
32	0x20	u64	pswpin	Pages that were swapped in
40	0x28	u64	pswpout	Pages that were swapped out
48	0x30	u64	sharedram	Shared RAM in KB, set to 0
56	0x38	u64	totalram	Total usable main memory size in KB
64	0x40	u64	freeram	Available memory size in KB
72	0x48	u64	totalhigh	Total high memory size in KB
80	0x50	u64	freehigh	Available high memory size in KB
88	0x58	u64	bufferram	Memory that was reserved for raw disk blocks, corresponding to "Buffers" from `/proc/meminfo`, in KB
96	0x60	u64	cached	Size of used cache, including "Cached" and "SwapCached" from `/proc/meminfo`, in KB
104	0x68	u64	totalswap	Total swap space size in KB
112	0x70	u64	freeswap	Free swap space in KB
120	0x78	u64	pgalloc	Page allocations
128	0x80	u64	pgfault	Page faults (major+minor)
136	0x88	u64	pgmajfault	Page faults (major only)

Table 52. APPLDATA_OS_DATA record (Record ID 0x02)

Offset (Decimal)	Offset (Hex)	Type (size)	Name	Description
0	0x0	u64	timestamp	TOD time stamp that is generated on the Linux side after record update
8	0x8	u32	sync_count_1	After z/VM collected the record data, sync_count_1 and sync_count_2 must be the same. Otherwise, the record was updated on the Linux side while z/VM was collecting the data. As a result, the data might be inconsistent.
12	0xC	u32	sync_count_2	See sync_count_1.
16	0x10	u32	nr_cpus	Number of virtual CPUs.
20	0x14	u32	per_cpu_size	Size of the per_cpu_data for each CPU (= 36).

Table 52. APPLDATA_OS_DATA record (Record ID 0x02) (continued)

Offset (Decimal)	Offset (Hex)	Type (size)	Name	Description
24	0x18	u32	cpu_offset	Offset of the first per_cpu_data (= 52).
28	0x1C	u32	nr_running	Number of runnable threads.
32	0x20	u32	nr_threads	Number of threads.
36	0x24	3 × u32	avenrun[3]	Average number of running processes during the last 1 (first value), 5 (second value) and 15 (third value) minutes. These values are "fake fix-point". Each value is composed of a 10-bit integer and an 11-bit fractional part. See note 1 at the end of this table.
48	0x30	u32	nr_iowait	Number of blocked threads (waiting for I/O).
52	0x34	See note 2.	per_cpu_data	Time spent in user, kernel, idle, nice, etc for every CPU. See note 3 at the end of this table.
52	0x34	u32	per_cpu_user	Timer ticks that were spent in user mode.
56	0x38	u32	per_cpu_nice	Timer ticks that were spent with modified priority.
60	0x3C	u32	per_cpu_system	Timer ticks that were spent in kernel mode.
64	0x40	u32	per_cpu_idle	Timer ticks that were spent in idle mode.
68	0x44	u32	per_cpu_irq	Timer ticks that were spent in interrupts.
72	0x48	u32	per_cpu_softirq	Timer ticks that were spent in softirqs.
76	0x4C	u32	per_cpu_iowait	Timer ticks that were spent while waiting for I/O.
80	0x50	u32	per_cpu_steal	Timer ticks "stolen" by the hypervisor.
84	0x54	u32	cpu_id	The number of this CPU.

Note:

1. The following C-Macros are used inside Linux to transform these into values with two decimal places:

   ```
   #define LOAD_INT(x) ((x) >> 11)
   #define LOAD_FRAC(x) LOAD_INT(((x) & ((1 << 11) - 1)) * 100)
   ```

2. nr_cpus * per_cpu_size
3. per_cpu_user through cpu_id are repeated for each CPU

Table 53. APPLDATA_NET_SUM_DATA record (Record ID 0x03)

Offset (Decimal)	Offset (Hex)	Type	Name	Description
0	0x0	u64	timestamp	TOD time stamp that is generated on the Linux side after record update
8	0x8	u32	sync_count_1	After z/VM collected the record data, sync_count_1 and sync_count_2 must be the same. Otherwise, the record was updated on the Linux side while z/VM was collecting the data. As a result, the data might be inconsistent.

Table 53. APPLDATA_NET_SUM_DATA record (Record ID 0x03) (continued)

Offset (Decimal)	Offset (Hex)	Type	Name	Description
12	0xC	u32	sync_count_2	See sync_count_1.
16	0x10	u32	nr_interfaces	Number of interfaces being monitored
20	0x14	u32	padding	Unused. The next value is 64-bit aligned, so these 4 bytes would be padded out by compiler
24	0x18	u64	rx_packets	Total packets that were received
32	0x20	u64	tx_packets	Total packets that were transmitted
40	0x28	u64	rx_bytes	Total bytes that were received
48	0x30	u64	tx_bytes	Total bytes that were transmitted
56	0x38	u64	rx_errors	Number of bad packets that were received
64	0x40	u64	tx_errors	Number of packet transmit problems
72	0x48	u64	rx_dropped	Number of incoming packets that were dropped because of insufficient space in Linux buffers
80	0x50	u64	tx_dropped	Number of outgoing packets that were dropped because of insufficient space in Linux buffers
88	0x58	u64	collisions	Number of collisions while transmitting

Programming interfaces

The monitor stream support base module exports two functions.

- 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 422 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 33. Writing z/VM monitor records

Applications can use the monitor stream application device driver to write z/VM monitor APPLDATA records to the z/VM *MONITOR stream.

For an overview of performance monitoring support, see “Performance monitoring for z/VM guest virtual machines” on page 413.

The monitor stream application device driver interacts with the z/VM monitor APPLDATA facilities for performance monitoring. A better understanding of these z/VM facilities might help when you are using this device driver. See *z/VM Performance*, SC24-6208 for information about monitor APPLDATA.

The monitor stream application device driver provides the following functions:
- An interface to the z/VM monitor stream.
- A means of writing z/VM monitor APPLDATA records.

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

To build a kernel with the monitor stream application device driver, you must select option CONFIG_MONWRITER in the configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

```
Device Drivers --->
   ...
   Character devices --->
      ...
      --- S/390 character device drivers (depends on S390) ---
      ...
      API for writing z/VM monitor service records         (CONFIG_MONWRITER)
```

The monitor stream write support can be compiled into the kernel or as a separate module, MONWRITER.

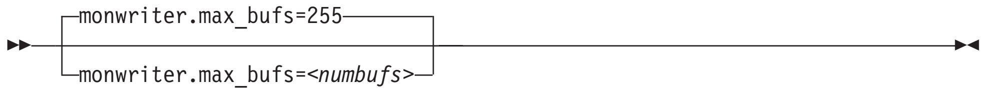
Setting up the z/VM *MONITOR record writer device driver

On Linux, configure the z/VM *MONITOR record writer device driver through kernel or module parameters. You also must set up your guest virtual machine for monitor records on z/VM.

Kernel parameters

If the the z/VM *MONITOR record writer device driver was compiled into the kernel, you configure the device driver by adding parameters to the kernel parameter line.

Monitor stream application device driver kernel parameter syntax



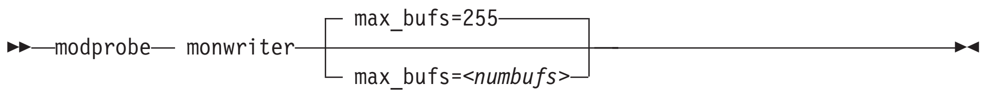
where <*numbufs*> is the maximum number of monitor sample and configuration data buffers that can exist in the Linux instance at one time. The default is 255.

Module parameters

If the z/VM *MONITOR record writer device driver was compiled as a separate module, you configure the device driver through module parameters when you load the device driver module.

This section describes how to load and configure those components that were compiled as separate modules.

Monitor stream application device driver module parameter syntax



where <*numbufs*> is the maximum number of monitor sample and configuration data buffers that can exist in the Linux guest at one time. The default is 255.

Example

If you have compiled the monitor stream application device driver as a separate module, you must load it before you can work with it. To load the monwriter module and set the maximum number of buffers to 400, use the following command:

```
# modprobe monwriter max_bufs=400
```

Setting up the user z/VM guest virtual machine

You must enable your z/VM guest virtual machine to write monitor records and configure the z/VM system to collect these records.

Procedure

Perform these steps:

1. Set this option in the z/VM user directory entry of the virtual machine in which the application that uses this device driver is to run:
   - OPTION APPLMON

2. Issue the following CP commands to have CP collect the respective types of monitor data:
   - `MONITOR SAMPLE ENABLE APPLDATA ALL`
   - `MONITOR EVENT ENABLE APPLDATA ALL`

   You can log in to the z/VM console to issue the CP commands. These commands must be preceded with #CP. Alternatively, you can use the **`vmcp`** command for issuing CP commands from your Linux instance.

   See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about the CP MONITOR command.

Working with the z/VM *MONITOR record writer

The monitor stream application device driver uses the z/VM CP instruction DIAG X'DC' to write to the z/VM monitor stream. Monitor data must be preceded by a data structure, monwrite_hdr.

See *z/VM CP Programming Services*, SC24-6179 for more information about the DIAG X'DC' instruction and the different monitor record types (sample, config, event).

The application writes monitor data by passing a monwrite_hdr structure that is followed by monitor data. The only exception is the STOP function, which requires no monitor data. The monwrite_hdr structure, as described in monwriter.h, is filled in by the application. The structure includes the DIAG X'DC' function to be performed, the product identifier, the header length, and the data length.

All records that are written to the z/VM monitor stream begin with a product identifier. This device driver uses the product ID. The product ID is a 16-byte structure of the form pppppppffnvvrrmm, where:

ppppppp
: is a fixed ASCII string, for example, LNXAPPL.

ff
: is the application number (hexadecimal number). This number can be chosen by the application. You can reduce the chance of conflicts with other applications, by requesting an application number from the IBM z/VM Performance team at

  `www.ibm.com/vm/perf`

n
: is the record number as specified by the application.

vv, rr, and mm
: can also be specified by the application. A possible use is to specify version, release, and modification level information, allowing changes to a certain record number when the layout is changed, without changing the record number itself.

The first 7 bytes of the structure (LNXAPPL) are filled in by the device driver when it writes the monitor data record to the CP buffer. The last 9 bytes contain information that is supplied by the application on the `write()` call when writing the data.

The monwrite_hdr structure that must be written before any monitor record data is defined as follows:

```
/* the header the app uses in its write() data */
struct monwrite_hdr {
 unsigned char mon_function;
 unsigned short applid;
 unsigned char record_num;
 unsigned short version;
 unsigned short release;
 unsigned short mod_level;
 unsigned short datalen;
 unsigned char hdrlen;
}__attribute__((packed));
```

The following function code values are defined:

```
/* mon_function values */
#define MONWRITE_START_INTERVAL 0x00 /* start interval recording */
#define MONWRITE_STOP_INTERVAL  0x01 /* stop interval or config recording */
#define MONWRITE_GEN_EVENT      0x02 /* generate event record */
#define MONWRITE_START_CONFIG   0x03 /* start configuration recording */
```

Writing data and stopping data-writing

Applications use the `open()`, `write()`, and `close()` calls to work with the z/VM monitor stream.

Before an application can write monitor records, it must issue `open()` to open the device driver. Then, the application must issue `write()` calls to start or stop the collection of monitor data and to write any monitor records to buffers that CP can access.

When the application has finished writing monitor data, it must issue `close()` to close the device driver.

Using the monwrite_hdr structure

The structure monwrite_hdr is used to pass DIAG x'DC' functions and the application-defined product information to the device driver on `write()` calls.

When the application calls `write()`, the data it is writing consists of one or more monwrite_hdr structures. Each structure is followed by monitor data. The only exception is the STOP function, which is not followed by data.

The application can write to one or more monitor buffers. A new buffer is created by the device driver for each record with a unique product identifier. To write new data to an existing buffer, an identical monwrite_hdr structure must precede the new data on the `write()` call.

The monwrite_hdr structure also includes a field for the header length, which is useful for calculating the data offset from the beginning of the header. There is also a field for the data length, which is the length of any monitor data that follows. See `/usr/include/asm-s390/monwriter.h` for the definition of the monwrite_hdr structure.

Chapter 34. Reading z/VM monitor records

Monitoring software on Linux can access z/VM guest data through the z/VM *MONITOR record reader device driver.

z/VM uses the z/VM monitor system service (*MONITOR) to collect monitor records from agents on its guests. z/VM writes the records to a discontiguous saved segment (DCSS). The z/VM *MONITOR record reader device driver uses IUCV to connect to *MONITOR and accesses the DCSS as a character device.

For an overview of performance monitoring support, see “Performance monitoring for z/VM guest virtual machines” on page 413.

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.
- Access to *MONITOR records as described on `www.ibm.com/vm/pubs/ctlblk.html`
- Access to the kernel APPLDATA records from the Linux monitor stream (see Chapter 32, “Writing kernel APPLDATA records,” on page 419).

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

The data that is collected by *MONITOR depends on the setup of the monitor stream service.

The z/VM *MONITOR record reader device driver only reads data from the monitor DCSS; it does not control the system service.

z/VM supports only one monitor DCSS. All monitoring software that requires monitor records from z/VM uses the same DCSS to read *MONITOR data. Usually, a DCSS called MONDCSS is already defined and used by existing monitoring software.

If a monitor DCSS is already defined, you must use it. To find out whether a monitor DCSS exists, issue the following CP command from a z/VM guest virtual machine with privilege class E:

```
q monitor
```

The command output also shows the name of the DCSS.

Further information

- See *z/VM Saved Segments Planning and Administration*, SC24-6229 for general information about DCSSs.
- See *z/VM Performance*, SC24-6208 for information about creating a monitor DCSS.

- See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about the CP commands that are used in the context of DCSSs and for controlling the z/VM monitor system service.
- For the layout of the monitor records, go to www.ibm.com/vm/pubs/ctlblk.html and click the link to the monitor record format for your z/VM version. Also, see Chapter 32, "Writing kernel APPLDATA records," on page 419.

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

You must select the kernel configuration option CONFIG_MONREADER to compile a kernel with the z/VM *MONITOR record reader device driver.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

```
Device Drivers --->
   ...
   Character devices --->
      ...
      --- S/390 character device drivers (depends on S390) ---
      ...
      API for reading z/VM monitor service records           (CONFIG_MONREADER)
```

*Figure 95. z/VM *MONITOR record kernel configuration menu options*

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

You also need IUCV support.

Setting up the z/VM *MONITOR record reader device driver

You must set up a Linux instance and the z/VM guest virtual machine for accessing an existing monitor DCSS with the z/VM *MONITOR record reader device driver.

Before you begin

Some of the CP commands you use for setting up the z/VM *MONITOR record reader device driver require class E authorization.

Setting up the monitor system service and the monitor DCSS on z/VM is beyond the scope of this information. See "What you should know about the z/VM *MONITOR record reader device driver" on page 431 for documentation about the monitor system service, DCSS, and related CP commands.

Providing the required z/VM user directory statements

The z/VM guest virtual machine where your Linux instance is to run must be permitted to establish an IUCV connection to the z/VM *MONITOR system service.

Procedure

Ensure that the guest entry in the user directory includes the following statement:

```
IUCV *MONITOR
```

If the DCSS is restricted, you also need this statement:

```
NAMESAVE <dcss>
```

where *<dcss>* is the name of the DCSS that is used for the monitor records. You can find out the name of an existing monitor DCSS by issuing the following CP command from a z/VM guest virtual machine with privilege class E:

```
q monitor
```

Assuring that the DCSS is addressable for your Linux instance

The DCSS address range must not overlap with the storage of you z/VM guest virtual machine.

Procedure

To find out the start and end address of the DCSS, issue the following CP command from a z/VM guest virtual machine with privilege class E:

```
q nss map
```

The output gives you the start and end addresses of all defined DCSSs in units of 4-kilobyte pages. For example:

```
00: FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
...
00: 0011 MONDCSS  CPDCSS   N/A     09000  097FF  SC   R   00003 N/A      N/A
...
```

What to do next

If the DCSS overlaps with the guest storage, follow the procedure in "Avoiding overlaps with your guest storage" on page 453.

Specifying the monitor DCSS name

You can specify the DCSS name as a module parameter when you load the device driver module, or as a kernel parameter if the device driver is compiled into the kernel.

About this task

By default, the z/VM *MONITOR record reader device driver assumes that the monitor DCSS on z/VM is called MONDCSS. If you want to use a different DCSS name, you must specify it. Proceed according to your distribution:

- If your device driver is compiled into the kernel, specify the DCSS name as a kernel parameter.
- If your device driver is compiled as a separate module, specify the DCSS name as a module parameter when you load the module.

Kernel parameter

You can specify a DCSS name by adding the mondcss parameter to the kernel parameter line.

About this task

This section describes how to specify a DCSS name if the z/VM *MONITOR record reader device driver is compiled into the kernel.

z/VM *MONITOR stream read support kernel parameter syntax

```
   ┌─monreader.mondcss=MONDCSS─┐
►►─┼───────────────────────────┼──────────────────────────────►◄
   └─monreader.mondcss=<dcss>──┘
```

where <*dcss*> is the name of the DCSS that z/VM uses for the monitor records.

Example

To specify MYDCSS as the DCSS name add the following parameter to the kernel parameter line:

```
monreader.mondcss=MYDCSS
```

The value is automatically converted to uppercase.

Module parameter

You can load the monitor read support if it is compiled as separate module. You can also specify a DCSS name, if applicable.

About this task

Load the monitor read support module with **modprobe** to assure that any other required modules are also loaded. You need IUCV support if you want to use the monitor read support.

monitor stream support module parameter syntax

```
                          ┌─ mondcss=MONDCSS─┐
►►──modprobe── monreader──┼──────────────────┼──────────────────►◄
                          └─ mondcss=<dcss>──┘
```

where <*dcss*> is the name of the DCSS that z/VM uses for the monitor records.

Example

To load the monitor read support module and specify MYDCSS as the DCSS issue:

```
modprobe monreader mondcss=mydcss
```

Assuring that the required device node exists

You need a device node for a miscellaneous character device to access the monitor DCSS.

About this task

Your distribution might create this device node for you (for example, by using udev).

Procedure

Perform these steps:

1. To find out whether there is already a device node, issue this command:

```
# find /dev -name monreader
```

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

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

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

3. To create a device node, issue a command of the form:

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

   where <*node*> is your device node.

Example

To create a device node `/dev/monreader`:

```
# cat /sys/class/misc/monreader/dev
10:63
# mknod /dev/monreader c 10 63
```

In the example, the major number was 10 and the minor 63.

Working with the z/VM *MONITOR record reader support

You can open the z/VM *MONITOR record character device to read records from it.

This section describes how to work with the monitor read support.

- "Opening and closing the character device"
- "Reading monitor records" on page 436

Opening and closing the character device

Only one user can open the character device at any one time. Once you have opened the device, you must close it to make it accessible to other users.

About this task

The open function can fail (return a negative value) with one of the following values for errno:

EBUSY
The device has already been opened by another user.

EIO
No IUCV connection to the z/VM MONITOR system service could be established. An error message with an IPUSER SEVER code is printed into syslog. See *z/VM Performance*, SC24-6208 for details about the codes.

Once the device is opened, incoming messages are accepted and account for the message limit. If you keep the device open indefinitely, expect to eventually reach the message limit (with error code EOVERFLOW).

Reading monitor records

You can either read in non-blocking mode with polling, or you can read in blocking mode without polling.

About this task

Reading from the device provides a 12-byte monitor control element (MCE), followed by a set of one or more contiguous monitor records (similar to the output of the CMS utility MONWRITE without the 4 K control blocks). The MCE contains information about:

- The type of the following record set (sample/event data)
- The monitor domains contained within it
- The start and end address of the record set in the monitor DCSS

The start and end address can be used to determine the size of the record set. The end address is the address of the last byte of data. The start address is needed to handle "end-of-frame" records correctly (domain 1, record 13), that is, it can be used to determine the record start offset relative to a 4 K page (frame) boundary.

See "Appendix A: *MONITOR" in *z/VM Performance*, SC24-6208 for a description of the monitor control element layout. The layout of the monitor records can be found on

`www.ibm.com/vm/pubs/ctlblk.html`

The layout of the data stream that is provided by the monreader device is as follows:

```
...
<0 byte read>
<first MCE>              \
<first set of records>    |...
...                       |- data set
<last MCE>                |
<last set of records>    /
<0 byte read>
...
```

There might be more than one combination of MCE and a corresponding record set within one data set. The end of each data set is indicated by a successful read with a return value of 0 (0 byte read). Received data is not to be considered valid unless a complete record set is read successfully, including the closing 0-Byte read. You are advised to always read the complete set into a user space buffer before processing the data.

When designing a buffer, allow for record sizes up to the size of the entire monitor DCSS, or use dynamic memory allocation. The size of the monitor DCSS will be printed into syslog after loading the module. You can also use the (Class E privileged) CP command **Q NSS MAP** to list all available segments and information about them (see "Assuring that the DCSS is addressable for your Linux instance" on page 433).

Error conditions are indicated by returning a negative value for the number of bytes read. For an error condition, the errno variable can be:

EIO
Reply failed. All data that was read since the last successful read with 0 size is not valid. Data is missing. The application must decide whether to continue reading subsequent data or to exit.

EFAULT
Copy to user failed. All data that was read since the last successful read with 0 size is not valid. Data is missing. The application must decide whether to continue reading subsequent data or to exit.

EAGAIN
Occurs on a non-blocking read if there is no data available at the moment. No data is missing or damaged, retry or use polling for non-blocking reads.

EOVERFLOW
The message limit is reached. The data that was read since the last successful read with 0 size is valid, but subsequent records might be missing. The application must decide whether to continue reading subsequent data or to exit.

Chapter 35. z/VM recording device driver

The z/VM recording device driver enables Linux on z/VM to read from the CP recording services and, thus, act as a z/VM wide control point.

The z/VM recording device driver uses the z/VM CP RECORDING command to collect records and IUCV to transmit them to the Linux instance.

For general information about CP recording system services, see *z/VM CP Programming Services*, SC24-6179.

Features

With the z/VM recording device driver, you can read from several CP services and collect records.

In particular, the z/VM recording device driver supports:

- Reading records from the CP error logging service, *LOGREC.
- Reading records from the CP accounting service, *ACCOUNT.
- Reading records from the CP diagnostic service, *SYMPTOM.
- Automatic and explicit record collection (see "Starting and stopping record collection" on page 442).

What you should know about the z/VM recording device driver

You can read records from different recording services, one record at a time.

The z/VM recording device driver is a character device driver that is grouped under the IUCV category of device drivers (see "Device categories" on page 9). There is one device for each recording service. The device nodes are created for you. If the z/VM recording device driver is compiled as a separate module, the device nodes are created when the module is loaded.

z/VM recording device nodes

Each recording service has a fixed minor number and a name that corresponds to the name of the service.

Table 54 shows the mapping of names and minor numbers.

Table 54. Device names and minor numbers

z/VM recording service	Standard device name	Minor number
*LOGREC	logrec	0
*ACCOUNT	account	1
*SYMPTOM	symptom	2

The major device number for the z/VM recording device driver is assigned dynamically. Read the `dev` attribute of any one of the z/VM recording devices to find out the major number. The `dev` attribute is of the form <*major*>:<*minor*>.

Example: To read the `dev` attribute of the logrec device:

```
# cat /sys/class/vmlogrdr/logrec/dev
254:0
```

While vmlogrdr registers its driver and device structures with the iucv bus, it also must register a class and a class device under `/sys/class`. The `dev` attribute is member of that class device. In the example, the assigned major number is 254 and the minor number is 0, as expected.

Creating device nodes for the z/VM recording devices

You access z/VM recording data through device nodes. Unless udev or your distribution create the nodes for you, you must create them.

If there are no device nodes, use a command of this form to create a node:

```
mknod -m 440 /dev/<file> c <major> <minor>
```

where:

<file>
: is the file name that you assign to the device node.

<major>
: is the major number that was dynamically assigned to the z/VM recording device driver (see "z/VM recording device nodes" on page 439).

<minor>
: is the minor number of the recording service for which you are creating the device node.

Example

Using the standard device names (see Table 54 on page 439) and assuming that the assigned major number is 254, you could create the device nodes like this:

```
# mknod -m 440 /dev/logrec c 254 0
# mknod -m 440 /dev/account c 254 1
# mknod -m 440 /dev/symptom c 254 2
```

About records

Records for different services are different in details, but follow the same overall structure.

The read function returns one record at a time. If there is no record, the read function waits until a record becomes available.

Each record begins with a 4-byte field that contains the length of the remaining record. The remaining record contains the binary z/VM data followed by the four bytes X'454f5200' to mark the end of the record. These bytes build the zero-terminated ASCII string "EOR", which is useful as an eye catcher.

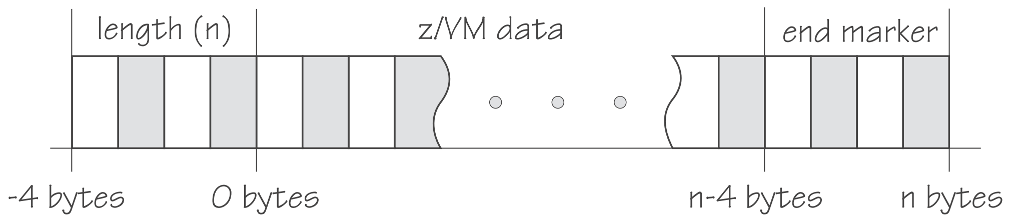


Figure 96. Record structure

Figure 96 illustrates the structure of a complete record as returned by the device. If the buffer assigned to the read function is smaller than the overall record size, multiple reads are required to obtain the complete record.

The format of the z/VM data (*LOGREC) depends on the record type that is described in the common header for error records HDRREC.

For more information about the z/VM record layout, see the *CMS and CP Data Areas and Control Blocks* documentation at www.ibm.com/vm/pubs/ctlblk.html.

Building a kernel with the z/VM recording device driver

For a Linux kernel that supports the z/VM recording device driver you need a kernel that includes the IUCV device driver. Also, select the CONFIG_VMLOGRDR configuration menu option.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

```
Device Drivers --->
   ...
   Character devices --->
      ...
      --- S/390 character device drivers (depends on S390) ---
      ...
      Support for the z/VM recording system services (VM only) (CONFIG_VMLOGRDR)
```

Figure 97. z/VM recording kernel configuration menu option

The z/VM recording device driver can be compiled into the kernel or as a separate module, vmlogrdr.

Setting up the z/VM recording device driver

Before you can collect records, you must authorize your z/VM guest virtual machine and, unless it was compiled into the kernel, load the device driver module.

About this task

This section provides information about the guest authorization that is required for collecting records and about how to load the device driver if it was compiled as a module.

Procedure

1. Authorize the z/VM guest virtual machine on which your Linux instance runs to:
   - Use the z/VM CP RECORDING command.
   - Connect to the IUCV services to be used: one or more of *LOGREC, *ACCOUNT, and *SYMPTOM.
2. If the z/VM recording device driver was compiled as a separate module, load the vmlogrdr module. Use the **modprobe** command to ensure that any other required modules are loaded in the correct order:

   ```
   # modprobe vmlogrdr
   ```

   There are no kernel or module parameters for the z/VM recording device driver.

Working with z/VM recording devices

Typical tasks that you perform with z/VM recording devices include starting and stopping record collection, purging records, and opening and closing devices.

- “Starting and stopping record collection”
- “Purging existing records” on page 443
- “Querying the z/VM recording status” on page 444
- “Opening and closing devices” on page 444

Starting and stopping record collection

By default, record collection for a particular z/VM recording service begins when the corresponding device is opened and stops when the device is closed.

About this task

You can use a device's `autorecording` attribute to be able to open and close a device without also starting or stopping record collection. You can use a device's `recording` attribute to start and stop record collection regardless of whether the device is opened or not.

You cannot start record collection if a device is open and records already exist. Before you can start record collection for an open device, you must read or purge any existing records for this device (see “Purging existing records” on page 443).

Procedure

To be able to open a device without starting record collection and to close a device without stopping record collection write `0` to the device’s `autorecording` attribute. To restore the automatic starting and stopping of record collection, write `1` to the device’s `autorecording` attribute. Issue a command of this form:

```
# echo <flag> > /sys/bus/iucv/drivers/vmlogrdr/<device>/autorecording
```

where <*flag*> is either 0 or 1, and <*device*> is one of: `logrec`, `symptom`, or `account`. To explicitly turn on record collection, write `1` to the device’s `recording` attribute. To explicitly turn off record collection, write `0` to the device’s `recording` attribute. Issue a command of this form:

```
# echo <flag> > /sys/bus/iucv/drivers/vmlogrdr/<device>/recording
```

where <*flag*> is either 0 or 1, and <*device*> is one of: logrec, symptom, or account. You can read both the `autorecording` and the `recording` attribute to find the current settings.

Examples

- In this example, first the current setting of the `autorecording` attribute of the logrec device is checked, then automatic recording is turned off:

  ```
  # cat /sys/bus/iucv/drivers/vmlogrdr/logrec/autorecording
  1
  # echo 0 > /sys/bus/iucv/drivers/vmlogrdr/logrec/autorecording
  ```

- In this example, record collection is started explicitly and later stopped for the account device:

  ```
  # echo 1 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
  ...
  # echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
  ```

  To confirm whether recording is on or off, read the `recording_status` attribute as described in "Querying the z/VM recording status" on page 444.

Purging existing records

By default, existing records for a particular z/VM recording service are purged automatically when the corresponding device is opened or closed.

About this task

You can use a device's `autopurge` attribute to prevent records from being purged when a device is opened or closed. You can use a device's `purge` attribute to purge records for a particular device at any time without having to open or close the device.

Procedure

To be able to open or close a device without purging existing records write `0` to the device's `autopurge` attribute. To restore automatic purging of existing records, write `1` to the device's `autopurge` attribute. You can read the `autopurge` attribute to find the current setting. Issue a command of this form:

```
# echo <flag> > /sys/bus/iucv/drivers/vmlogrdr/<device>/autopurge
```

where <*flag*> is either 0 or 1, and <*device*> is one of: `logrec`, `symptom`, or `account`. To purge existing records for a particular device without opening or closing the device write 1 to the device's purge attribute. Issue a command of this form:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/<device>/purge
```

where <*device*> is one of: `logrec`, `symptom`, or `account`.

Examples

- In this example, the setting of the autopurge attribute for the logrec device is checked first, then automatic purging is switched off:

```
# cat /sys/bus/iucv/drivers/vmlogrdr/logrec/autopurge
1
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/logrec/autopurge
```

- In this example, the existing records for the symptom device are purged:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/symptom/purge
```

Querying the z/VM recording status

Use the `recording_status` attribute of the z/VM recording device driver representation in sysfs to query the z/VM recording status.

Example

This example runs the z/VM CP command QUERY RECORDING and returns the complete output of that command. This list does not necessarily have an entry for all three services and there might also be entries for other guests.

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
```

This command results in output similar to the following example:

```
RECORDING     COUNT     LMT     USERID     COMMUNICATION
EREP ON       00000000  002     EREP       ACTIVE
ACCOUNT ON    00001774  020     DISKACNT   INACTIVE
SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
ACCOUNT OFF   00000000  020     LINUX31    INACTIVE
```

where the lines represent:
- The service
- The recording status
- The number of queued records
- The number of records that result in a message to the operator
- The guest that is or was connected to that service and the status of that connection

A detailed description of the QUERY RECORDING command can be found in the *z/VM CP Commands and Utilities Reference*, SC24-6175.

Opening and closing devices

You can open, read, and release the device. You cannot open the device multiple times. Each time the device is opened it must be released before it can be opened again.

About this task

You can use a device's `autorecord` attribute (see "Starting and stopping record collection" on page 442) to enable automatic record collection while a device is open.

You can use a device's `autopurge` attribute (see "Purging existing records" on page 443) to enable automatic purging of existing records when a device is opened and closed.

Scenario: Connecting to the *ACCOUNT service

A typical sequence of tasks is autorecording, turning autorecording off, purging records, and starting recording.

Procedure

1. Query the status of z/VM recording. As root, issue the following command:

   ```
   # cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
   ```

   The results depend on the system, and look similar to the following example:

   ```
   RECORDING     COUNT     LMT     USERID     COMMUNICATION
   EREP ON       00000000  002     EREP       ACTIVE
   ACCOUNT ON    00001812  020     DISKACNT   INACTIVE
   SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
   ACCOUNT OFF   00000000  020     LINUX31    INACTIVE
   ```

2. Open `/dev/account` with an appropriate application. This action connects the guest to the *ACCOUNT service and starts recording. The entry for *ACCOUNT on guest LINUX31 changes to ACTIVE and ON:

   ```
   # cat /sys/bus/iucv/drivers/vmlogrdr/recording_status

   RECORDING     COUNT     LMT     USERID     COMMUNICATION
   EREP ON       00000000  002     EREP       ACTIVE
   ACCOUNT ON    00001812  020     DISKACNT   INACTIVE
   SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
   ACCOUNT ON    00000000  020     LINUX31    ACTIVE
   ```

3. Switch autopurge and autorecord off:

   ```
   # echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/autopurge
   ```

   ```
   # echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/autorecording
   ```

4. Close the device by ending the application that reads from it and check the recording status. While the connection is INACTIVE, RECORDING is still ON:

   ```
   # cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
   RECORDING     COUNT     LMT     USERID     COMMUNICATION
   EREP ON       00000000  002     EREP       ACTIVE
   ACCOUNT ON    00001812  020     DISKACNT   INACTIVE
   SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
   ACCOUNT ON    00000000  020     LINUX31    INACTIVE
   ```

5. The next status check shows that some event created records on the *ACCOUNT queue:

   ```
   # cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
   RECORDING     COUNT     LMT     USERID     COMMUNICATION
   EREP ON       00000000  002     EREP       ACTIVE
   ACCOUNT ON    00001821  020     DISKACNT   INACTIVE
   SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
   ACCOUNT ON    00000009  020     LINUX31    INACTIVE
   ```

6. Switch recording off:

```
# echo 0 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
```

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING     COUNT     LMT     USERID     COMMUNICATION
EREP ON       000000000 002     EREP       ACTIVE
ACCOUNT ON    00001821  020     DISKACNT   INACTIVE
SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
ACCOUNT OFF   00000009  020     LINUX31    INACTIVE
```

7. Try to switch it on again, and check whether it worked by checking the recording status:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
```

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING     COUNT     LMT     USERID     COMMUNICATION
EREP ON       000000000 002     EREP       ACTIVE
ACCOUNT ON    00001821  020     DISKACNT   INACTIVE
SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
ACCOUNT OFF   00000009  020     LINUX31    INACTIVE
```

Recording did not start, in the message logs you might find a message:

```
 vmlogrdr: recording response: HCPCRC8087I Records are queued for user LINUX31 on the
*ACCOUNT recording queue and must be purged or retrieved before recording can be turned on.
```

This kernel message has priority 'debug' so it might not be written to any of your log files.

8. Now remove all the records on your *ACCOUNT queue either by starting an application that reads them from `/dev/account` or by explicitly purging them:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/account/purge
```

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING     COUNT     LMT     USERID     COMMUNICATION
EREP ON       00000000  002     EREP       ACTIVE
ACCOUNT ON    00001821  020     DISKACNT   INACTIVE
SYMPTOM ON    00000000  002     OPERSYMP   ACTIVE
ACCOUNT OFF   00000000  020     LINUX31    INACTIVE
```

9. Now start recording and check status again:

```
# echo 1 > /sys/bus/iucv/drivers/vmlogrdr/account/recording
```

```
# cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
RECORDING     COUNT      LMT     USERID     COMMUNICATION
EREP ON       00000000   002     EREP       ACTIVE
ACCOUNT ON    00001821   020     DISKACNT   INACTIVE
SYMPTOM ON    00000000   002     OPERSYMP   ACTIVE
ACCOUNT ON    00000000   020     LINUX31    INACTIVE
```

Chapter 36. z/VM unit record device driver

The z/VM unit record device driver provides Linux on z/VM with access to virtual unit record devices. Unit record devices comprise punch card readers, card punches, and line printers.

Linux access is limited to virtual unit record devices with default device types (2540 for reader and punch, 1403 for printer).

To write Linux files to the virtual punch or printer (that is, to the corresponding spool file queues) or to receive z/VM reader files (for example CONSOLE files) to Linux files, use the **vmur** command that is part of the s390-tools package (see "vmur - Work with z/VM spool file queues" on page 686).

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

When the vmur module is loaded, it registers a character device with dynamic major number and a minor number region from `0x0` to `0xffff`.

The minor number corresponds to the virtual device number of the virtual unit record device. A mechanism like udev is needed to automatically generate the respective device nodes. The default udev rules create the following device nodes:

- Reader: `/dev/vmrdr-0.0.<`*`device_number`*`>`
- Punch: `/dev/vmpun-0.0.<`*`device_number`*`>`
- Printer: `/dev/vmprt-0.0.<`*`device_number`*`>`

Building a kernel with the z/VM unit record device driver

Select the kernel configuration option CONFIG_S390_VMUR to compile a kernel that supports unit record devices.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

```
Device Drivers --->
   ...
   Character devices --->
      ...
      --- S/390 character device drivers (depends on S390) ---
      ...
      z/VM unit record device driver                          (CONFIG_S390_VMUR)
```

Figure 98. Kernel configuration menu option for the z/VM unit record device driver

The z/VM unit record device driver can be compiled into the kernel or as a separate module, vmur.

Working with z/VM unit record devices

Load the z/VM unit record device driver, if needed, then set the required virtual unit record devices online.

Procedure

1. If the z/VM unit record device driver was compiled as a separate module, load it with the **modprobe** command.

   ```
   # modprobe vmur
   ```

   There are no kernel or module parameters for the vmur device driver.
2. Set the devices that you want to work with online.

   For example, to set the devices with device bus-IDs 0.0.000c, 0.0.000d, and 0.0.000e online, issue the following command:

   ```
   # chccwdev -e 0.0.000c-0.0.000e
   ```

What to do next

You can now use the **vmur** command to work with the devices ("vmur - Work with z/VM spool file queues" on page 686).

If you want to unload the vmur module, close all unit record device nodes. Attempting to unload the module while a device node is open results in error message `Module vmur is in use`. You can unload the vmur module, for example, by issuing **modprobe -r**.

Serialization is implemented per device; only one process can open a particular device node at any one time.

Chapter 37. 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.

In particular, 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 read-write RAM disk that can be shared among multiple Linux instances that run as guests of the same z/VM system. For example, such a RAM disk can provide a shared file system.

For information about DCSS, see *z/VM Saved Segments Planning and Administration*, SC24-6229

For an example of how the xip option for the ext2 file system and DCSS can be used see *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594 on developerWorks at
www.ibm.com/developerworks/linux/linux390/documentation_dev.html

What you should know about DCSSs

The DCSS device names and nodes adhere to a naming scheme. There are different modes and options for mounting a DCSS.

Important: DCSSs occupy spool space. Be sure that you have enough spool space available (multiple times the DCSS size).

DCSS naming scheme

The standard device names are of the form dcssblk<*n*>, where <*n*> is the corresponding minor number.

The first DCSS device that is added is assigned the name dcssblk0, the second dcssblk1, and so on. When a DCSS device is removed, its device name and corresponding minor number are free and can be reassigned. A DCSS device that is added always receives the lowest free minor number.

Creating device nodes

User space programs access DCSS devices by device nodes. Your distribution might create these device nodes for you.

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

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

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

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

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

Accessing a DCSS in exclusive-writable mode

You must access a DCSS in exclusive-writable mode, for example, to create or update the DCSS.

To access a DCSS in exclusive-writable mode at least one of the following conditions must apply:

- The DCSS fits below the maximum definable address space size of the z/VM guest virtual machine.

  For large read-only DCSS, you can use suitable guest sizes to restrict exclusive-writable access to a specific z/VM guest virtual machine with a sufficient maximum definable address space size.
- The z/VM user directory entry for the z/VM guest virtual machine includes a NAMESAVE statement for the DCSS. See *z/VM CP Planning and Administration*, SC24-6178 for more information about the NAMESAVE statement.
- The DCSS was defined with the LOADNSHR operand.

  See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about the LOADNSHR operand.

  See "DCSS options" about saving DCSSs with the LOADNSHR operand or with other optional properties.

DCSS options

The z/VM DCSS device driver always saves DCSSs with default properties. Any previously defined options are removed.

For example, a DCSS that was defined with the LOADNSHR operand loses this property when it is saved with the z/VM DCSS device driver.

To save a DCSS with optional properties, you must unmount the DCSS device, then use the CP DEFSEG and SAVESEG commands to save the DCSS. See "Workaround for saving DCSSs with optional properties" on page 458 for an example.

See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about DCSS options.

Building a kernel with the DCSS device driver

To build a kernel with DCSS support, you must select option CONFIG_DCSSBLK in the configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

```
Device Drivers --->
   ...
   Block devices --->                         (common code option CONFIG_BLK_DEV)
      ...
      --- S/390 block device drivers (depends on S390 && BLOCK) ---
      ...
      DCSSBLK support                                            (CONFIG_DCSSBLK)
```

Figure 99. DCSS kernel configuration menu option

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

Setting up the DCSS device driver

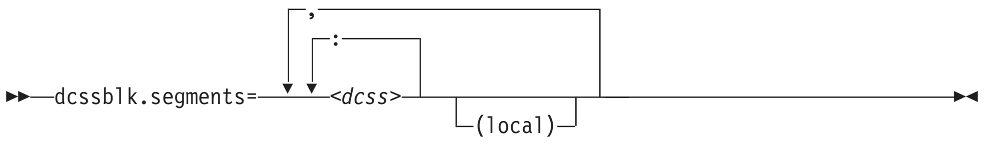
Configure the DCSS device driver through the `dcssblk.segments=` kernel or the `segments=` module parameter.

Kernel parameters

If the DCSS block device support has been compiled into the kernel, you configure the device driver by adding parameters to the kernel parameter line.

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

DCSS kernel parameter syntax



<dcss>
specifies the name of a DCSS as defined on the z/VM hypervisor. The specification for *<dcss>* is converted from ASCII to uppercase EBCDIC.

: the colon (:) separates DCSSs within a set of DCSSs to be mapped to a single DCSS device. You can map a set of DCSSs to a single DCSS device if the DCSSs in the set form a contiguous memory space.

You can specify the DCSSs in any order. The name of the first DCSS you specify is used to represent the device under `/sys/devices/dcssblk`.

(local)
sets the access mode to exclusive-writable after the DCSS or set of DCSSs are loaded.

, the comma (,) separates DCSS devices.

Examples

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

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

The following parameter in the kernel parameter line loads four DCSSs during the boot process: DCSS4, DCSS5, DCSS6, and DCSS7. The device driver creates two DCSS devices. One device maps to DCSS4. The other device maps to the combined storage space of DCSS5, DCSS6, and DCSS7 as a single device.

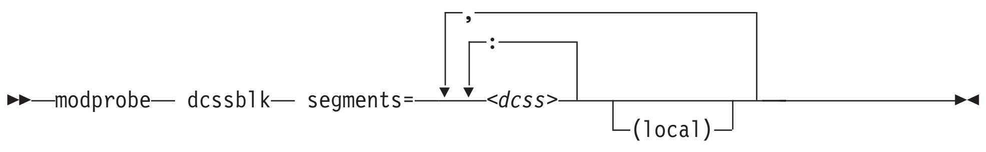
```
dcssblk.segments="dcss4,dcss5:dcss6:dcss7"
```

Module parameters

If the DCSS block device support has been compiled as a separate module, you load and configure the DCSS block device driver with **`modprobe`**.

Use the `segments` module parameter to load one or more DCSSs when the DCSS device driver is loaded.

DCSS module parameter syntax



<dcss>
: specifies the name of a DCSS as defined on the z/VM hypervisor. The specification for *<dcss>* is converted from ASCII to uppercase EBCDIC.

:
: the colon (:) separates DCSSs within a set of DCSSs to be mapped to a single DCSS device. You can map a set of DCSSs to a single DCSS device if the DCSSs in the set form a contiguous memory space.

  You can specify the DCSSs in any order. The name of the first DCSS you specify is used to represent the device under `/sys/devices/dcssblk`.

`(local)`
: sets the access mode to exclusive-writable after the DCSS or set of DCSSs are loaded.

,
: the comma (,) separates DCSS devices.

Examples

The following command loads the DCSS device driver and three DCSSs: DCSS1, DCSS2, and DCSS3. DCSS2 is accessed in exclusive-writable mode.

```
# modprobe dcssblk segments="dcss1,dcss2(local),dcss3"
```

The following command loads the DCSS device driver and four DCSSs: DCSS4, DCSS5, DCSS6, and DCSS7. The device driver creates two DCSS devices. One device maps to DCSS4. The other device maps to the combined storage space of DCSS5, DCSS6, and DCSS7 as a single device.

```
# modprobe dcssblk segments="dcss4,dcss5:dcss6:dcss7"
```

Avoiding overlaps with your guest storage

Ensure that your DCSSs do not overlap with the memory of your z/VM guest virtual machine (guest storage).

About this task

To find the start and end addresses of the DCSSs, enter the following CP command; this command requires privilege class E:

```
#cp q nss map
```

the output gives you the start and end addresses of all defined DCSSs in units of 4-kilobyte pages:

```
00: FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
...
00: 0011 MONDCSS  CPDCSS   N/A     09000  097FF  SC   R   00003 N/A      N/A
...
```

If all DCSSs that you intend to access are located above the guest storage, you do not need to take any action.

Procedure

If any DCSS that you intend to access with your guest machine overlaps with the guest storage, redefine the guest storage. Define two or more discontiguous storage extents such that the storage gap with the lowest address range covers the address ranges of all your DCSSs.

Note:
- You cannot place a DCSS into a storage gap other than the storage gap with the lowest address range.
- A z/VM guest that was defined with one or more storage gaps cannot access a DCSS above the guest storage.

From a CMS session, use the DEF STORE command to define your guest storage as discontiguous storage extents. Ensure that the storage gap between the extents covers all your DCSSs' address ranges. Issue a command of this form:

```
DEF STOR CONFIG 0.<storage_gap_begin> <storage_gap_end>.<storage above gap>
```

where:

<storage_gap_begin>
: is the lower limit of the storage gap. This limit must be at or below the lowest address of the DCSS with the lowest address range.

Because the lower address ranges are needed for memory management functions, make the lower limit at least 128 MB. The lower limit for the DCSS increases with the total memory size. Although 128 MB is not an exact value, it is an approximation that is sufficient for most cases.

`<storage_gap_end>`
: is the upper limit of the storage gap. The upper limit must be above the upper limit of the DCSS with the highest address range.

`<storage above gap>`
: is the amount of storage above the storage gap. The total guest storage is *<storage_gap_begin>* + *<storage above gap>*.

All values can be suffixed with M to provide the values in megabyte. See *z/VM CP Commands and Utilities Reference*, SC24-6175 for more information about the DEF STORE command.

Example

To make a DCSS that starts at 144 MB and ends at 152 MB accessible to a z/VM guest with 512 MB guest storage:

```
DEF STORE CONFIG 0.140M 160M.372M
```

This specification is one example of how a suitable storage gap can be defined. In this example, the storage gap covers 140 - 160 MB and, thus, the entire DCSS range. The total guest storage is 140 MB + 372 MB = 512 MB.

Working with DCSS devices

Typical tasks for working with DCSS devices include mapping DCSS representations in z/VM and Linux, adding and removing DCSSs, and accessing and updating DCSS contents.

- "Adding a DCSS device"
- "Listing the DCSSs that map to a particular device" on page 455
- "Finding the minor number for a DCSS device" on page 456
- "Setting the access mode" on page 456
- "Saving updates to a DCSS or set of DCSSs" on page 457
- "Workaround for saving DCSSs with optional properties" on page 458
- "Removing a DCSS device" on page 459

Adding a DCSS device

Storage gaps or overlapping storage ranges can prevent you from adding a DCSS.

Before you begin

- You must have set up one or more DCSSs on z/VM and know their names on z/VM.
- If you use the watchdog device driver, turn off the watchdog before adding a DCSS device. Adding a DCSS device can result in a watchdog timeout if the watchdog is active.
- You cannot concurrently access overlapping DCSSs.
- You cannot access a DCSS that overlaps with your guest virtual storage (see "Avoiding overlaps with your guest storage" on page 453).

- On z/VM guest virtual machines with one or more storage gaps, you cannot add a DCSS that is above the guest storage.
- On z/VM guest virtual machines with multiple storage gaps, you cannot add a DCSS unless it fits in the storage gap with the lowest address range.

Procedure

To add a DCSS device enter a command of this form:

```
# echo <dcss-list> > /sys/devices/dcssblk/add
```

<dcss-list>
: the name, as defined on z/VM, of a single DCSS or a colon (:) separated list of names of DCSSs to be mapped to a single DCSS device. You can map a set of DCSSs to a single DCSS device if the DCSSs in the set form a contiguous memory space. You can specify the DCSSs in any order. The name of the first DCSS you specify is used to represent the device under `/sys/devices/dcssblk`.

Examples

To add a DCSS called “MYDCSS” enter:

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

To add three contiguous DCSSs “MYDCSS1”, “MYDCSS2”, and “MYDCSS3” as a single device, enter:

```
# echo MYDCSS2:MYDCSS1:MYDCSS3 > /sys/devices/dcssblk/add
```

In sysfs, the resulting device is represented as `/sys/devices/dcssblk/MYDCSS2`.

Listing the DCSSs that map to a particular device

Read the `seglist` sysfs attribute to find out how DCSS devices in Linux map to DCSSs as defined in z/VM.

Procedure

To list the DCSSs that map to a DCSS device, issue a command of this form:

```
# cat /sys/devices/dcssblk/<dcss-name>/seglist
```

where *<dcss-name>* is the DCSS name that represents the DCSS device.

Examples

In this example, DCSS device `MYDCSS` maps to a single DCSS, “MYDCSS”.

```
# cat /sys/devices/dcssblk/MYDCSS/seglist
MYDCSS
```

In this example, DCSS device `MYDCSS2` maps to three contiguous DCSSs, “MYDCSS1”, “MYDCSS2”, and “MYDCSS3”.

```
# cat /sys/devices/dcssblk/MYDCSS2/seglist
MYDCSS2
MYDCSS1
MYDCSS3
```

Finding the minor number for a DCSS device

When you add a DCSS device, a minor number is assigned to it.

About this task

Unless you use dynamically created device nodes as provided by udev, you might need to know the minor device number that has been assigned to the DCSS (see "DCSS naming scheme" on page 449).

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.

About this task

There are two possible write access modes to the DCSS device:

shared
In the shared mode, changes to DCSSs are immediately visible to all z/VM guests that access them. Shared is the default.

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

exclusive-writable
In the exclusive-writable mode you write to private copies of DCSSs. A private copy is writable, even if the original DCSS is read-only. Changes that you make to a private copy are invisible to other guests until you save the changes (see "Saving updates to a DCSS or set of DCSSs" on page 457).

After saving the changes to a DCSS, all guests that open the DCSS access the changed copy. z/VM retains a copy of the original DCSS for those guests that continue accessing it, until the last guest stops using it.

To access a DCSS in the exclusive-writable mode, the maximum definable storage size of your z/VM virtual machine must be above the upper limit of the DCSS. Alternatively, suitable authorizations must be in place (see "Accessing a DCSS in exclusive-writable mode" on page 450).

For either access mode the changes are volatile until they are saved (see "Saving updates to a DCSS or set of DCSSs").

Procedure

Set the access mode before you open the DCSS device. To set the access mode to exclusive-writable, set the DCSS device's shared attribute to `0`. To reset the access mode to shared set the DCSS device's shared attribute to 1.

Issue a command of this form:

```
# echo <flag> > /sys/devices/dcssblk/<dcss-name>/shared
```

where <*dcss-name*> is the DCSS name that represents the DCSS device.
You can read the shared attribute to find out the current access mode.

Example

To find out the current access mode of a DCSS device represented by the DCSS name "MYDCSS":

```
# cat /sys/devices/dcssblk/MYDCSS/shared
1
```

`1` means that the current access mode is shared. To set the access mode to exclusive-writable, issue:

```
# echo 0 > /sys/devices/dcssblk/MYDCSS/shared
```

Saving updates to a DCSS or set of DCSSs

Use the `save` sysfs attribute to save DCSSs that were defined without optional properties.

Before you begin

- Saving a DCSS as described in this section results in a default DCSS, without optional properties. For DCSSs that were defined with options (see "DCSS options" on page 450), see "Workaround for saving DCSSs with optional properties" on page 458.
- If you use the watchdog device driver, turn off the watchdog before saving updates to DCSSs. Saving updates to DCSSs can result in a watchdog timeout if the watchdog is active.
- Do not place save requests before you have accessed the DCSS device.

Procedure

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.

Examples

To check whether a save request exists for a DCSS device that is represented by the DCSS name "MYDCSS":

```
# cat /sys/devices/dcssblk/MYDCSS/save
0
```

The 0 means that no save request exists. To place a save request issue:

```
# echo 1 > /sys/devices/dcssblk/MYDCSS/save
```

To purge an existing save request issue:

```
# echo 0 > /sys/devices/dcssblk/MYDCSS/save
```

Workaround for saving DCSSs with optional properties

If you need a DCSS that is defined with special options, you must use a workaround to save the DCSSs.

Before you begin

Important: This section applies to DCSSs with special options only. The workaround in this section is error-prone and requires utmost care. Erroneous parameter values for the described CP commands can render a DCSS unusable. Use this workaround only if you really need a DCSS with special options.

Procedure

Perform the following steps to save a DCSS with optional properties:

1. Unmount the DCSS.

   Example: Enter this command to unmount a DCSS with the device node /dev/dcssblk0:

   ```
   # umount /dev/dcssblk0
   ```

2. Use the CP DEFSEG command to newly define the DCSS with the required properties.

   Example: Enter this command to newly define a DCSS, `mydcss`, with the range `80000-9ffff`, segment type `sr`, and the `loadnshr` operand:

   ```
   # vmcp defseg mydcss 80000-9ffff sr loadnshr
   ```

   Note: If your DCSS device maps to multiple DCSSs as defined to z/VM, you must perform this step for each DCSS. Be sure to specify the command correctly with the correct address ranges and segment types. Incorrect specifications can render the DCSS unusable.
3. Use the CP SAVESEG command to save the DCSS.

   Example: Enter this command to save a DCSS `mydcss`:

   ```
   # vmcp saveseg mydcss
   ```

   Note: If your DCSS device maps to multiple DCSSs as defined to z/VM, you must perform this step for each DCSS. Omitting this step for individual DCSSs can render the DCSS device unusable.

Reference

See *z/VM CP Commands and Utilities Reference*, SC24-6175 for details about the DEFSEG and SAVESEG CP commands.

Removing a DCSS device

Use the `remove` sysfs attribute to remove a DCSS from Linux.

Before you begin

A DCSS device can be removed only when it is not in use.

Procedure

You can remove the DCSS or set of DCSSs that are represented by a DCSS device from your Linux system by issuing a command of this form:

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

where *<dcss-name>* is the DCSS name that represents the DCSS device.

Example

To remove a DCSS device that is represented by the DCSS name “MYDCSS” issue:

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

What to do next

If you have created your own device nodes, you can keep the nodes for reuse. Be aware that the major number of the device might change when you unload and reload the DCSS device driver. When the major number of your device has changed, existing nodes become unusable.

Scenario: Changing the contents of a DCSS

Before you change the contents of a DCSS, you must add the DCSS to Linux, access it in a writable mode, and mount the file system on it.

About this task

The scenario that follows is based on these assumptions:

- The Linux instance runs as a z/VM guest with class E user privileges.
- A DCSS was set up and can be accessed in exclusive-writable mode by the Linux instance.
- The DCSS does not overlap with the guest's main storage.
- There is only a single DCSS named “MYDCSS”.
- The DCSS block device driver is set up and ready to be used.

The description in this scenario can readily be extended to changing the content of a set of DCSSs that form a contiguous memory space. The only change to the procedure would be mapping the DCSSs in the set to a single DCSS device in step 1. The assumptions about the set of DCSSs would be:

- The contiguous memory space that is formed by the set does not overlap with the guest storage.
- Only the DCSSs in the set are added to the Linux instance.

Procedure

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

1. Add the DCSS to the block device driver.

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

2. Ensure that there is a device node for the DCSS block device. If it is not created for you, for example by udev, create it yourself.

   a. Find out the major number that is used for DCSS block devices. Read `/proc/devices`:

      ```
      # cat /proc/devices
      ...
      Block devices
      ...
      254 dcssblk
      ...
      ```

      The major number in the example is 254.

   b. Find out the minor number that is used for MYDCSS. If MYDCSS is the first DCSS to be added, the minor number is 0. To be sure, you can read a symbolic link that is created when the DCSS is added.

      ```
      # readlink /sys/devices/dcssblk/MYDCSS/block
      ../../../block/dcssblk0
      ```

The trailing 0 in the standard device name dcssblk0 indicates that the minor number is, indeed, 0.

c. Create the node with the **mknod** command:

```
# mknod /dev/dcssblk0 b 254 0
```

3. Set the access mode to exclusive-write.

```
# echo 0 > /sys/devices/dcssblk/MYDCSS/shared
```

4. Mount the file system in the DCSS on a spare mount point.

```
# mount /dev/dcssblk0 /mnt
```

5. Update the data in the DCSS.
6. Create a save request to save the changes.

```
# echo 1 > /sys/devices/dcssblk/MYDCSS/save
```

7. Unmount the file system.

```
# umount /mnt
```

The changes to the DCSS are now saved. When the last z/VM guest stops accessing the old version of the DCSS, the old version is discarded. Each guest that opens the DCSS accesses the updated copy.

8. Remove the device.

```
# echo MYDCSS > /sys/devices/dcssblk/remove
```

9. Optional: If you have created your own device node, you can clean it up.

```
# rm -f /dev/dcssblk0
```

Chapter 38. Shared kernel support

You can save a Linux kernel in a z/VM named saved system (NSS).

Through an NSS, z/VM makes operating system code in shared real memory pages available to z/VM guest virtual machines. Multiple instances of Linux on 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

To create an NSS, you require a Linux instance that supports kernel sharing and that is installed on a conventional boot device, for example, a DASD or SCSI disk.

You create the NSS when you use a special boot parameter to boot the Linux system from this original boot device.

For more information about NSS and the CP commands that are used in this section, see:

- *z/VM CP Commands and Utilities Reference*, SC24-6175.
- *z/VM Virtual Machine Operation*, SC24-6241.

Building a kernel with NSS support

You must select the option CONFIG_SHARED_KERNEL if you want to be able to IPL from an NSS.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

```
Virtualization --->
   Pseudo page fault support                                (CONFIG_PFAULT)
   VM shared kernel support                                 (CONFIG_SHARED_KERNEL)
```

Figure 100. NSS kernel configuration menu option

Note: You cannot enable shared kernel support unless the common code option `CONFIG_JUMP_LABEL` is deselected.

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 1-8 characters long and must consist of alphabetic or numeric characters. Be sure not to assign a name that matches any of the device numbers that are used at your installation.

Note: If <*nss_name*> contains non-alphanumeric characters, the NSS might be created successfully. However, this name might not work in CP commands. Always use alphanumeric characters for the name.

Working with a Linux NSS

You might have to create, update, or delete a Linux NSS.

Before you begin

For information about booting Linux from an NSS, see "Booting from a named saved system" on page 96.

Note: Kexec is disabled for Linux instances that are booted from a kernel NSS. As a consequence, you cannot use the kdump feature on such Linux instances.

For each task described in this section you need:
- A kernel image that provides shared kernel support and was installed on a conventional boot device.

  Important: Both compiling your own kernel or recompiling an existing distribution usually means that you must maintain your kernel yourself.
- A z/VM guest virtual machine that runs with class E privileges.

Creating or updating a Linux NSS by using zipl

You can use the **`zipl`** command to create an NSS or update an existing NSS.

Procedure

Perform these steps:
1. Boot the Linux instance from which you want to create the new NSS or the Linux instance from which you want to update an existing NSS.
2. Add `savesys=`<*nssname*> to the kernel parameters in your boot configuration, where <*nssname*> is the name for the new NSS to be created or of the existing NSS to be updated.

   For example, you can add the `savesys=` parameter to a kernel parameter file (see "Including kernel parameters in a boot configuration" on page 26 for details).

   The NSS name must be 1 - 8 alphanumeric characters, for example, `73248734`, `LNXNSS`, or `NSS1`. Be sure not to assign a name that matches a device number that is used at your installation.

3. Issue a **zipl** command to write the modified boot configuration to the boot device (Chapter 5, "Initial program loader for System z - zipl," on page 65).
4. Close down Linux.
5. Issue an IPL command to boot Linux from the device that holds the Linux kernel. During the IPL process, the NSS is created or updated and Linux is rebooted from the NSS.

Results

You can now use the NSS to boot Linux in your z/VM guest virtual machines. See "Booting from a named saved system" on page 96 for details.

Creating or updating a Linux NSS from the CP command line

You can create or update a Linux NSS without adding kernel parameters to the boot configuration and without running **zipl**.

Procedure

To boot Linux and save it as an NSS issue an IPL command of this form:

```
IPL <devno> PARM savesys=<nssname>
```

where:

<devno>
: specifies the CCW device that holds the Linux instance to be saved as an NSS.

<nssname>
: is the name for the new NSS to be created or the name of an existing NSS to be updated.

If your IPL device is an FCP device with an attached SCSI boot device, you cannot use the PARM parameter. Instead, use the z/VM CP LOADDEV command to specify the `savesys=` kernel parameter as SCPDATA (see "Booting from a SCSI device" on page 95).
The name must be 1 - 8 alphanumeric characters, for example, `73248734`, `LNXNSS`, or `NSS1`. Be sure not to assign a name that matches a device number that is used at your installation.
During the IPL process, the NSS is created and Linux is booted from the NSS.
For information about the PARM attribute, see "Specifying kernel parameters when booting Linux" on page 27.

Results

You can now use the NSS to boot Linux in your z/VM guest virtual machines. See "Booting from a named saved system" on page 96 for details.

Example

To create an NSS from a Linux instance that is installed on a device with bus ID `0.0.1234`, enter:

```
IPL 1234 PARM savesys=lnxnss
```

Deleting a Linux NSS

Issue a CP PURGE NSS NAME command to delete an NSS.

Procedure

Issue a command of this form:

```
PURGE NSS NAME <nssname>
```

where <*nssname*> is the name of the NSS you want to delete.

Results

The NSS is removed from storage when the last Linux instance that is already using it is closed down. You cannot IPL new Linux instances from the NSS anymore.

Chapter 39. z/VM CP interface device driver

Using the z/VM CP interface device driver (vmcp), you can send control program (CP) commands to the z/VM hypervisor and display the response.

The vmcp device driver works only for Linux on z/VM.

What you should know about the z/VM CP interface

The z/VM CP interface driver (vmcp) uses the CP diagnose X'08' to send commands to CP and to receive responses. The behavior is similar but not identical to #CP on a 3270 or 3215 console.

Using the z/VM CP interface

There are two ways of using the z/VM CP interface device driver:
- As a device node (usually `/dev/vmcp`)
- As a user space tool (see "vmcp - Send CP commands to the z/VM hypervisor" on page 684)

Differences between vmcp and a 3270 or 3215 console

Most CP commands behave identically with vmcp and on a 3270 or 3215 console. However, some commands show a different behavior:
- Diagnose X'08' (see *z/VM CP Programming Services*, SC24-6179) requires you to specify a response buffer with the command. Because the response size is not known in advance, the default response buffer of vmcp might be too small and the response truncated.
- On a 3270 or 3215 console, the CP command is executed on virtual CPU 0. The vmcp device driver uses the CPU that is scheduled by the Linux kernel. For CP commands that depend on the CPU number (like trace), specify the CPU, for example: `cpu 3 trace count`.
- Some CP commands do not return specific error or status messages through diagnose X'08'. These messages are returned only on a 3270 or 3215 console. For example, the command `vmcp link user1 1234 123 mw` might return the message `DASD 123 LINKED R/W` in a 3270 or 3215 console. This message is not displayed if the CP command is issued with vmcp. For details, see the z/VM help system or *z/VM CP Commands and Utilities Reference*, SC24-6175.

Creating device nodes

User space programs access vmcp devices through device nodes. Your distribution might create these device nodes for you.

If no device nodes are created for you, you need to create them yourself, for example, with the **mknod** command. See the **mknod** man page for further details.

The `/dev/vmcp` device node is a character device node (major number 10) with a dynamic minor number. During load, a sysfs folder,`class/misc/vmcp/`, is created. This folder contains the `dev` attribute for getting the major and minor number of vmcp.

Building a kernel with the z/VM CP interface

You need to select the kernel configuration option CONFIG_VMCP to build a kernel with user space access to CP commands.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

```
Device Drivers --->
   ...
   Character devices --->
      ...
      --- S/390 character device drivers (depends on S390) ---
      ...
      Support for the z/VM CP interface                            (CONFIG_VMCP)
```

Figure 101. Kernel configuration menu option for the z/VM CP interface

There are no kernel parameters for the vmcp device driver.

Using the device node

You can send a command to z/VM CP by writing to the vmcp device node.

Observe the following rules for writing to the device node:
- Omit the newline character at the end of the command string. For example, use **echo -n** if you are writing directly from a terminal session.
- Write the command in the same case as required on z/VM.
- Escape characters that need escaping in the environment where you issue the command.

Example

The following command attaches a device to your z/VM guest virtual machine. The asterisk (*) is escaped to prevent the command shell from interpreting it.

```
# echo -n ATTACH 1234 \* > /dev/vmcp
```

Application programmers

You can also use the vmcp device node directly from an application 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 55. The vmcp ioctls

Name	Code definition	Description
VMCP_GETCODE	_IOR (0x10, 1, int)	Queries the return code of z/VM.
VMCP_SETBUF	_IOW(0x10, 2, int)	Sets the buffer size (the device driver has a default of 4 KB; vmcp calls this ioctl to set it to 8 KB instead).
VMCP_GETSIZE	_IOR(0x10, 3, int)	Queries the size of the response.

Chapter 40. z/VM CP special messages uevent support

The smsgiucv_app kernel device driver receives z/VM CP special messages (SMSG) and delivers these messages to user space as udev events (uevents).

The device driver receives only messages that start with APP. The generated uevents contain the message sender and content as environment variables (see Figure 102).

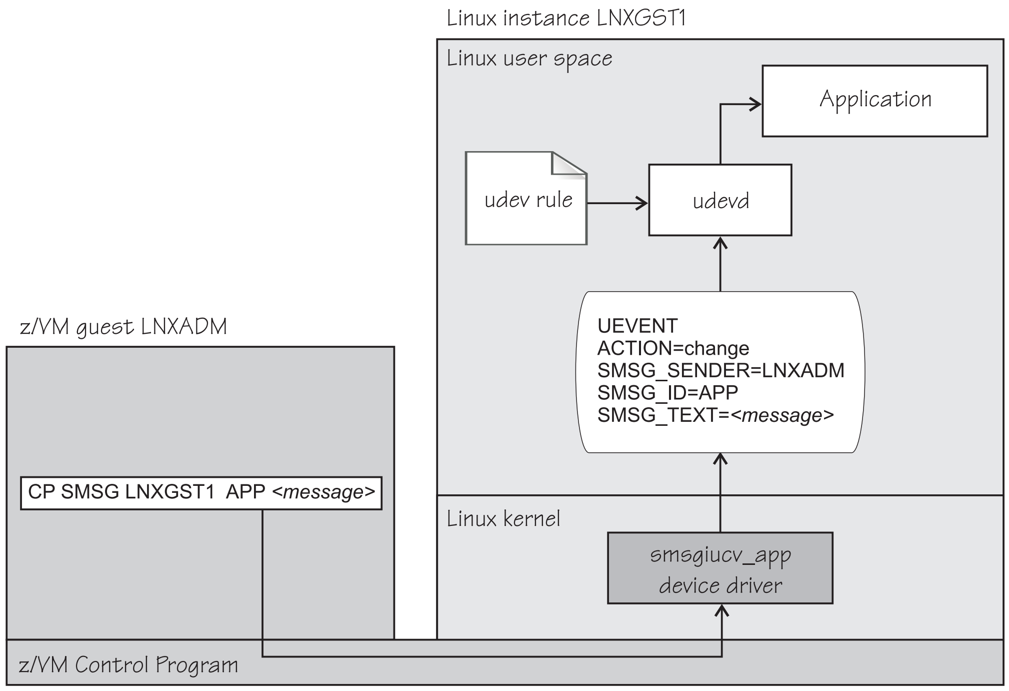


Figure 102. CP special messages as uevents in user space

You can restrict the received special messages to a particular z/VM user ID. CP special messages are discarded if the specified sender does not match the sender of the CP special message.

Building a kernel with the CP special message device driver

Select the kernel configuration option CONFIG_SMSGIUCV_EVENT to compile a kernel with support for CP special messages.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Figure 103 on page 470 summarizes the kernel configuration menu options that are relevant to the smsgiucv_app device driver:

```
Device Drivers --->
   ...
   Network device support --->                          (common code option CONFIG_NETDEVICES)
     ...
     S/390 network device drivers (depends on NETDEVICES && S390) --->
        ...
        IUCV special message support (VM only)                        (CONFIG_SMSGIUCV)*
        └─ Deliver IUCV special messages as uevents (VM only)         (CONFIG_SMSGIUCV_EVENT)
```

Figure 103. Special message kernel configuration menu options

CONFIG_SMSGIUCV
Select this option if you want to be able to receive SMSG messages from other z/VM guest virtual machines. Depends on IUCV.

CONFIG_SMSGIUCV_EVENT
Select this option to deliver CP special messages as uevents. The driver handles only those special messages that start with APP. To compile as a module, choose M. The module name is smsgiucv_app. Depends on SMSGIUCV.

Setting up the CP special message device driver

Configure the CP special message device driver through the `smsgiucv_app.sender=` kernel parameter or through the `sender=` module parameter.

The z/VM user ID does not require special authorizations to receive CP special messages. CP special messages can be issued from the local z/VM guest virtual machine or from other guest virtual machines. You can issue special messages from Linux or from a CMS or CP session.

See the Chapter 3, “Kernel and module parameters,” on page 25 chapter for more details about specifying kernel and module parameters.

Kernel parameters

If the CP special message device driver has been compiled into the kernel, you configure the device driver by adding the `smsgiucv_app.sender=` parameter to the kernel parameter line.

smsgiucv_app kernel parameter syntax

```
▶▶──┬────────────────────────────────┬──────────────────────────────▶◀
    └─smsgiucv_app.sender=<user_ID>──┘
```

where:

`smsgiucv_app.sender=``<`*`user_ID`*`>`
permits CP special messages from the specified z/VM user ID only. CP special messages are discarded if the specified sender does not match the sender of the CP special message. If the **`smsgiucv_app.sender=`** option is empty or not set, CP special messages are accepted from any z/VM user ID.

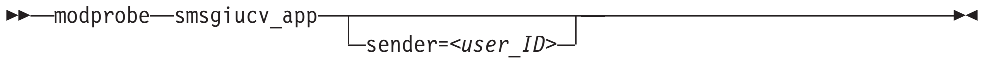
Lowercase characters are converted to uppercase.

To receive messages from several user IDs leave the `smsgiucv_app.sender=` parameter empty, or do not specify it, and then filter with udev rules (see "Example udev rule" on page 473).

Module parameters

If the CP special message device driver has been built as a separate module, you configure the device driver through the `sender=` module parameters when you load the module.

smsgiucv_app syntax



Where:

`sender` = *<user_ID>*
: permits CP special messages from the specified z/VM user ID only. CP special messages are discarded if the specified sender does not match the sender of the CP special message. If the **`sender`** option is empty or not set, CP special messages are accepted from any z/VM user ID.

  Lowercase characters are converted to uppercase.

To receive messages from several user IDs leave the `sender=` parameter empty, or do not specify it, and then filter with udev rules (see "Example udev rule" on page 473).

Working with CP special messages

You might have to send, access, or respond to CP special messages.
- "Sending CP special messages"
- "Accessing CP special messages through uevent environment variables" on page 472
- "Writing udev rules for handling CP special messages" on page 472

Sending CP special messages

Issue a CP SMSG command from a CP or CMS session or from Linux to send a CP special message.

Procedure

To send a CP special message to LXGUEST1 from Linux, enter a command of the following form:

```
# vmcp SMSG LXGUEST1 APP "<message text>"
```

To send a CP special message to LXGUEST1, enter the following command from a CP or CMS session:

```
#CP SMSG LXGUEST1 APP <message text>
```

The special messages cause uevents to be generated. See "Writing udev rules for handling CP special messages" for information about handling the uevents.

Accessing CP special messages through uevent environment variables

A uevent for a CP special message contains environment variables that you can use to access the message.

SMSG_ID
: Specifies the message prefix. The SMSG_ID environment variable is always set to APP, which is the prefix that is assigned to the smsgiucv_app device driver.

SMSG_SENDER
: Specifies the z/VM user ID that sent the CP special message.

  Use SMSG_SENDER in udev rules for filtering the z/VM user ID if you want to accept CP special messages from different senders. All alphabetic characters in the z/VM user ID are uppercase characters.

SMSG_TEXT
: Contains the message text of the CP special message. The APP prefix and leading white spaces are removed.

Writing udev rules for handling CP special messages

When using the CP special messages device driver, CP special messages trigger uevents.

change events
: The smsgiucv_app device driver generates `change` uevents for each CP special message that is received.

  For example, the special message:

```
#CP SMSG LXGUEST1 APP THIS IS A TEST MESSAGE
```

might trigger the following uevent:

```
UEVENT[1263487666.708881] change /devices/iucv/smsgiucv_app (iucv)
ACTION=change
DEVPATH=/devices/iucv/smsgiucv_app
SUBSYSTEM=iucv
SMSG_SENDER=MAINT
SMSG_ID=APP
SMSG_TEXT=THIS IS A TEST MESSAGE
DRIVER=SMSGIUCV
SEQNUM=1493
```

add and remove events
: In addition to the `change` event for received CP special messages, generic `add` and `remove` events are generated when the module is loaded or unloaded, for example:

```
UEVENT[1263487583.511146] add /module/smsgiucv_app (module)
ACTION=add
DEVPATH=/module/smsgiucv_app
SUBSYSTEM=module
SEQNUM=1487

UEVENT[1263487583.514622] add /devices/iucv/smsgiucv_app (iucv)
ACTION=add
DEVPATH=/devices/iucv/smsgiucv_app
SUBSYSTEM=iucv
DRIVER=SMSGIUCV
SEQNUM=1488

UEVENT[1263487628.955149] remove /devices/iucv/smsgiucv_app (iucv)
ACTION=remove
DEVPATH=/devices/iucv/smsgiucv_app
SUBSYSTEM=iucv
SEQNUM=1489

UEVENT[1263487628.957082] remove /module/smsgiucv_app (module)
ACTION=remove
DEVPATH=/module/smsgiucv_app
SUBSYSTEM=module
SEQNUM=1490
```

With the information from the uevents, you can create custom udev rules to trigger actions that depend on the settings of the SMSG_* environment variables (see "Accessing CP special messages through uevent environment variables" on page 472).

In your udev rules, use the `add` and `remove` uevents to initialize and clean up resources. To handle CP special messages, write udev rules that match `change` uevents. For more information about writing udev rules, see the udev man page.

Example udev rule

The udev rules that process CP special messages identify particular messages and define one or more specific actions as a response.

The following example shows how to process CP special messages by using udev rules. The example contains rules for actions, one for all senders and one for the MAINT, OPERATOR, and LNXADM senders only.

The rules are contained in a block that matches uevents from the smsgiucv_app device driver. If there is no match, processing ends:

```
#
# Sample udev rules for processing CP special messages.
#
#
DEVPATH!="*/smsgiucv_app", GOTO="smsgiucv_app_end"

# ---------- Rules for CP messages go here --------

LABEL="smsgiucv_app_end"
```

The example uses the **`vmur`** command. If the vmur kernel module has been compiled as a separate module, this module must be loaded first. Then, the z/VM virtual punch device is activated.

```
# --- Initialization ---

# load vmur and set the virtual punch device online
SUBSYSTEM=="module", ACTION=="add", RUN+="/sbin/modprobe --quiet vmur"
SUBSYSTEM=="module", ACTION=="add", RUN+="/sbin/chccwdev -e d"
```

The following rule accepts messages from all senders. The message text must match the string UNAME. If it does, the output of the **uname** command (the node name and kernel version of the Linux instance) is sent back to the sender.

```
# --- Rules for all senders ----

# UNAME: tell the sender which kernel is running
ACTION=="change", ENV{SMSG_TEXT}=="UNAME", \
    PROGRAM=="/bin/uname -n -r", \
    RUN+="/sbin/vmcp msg $env{SMSG_SENDER} '$result'"
```

In the following example block rules are defined to accept messages from certain senders only. If no sender matches, processing ends. The message text must match the string DMESG. If it does, the environment variable PATH is set and the output of the **dmesg** command is sent into the z/VM reader of the sender. The name of the spool file is LINUX DMESG.

```
# --- Special rules available for particular z/VM user IDs ---

ENV{SMSG_SENDER}!="MAINT|OPERATOR|LNXADM", GOTO="smsgiucv_app_end"

# DMESG: punch dmesg output to sender
ACTION=="change", ENV{SMSG_TEXT}=="DMESG", \
    ENV{PATH}="/bin:/sbin:/usr/bin:/usr/sbin", \
    RUN+="/bin/sh -c 'dmesg |fold -s -w 74 |vmur punch -r -t -N LINUX.DMESG -u $env{SMSG_SENDER}'"
```

Chapter 41. Cooperative memory management

Cooperative memory management (CMM, or "cmm1") can reduce the memory that is available to an instance of Linux on z/VM.

CMM allocates pages to page pools that are not available to Linux. A diagnose code indicates to z/VM that the pages in the page pools are out of use. z/VM can then immediately reuse these pages for other z/VM guests.

To set up CMM, you must perform these tasks:

1. Incorporate cmm, either by building a kernel that includes it, or by loading the cmm module.
2. Set up a resource management tool that controls the page pools. This tool can be the z/VM resource monitor (VMRM) or a third-party systems management tool.

This chapter describes how to set up CMM. For background information about CMM, see “Cooperative memory management background” on page 415.

You can also use the **cpuplugd** command to define rules for cmm behavior, see “cpuplugd - Control CPUs and memory” on page 566.

For information about setting up the external resource manager, see the chapter on VMRM in *z/VM Performance*, SC24-6208.

Building a kernel with cooperative memory management

To build a kernel with support for cooperative memory management, you must select option CONFIG_CMM in the configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Figure 104 on page 476 shows option CONFIG_CMM in the configuration menu. It also shows the options that you need in addition if you are using VMRM.

```
Virtualization --->
   ...
   Cooperative memory management                                                (CONFIG_CMM)
   └─ IUCV special message interface to cooperative memory management           (CONFIG_CMM_IUCV)
   ...
Networking support --->                                    (common code option CONFIG_NET)
   ...
   Networking options --->
      ...
      IUCV support (S390 - z/VM only)                                           (CONFIG_IUCV)
Device Drivers --->
   ...
   Network device support --->                             (common code option CONFIG_NETDEVICES)
      ...
      S/390 network device drivers (depends on NETDEVICES && S390) --->
         ...
         IUCV special message support (VM only)                                 (CONFIG_SMSGIUCV)
```

Figure 104. CMM and related kernel configuration menu options

CONFIG_CMM
This option includes the cooperative memory management support, which is available as a separate module, cmm, or built into the kernel.

CONFIG_CMM_IUCV
This option is required if you are using VMRM.

CONFIG_IUCV
This option is required for CONFIG_SMSGIUCV.

CONFIG_SMSGIUCV
This option is required for CONFIG_CMM_IUCV.

Setting up cooperative memory management

Configure Linux on z/VM to participate in cooperative memory management through the `cmm.sender=` kernel parameter or through the `sender=` module parameter.

Kernel parameters

If the cooperative memory management support has been compiled into the kernel, you configure Linux on z/VM by adding the cmm.sender= parameter to the kernel parameter line.

Cooperative memory management kernel parameter syntax

```
   ┌─cmm.sender=VMRMSVM───┐
►►─┤                      ├──────────────────────────────────────────────►◄
   └─cmm.sender=<user_ID>─┘
```

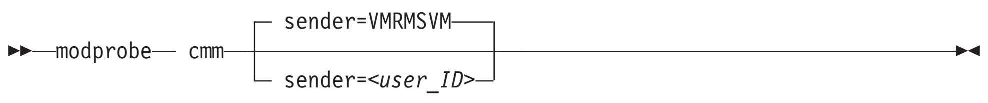
where *<user_ID>* specifies the z/VM guest virtual machine that is permitted to send messages to the module through the special messages interface. The default z/VM user ID is VMRMSVM, which is the default for the VMRM service machine.

Lowercase characters are converted to uppercase.

Loading the cooperative memory management module

If the cooperative memory management support has been compiled as a separate module, you configure Linux on z/VM through the `sender=` module parameter when you load the module with **`modprobe`**.

cooperative memory management module parameter syntax



where <*user_ID*> specifies the z/VM guest virtual machine that is permitted to send messages to the module through the special messages interface. The default z/VM user ID is VMRMSVM, which is the default for the VMRM service machine.

Lowercase characters are converted to uppercase.

Example

To load the cooperative memory management module and allow the z/VM guest virtual machine TESTID to send messages:

```
# modprobe cmm sender=TESTID
```

Working with cooperative memory management

After it has been set up, CMM works through the resource manager. No further actions are necessary. You might want to read the sizes of the page pools for diagnostic purposes.

To reduce the Linux memory size, CMM allocates pages to page pools that make the pages unusable to Linux. There are two such page pools, a static pool and a timed pool. You can use the procfs interface to read the sizes of the page pools.

Reading the size of the static page pool

You can read the current size of the static page pool from procfs.

Procedure

Issue this command:

```
# cat /proc/sys/vm/cmm_pages
```

Reading the size of the timed page pool

You can read the current size of the timed page pool from procfs.

Procedure

Issue this command:

```
# cat /proc/sys/vm/cmm_timed_pages
```

Part 7. Security

These device drivers and features support security aspects of Linux on System z.

Newest version

You can find the newest version of this publication at
www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Restrictions

For prerequisites and restrictions see
www.ibm.com/developerworks/linux/linux390/development_technical.html
www.ibm.com/developerworks/linux/linux390/development_restrictions.html

Chapter 42. Generic cryptographic device driver

The generic cryptographic device driver (zcrypt) supports cryptographic coprocessor and accelerator hardware. Cryptographic coprocessors provide secure key cryptographic operations for the IBM Common Cryptographic Architecture (CCA) and the Enterprise PKCS#11 feature (EP11).

Some cryptographic processing in Linux can be offloaded from the processor and performed by CCA or EP11 coprocessors or accelerators. Several of these CCA or EP11 coprocessors and accelerators are available offering a range of features. The generic cryptographic device driver (zcrypt) is required to use any available cryptographic hardware.

Features

The cryptographic device driver supports a range of hardware and software functions.

Supported cryptographic adapters

The cryptographic hardware feature might contain one or two cryptographic adapters. Each adapter can be configured either as a CCA coprocessor or as an accelerator.

The following types of cryptographic adapters are supported:
- PCI Cryptographic Coprocessor (PCICC)
- PCI Cryptographic Accelerator (PCICA)
- PCI-X Cryptographic Coprocessor (PCIXCC)
- Crypto Express2 Coprocessor (CEX2C)
- Crypto Express2 Accelerator (CEX2A)
- Crypto Express3 Coprocessor (CEX3C)
- Crypto Express3 Accelerator (CEX3A)
- Crypto Express4S (CCA) Coprocessor (CEX4C)
- Crypto Express4S Accelerator (CEX4A)
- Crypto Express4S (EP11) Coprocessor (CEX4P)

PCIXCC coprocessors are further distinguished by their Licensed Internal Code (LIC) level:
- PCIXCC (MCL3) as of LIC EC J12220 level 29
- PCIXCC (MCL2) with a LIC before EC J12220 level 29

For information about setting up your cryptographic environment on Linux under z/VM, see *Security on z/VM*, SG24-7471 and *Security for Linux on System z*, SG24-7728.

Cryptographic devices for Linux on z/VM

A z/VM guest virtual machine can either have one or more dedicated cryptographic devices or one shared cryptographic device, but not both.

Dedicated devices
Each dedicated device maps to exactly one hardware device. The device representations in Linux on z/VM show the type of the actual hardware.

Shared device
The shared device can map to one or more hardware devices. The device representation in Linux on z/VM shows the type of the most advanced of these hardware devices. In this representation, cryptographic accelerators are considered more advanced than CCA coprocessors.

As a consequence, Linux on z/VM with access to a shared cryptographic accelerator can either observe an accelerator or a CCA coprocessor, but not both.

When cryptographic coprocessors are shared, only the clear-key functions are available to the Linux instance.

Supported facilities

The cryptographic device driver supports several cryptographic accelerators and CCA coprocessors.

Cryptographic accelerators support clear key cryptographic algorithms. In particular, they provide fast RSA encryption and decryption for key sizes 1024-bit, 2048-bit, and 4096-bit (CEX4A and CEX3A only).

Cryptographic coprocessors act as a hardware security module (HSM) and provide secure key cryptographic operations for the IBM Common Cryptographic Architecture (CCA) and the Enterprise PKCS#11 feature (EP11).

For more information about CCA, see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294. You can obtain this publication at www.ibm.com/security/cryptocards/pciecc/library.shtml.

For more information about EP11, see *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713. You can obtain this publication at www.ibm.com/developerworks/linux/linux390/documentation_dev.html.

Cryptographic CCA coprocessors also provide clear key RSA operations for 1024-bit, 2048-bit, and 4096-bit keys, and a true random number generator. The EP11 coprocessor supports only secure key operations.

Hardware and software prerequisites

Support for the Crypto Express4S, Crypto Express3, and Crypto Express2 features depends on the System z hardware.

Table 56 lists the support for the cryptographic adapters.

Table 56. Support for cryptographic adapters by mainframe model

Cryptographic adapters	Mainframe support
CEX4A and CEX4C	• zEC12 • zBC12

Table 56. Support for cryptographic adapters by mainframe model (continued)

Cryptographic adapters	Mainframe support
CEX3A and CEX3C	• zEC12 • zBC12 • z196 • z114 • System z10 (as of October 2009)
CEX2A and CEX2C	System z10 and System z9

Table 57 lists the required software by function.

Table 57. Required software

Software required	Function that is supported by the software
The CCA library	For the secure key cryptographic functions on CEX4C, CEX3C, or CEX2C features. For information about CEX4C, CEX3C, and CEX2C adapter coexistence and how to use CCA functions, see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294. You can obtain it at www.ibm.com/security/cryptocards/pciecc/library.shtml.
The EP11 library	For the secure key cryptographic functions on CEX4P features. See *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713. You can obtain it at www.ibm.com/developerworks/linux/linux390/documentation_dev.html.
The libica library	For the clear key cryptographic functions. See *libica Programmer's Reference*, SC34-2602. You can obtain it at www.ibm.com/developerworks/linux/linux390/documentation_dev.html.
APAR VM65007	To support CEX4A and CEX4C adapters on z/VM 5.4, 6.1, and 6.2.
APAR VM65308	To share CEX4C CCA coprocessor adapters (APVIRT) on z/VM 5.4, 6.1, and 6.2.
APAR VM64656	To support CEX3C and CEX3A adapters for Linux on z/VM 6.1 or 5.4.
APAR VM64727	To correct a shared CCA coprocessor problem on z/VM 5.4.
APAR VM64793	To use the protected key functionality under z/VM and CCA on z/VM 5.4 and 6.1.

What you should know about the cryptographic device driver

Your use of the cryptographic device driver and the cryptographic hardware might require additional software. There are special considerations for Linux on z/VM, for performance, and for specific cryptographic operations.

Functions provided by the cryptographic device driver

The functions that are provided by the cryptographic device driver depend on whether the device driver finds an accelerator or CCA coprocessor.

If the cryptographic device driver finds a cryptographic accelerator, it provides Rivest-Shamir-Adleman (RSA) encryption and RSA decryption functions using clear keys. RSA operations are supported in both the modulus-exponent and the Chinese-Remainder Theorem (CRT) variants using 1024-bit, 2048-bit, and 4096-bit size keys.

If the cryptographic device driver finds a CCA coprocessor, it provides RSA encryption and RSA decryption functions using clear keys. RSA operations are supported in both the modulus-exponent and the CRT variants using 1024-bit, 2048-bit, and 4096-bit size keys. It also provides a function to pass CCA requests to the cryptographic coprocessor and an access to the true random number generator of the CCA coprocessor.

32-bit systems do not support 4096-bit key length for clear-key RSA operations.

Adapter discovery

The cryptographic device driver provides two misc device nodes, one for cryptographic requests, and one for a device from which random numbers can be read.

Cryptographic adapters are detected automatically when the module is loaded. They are reprobed periodically, and following any hardware problem.

Upon detection of a cryptographic adapter, the device driver presents a Linux misc device, z90crypt, to user space. A user space process can open the misc device to submit cryptographic requests to the adapter through IOCTLs.

If at least one of the detected cryptographic adapters is a CCA coprocessor, an additional misc device, `hwrng`, is created from which random numbers can be read.

You can set cryptographic adapters online or offline in the device driver. The cryptographic device driver ignores adapters that are configured offline even if the hardware is detected. The online or offline configuration is independent of the hardware configuration.

Request processing

Cryptographic adapters process requests asynchronously.

The device driver detects request completion either by standard polling, a special high-frequency polling thread, or by hardware interrupts. Hardware interrupt support is only available for Linux instances that run in an LPAR of a System z10 or later mainframe. If hardware interrupt support is available, the device driver does not use polling to detect request completion.

All requests to either of the two misc devices are routed to a cryptographic adapter using a crypto request scheduling function that, for each adapter, takes into account:

- The supported functions
- The number of pending requests
- A speed rating

A cryptographic adapter can be partitioned into multiple domains. Each domain acts as an independent virtual HSM that maintains its own master key. The cryptographic device driver uses only a single domain for all adapters. By default the kernel selects a domain. Alternatively, you can select the domain using a module parameter (see “Module parameters” on page 486).

Building a kernel with the cryptographic device driver

Control the build options for the cryptographic device driver through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Select the option CONFIG_ZCRYPT to include the cryptographic device driver.

```
Cryptographic API --->
   ...
   Hardware crypto devices  --->
      Support for PCI-attached cryptographic adapters (CONFIG_ZCRYPT)
```

Figure 105. zcrypt kernel configuration menu option

You can compile the device driver into the kernel or as multiple, separate modules:

ap AP bus module.

zcrypt_api
request router module. Loads the `rng_core` module.

zcrypt_cex4
device driver for CEX4A, CEX4C, and CEX4P adapters.

zcrypt_cex2a
device driver for CEX2A, and CEX3A adapters.

zcrypt_pcica
device driver for PCICA adapters.

zcrypt_pcicc
device driver for PCICC adapters.

zcrypt_pcixcc
device driver for PCIXCC, CEX2C, and CEX3C adapters.

zcrypt_msgtype6
secure key message module.

zcrypt_msgtype50
clear key message module.

Setting up the cryptographic device driver

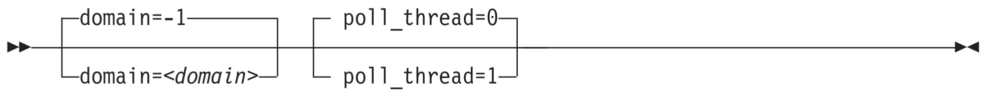
Configure the cryptographic device driver through the `domain=` and the `poll_thread=` kernel or module parameters. You might also have to set up libraries and create a device node.

For information about setting up cryptographic hardware on your mainframe, see *zSeries Crypto Guide Update*, SG24-6870.

Kernel parameters

If the zcrypt device driver was compiled into the kernel, you configure the device driver by adding parameters to the kernel parameter line.

zcrypt kernel parameter syntax



where

<domain>
: is an integer in the range 0 - 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 must specify the domain parameter only if you are running Linux in an LPAR for which multiple cryptographic domains were defined.

<poll_thread>
: is an integer argument and enables a polling thread to tune cryptographic performance. Valid values are 1 (enabled) or 0 (disabled, this value is the default). For details, see "Setting the polling thread" on page 491.

  Note: If you are running Linux in an LPAR on a z10™ EC or later, AP interrupts are used instead of the polling thread. The polling thread is disabled when AP interrupts are available. See "Using AP adapter interrupts" on page 492.

Example

The following kernel parameter line specification makes the zcrypt device driver operate within the cryptographic domain "7" with `poll_thread` enabled:

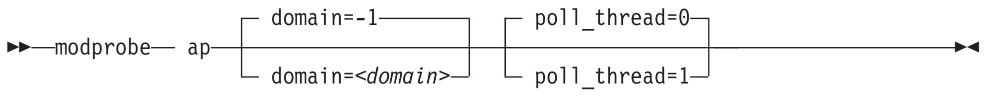
```
domain=7 poll_thread=1
```

Module parameters

If the zcrypt device driver was built as multiple, separate modules, you configure the device driver through module parameters when you load the AP bus module.

To load the AP bus module:

ap module syntax



where

<domain>
: is an integer in the range 0 - 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 must specify the domain parameter only if you are running Linux in an LPAR for which multiple cryptographic domains were defined.

<poll_thread>
: is an integer argument and enables a polling thread to tune cryptographic performance. Valid values are 1 (enabled) or 0 (disabled, this value is the default). For details, see "Setting the polling thread" on page 491.

  Note: If you are running Linux in an LPAR on a z10 EC or later, AP interrupts are used instead of the polling thread. The polling thread is disabled when AP interrupts are available. See "Using AP adapter interrupts" on page 492.

All other modules are loaded automatically when they are required.

To remove a single module, for example, a module that supports a card type that is no longer available, issue a command of the following form:

```
# rmmod <module name>
```

For example, if you no longer need the PCICC module, use:

```
# rmmod zcrypt_pcicc
```

Examples

- This example loads the discrete cryptographic device driver module `ap` if Linux runs with only one cryptographic domain:

  ```
  # modprobe ap
  ```

  Note: Only one cryptographic domain is supported per LPAR or z/VM.
- This example loads the discrete cryptographic device driver module `ap` to operate within the cryptographic domain 1:

  ```
  # modprobe ap domain=1
  ```

Assuring that you have a device node

User-space programs access cryptographic functions through a single device node.

Both the major and minor number can be dynamic, depending on your Linux distribution and configuration.

The major device number is then that of the misc devices. You can find it as the value for the entry "misc" in `/proc/devices`.

The minor number is dynamically assigned and you can find it in `/proc/misc` as the value for the entry "z90crypt".

If the device node /dev/z90crypt is not created for you, you can create it yourself by issuing a command of this form:

```
# mknod /dev/z90crypt c <misc_major> <dynamic_minor>
```

Accessing long random numbers

Applications can access large amounts of random number data through a character device.

Prerequisites:

- At least one cryptographic feature must be installed in the system and one CCA coprocessor, PCIXCC, CEX2C, CEX3C, or CEX4C, must be configured.

  PCIXCC only: The CCA code version that is installed on the feature must be version 3.30 or later.
- Linux on z/VM needs a dedicated CCA coprocessor or a shared cryptographic device that is backed only by CCA coprocessors.
- Automatic creation of the random number character device requires udev.
- The cryptographic device driver zcrypt must be loaded.

If zcrypt detects at least one CCA coprocessor capable of generating long random numbers, a new miscellaneous character device is registered. The new device can be found under /proc/misc as hw_random. If udev is installed, the default rules provided with udev create a character device called /dev/hwrng and a symbolic link called /dev/hw_random and pointing to /dev/hwrng.

If udev is not installed, you must create a device node:

1. You require the minor number of the hardware random number generator. Read this number from /proc/misc where it is registered as hw_random, for example:

   ```
   # grep hw_random /proc/misc
   183 hw_random
   ```

2. Create the device node by issuing a command of this form:

   ```
   # mknod /dev/hwrng c <misc_major> <dynamic_minor>
   ```

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 CCA coprocessor adapter while zcrypt is loaded automatically removes the random number character device. Reading from the random number character device while all CCA coprocessor adapters are set offline results in an input/output error (EIO). After at least one adapter is set online again, reading from the random number character device continues to return random number data.

Working with cryptographic devices

Typically, cryptographic devices are not directly accessed by users but through user programs. Some tasks can be performed through the sysfs interface.

- “Displaying information about cryptographic devices”
- “Setting devices online or offline” on page 490
- “Setting the polling thread” on page 491
- “Using AP adapter interrupts” on page 492
- “Setting the polling interval” on page 492
- “Dynamically adding and removing cryptographic adapters” on page 493
- “Displaying information about the AP bus” on page 494
- “Unloading the cryptographic device driver” on page 495

Displaying information about cryptographic devices

Use the **lszcrypt** command to display status information about your cryptographic devices; alternatively, you can use sysfs.

About this task

For information about **lszcrypt**, see “lszcrypt - Display zcrypt devices” on page 637.

Each cryptographic adapter is represented in sysfs as a directory of the form

`/sys/bus/ap/devices/card<XX>`

where <*XX*> is the device index for each device. The valid device index range is hex 00 to hex 3f. For example, device 0x1a can be found under `/sys/bus/ap/devices/card1a`. The sysfs directory contains a number of attributes with information about the cryptographic adapter.

Table 58. Cryptographic adapter attributes

Attribute	Explanation
ap_functions	Read-only attribute that represents the function facilities that are installed on this device.
depth	Read-only attribute that represents the input queue length for this device.
hwtype	Read-only attribute that represents the hardware type for this device. The following values are defined: **3** PCICC adapters. **4** PCICA adapters. **5** PCIXCC adapters. **6** CEX2A adapters. **7** CEX2C adapters. **8** CEX3A adapters. **9** CEX3C adapters. **10** CEX4A, CEX4C, or CEX4P adapters.
modalias	Read-only attribute that represents an internally used device bus-ID.

Table 58. Cryptographic adapter attributes (continued)

Attribute	Explanation
online	Read-write attribute that shows whether the device is online (1) or offline (0).
pendingq_count	Read-only attribute that represents the number of requests in the hardware queue.
request_count	Read-only attribute that represents the number of requests that are already processed by this device.
requestq_count	Read-only attribute that represents the number of outstanding requests (not including the requests in the hardware queue).
type	Read-only attribute that represents the type of this device. The following types are defined: • PCICC • PCICA • PCIXCC_MCL2 • PCIXCC_MCL3 • CEX2C • CEX2A • CEX3A • CEX3C • CEX4A • CEX4C • CEX4P

To display status information about your cryptographic devices, you can also use the **lszcrypt** command (see "lszcrypt - Display zcrypt devices" on page 637).

Setting devices online or offline

Use the **chzcrypt** command to set cryptographic devices online or offline.

Procedure

- Preferably, use the **chzcrypt** command with the **-e** option to set cryptographic devices online, or use the **-d** option to set devices offline.

  Examples:

  – To set cryptographic devices (in decimal notation) 0, 1, 4, 5, and 12 online issue:

  ```
  # chzcrypt -e 0 1 4 5 12
  ```

  – To set all available cryptographic devices offline issue:

  ```
  # chzcrypt -d -a
  ```

  For more information about **chzcrypt**, see "chzcrypt - Modify the zcrypt configuration" on page 556.

- Alternatively, write `1` to the `online` sysfs attribute of a cryptographic device to set the device online, or write `0` to set the device offline.

  Examples:

  – To set a cryptographic device with device ID 0x3e online issue:

  ```
  # echo 1 > /sys/bus/ap/devices/card3e/online
  ```

- To set a cryptographic device with device ID 0x3e offline issue:

```
# echo 0 > /sys/bus/ap/devices/card3e/online
```

- To check the online status of the cryptographic device with device ID 0x3e issue:

```
# cat /sys/bus/ap/devices/card3e/online
```

The value is 1 if the device is online or `0` otherwise.

Setting the polling thread

For Linux on z/VM or for Linux instances in LPAR mode on mainframe systems earlier than System z10, enabling the polling thread can improve cryptographic performance.

About this task

As of System z10, Linux in LPAR mode supports AP interrupts that indicate the completion of cryptographic requests. See "Using AP adapter interrupts" on page 492. If AP interrupts are available, it is not possible to activate the polling thread.

The zcrypt device driver can run with or without the polling thread. When zcrypt runs with the polling thread, one processor constantly polls the cryptographic cards for finished cryptographic requests while requests are being processed. The polling thread sleeps when no cryptographic requests are being processed. This mode uses the cryptographic cards as much as possible, at the cost of blocking one processor during cryptographic operations.

Without the polling thread, the cryptographic cards are polled at a much lower rate. The lower rate results in higher latency and reduced throughput for cryptographic requests, but without a noticeable processor load.

Procedure

- Use the **`chzcrypt`** command to set the polling thread.

  Examples:

  - To activate the polling thread issue:

  ```
  # chzcrypt -p
  ```

  - To deactivate the polling thread issue:

  ```
  # chzcrypt -n
  ```

  For more information about **`chzcrypt`**, see "chzcrypt - Modify the zcrypt configuration" on page 556.

- Alternatively, you can set the polling thread through the `poll_thread` sysfs attribute. This read-write attribute can be found at the AP bus level.

  Examples:

  - To activate the polling thread issue:

  ```
  # echo 1 > /sys/bus/ap/poll_thread
  ```

– To deactivate the polling thread issue:

```
# echo 0 > /sys/bus/ap/poll_thread
```

Using AP adapter interrupts

To improve cryptographic performance for Linux instances that run in LPAR mode on an IBM System z10 or later mainframe, use AP interrupts.

About this task

Using AP interrupts instead of the polling thread frees one processor while cryptographic requests are processed.

During module initialization, the zcrypt device driver checks whether AP adapter interrupts are supported by the hardware. If so, polling is disabled and the interrupt mechanism is automatically used.

To tell whether AP adapter interrupts are used, a sysfs attribute called ap_interrupts is defined. The read-only attribute can be found at the AP bus level.

Example

To read the ap_interrupts attribute issue:

```
# cat  /sys/bus/ap/ap_interrupts
```

If interrupts are used, the attribute shows 1, otherwise 0.

Setting the polling interval

Request polling is supported at nanosecond intervals.

Procedure

- Use the **lszcrypt** and **chzcrypt** commands to read and set the polling time.

  Examples:

  – To find out the current polling time, issue:

  ```
  # lszcrypt -b
  ...
  poll_timeout=250000 (nanoseconds)
  ```

  – To set the polling time to 1 microsecond, issue:

  ```
  # chzcrypt -t 1000
  ```

  For more information about **lszcrypt** and **chzcrypt**, see “lszcrypt - Display zcrypt devices” on page 637 and “chzcrypt - Modify the zcrypt configuration” on page 556.
- Alternatively, you can set the polling time through the `poll_timeout` sysfs attribute. This read-write attribute can be found at the AP bus level.

  Examples:

  – To read the poll_timeout attribute for the ap bus issue:

```
# cat  /sys/bus/ap/poll_timeout
```

– To set the poll_timeout attribute for the ap bus to poll, for example, every microsecond, issue:

```
# echo 1000 > /sys/bus/ap/poll_timeout
```

Dynamically adding and removing cryptographic adapters

On an LPAR, you can add or remove cryptographic adapters without the need to reactivate the LPAR after a configuration change.

About this task

z/VM does not support dynamically adding or removing cryptographic adapters.

Linux attempts to detect new cryptographic adapters and set them online every time a configuration timer expires. Read or modify the expiration time with the **lszcrypt** and **chzcrypt** commands.

For more information about **lszcrypt** and **chzcrypt**, see "lszcrypt - Display zcrypt devices" on page 637 and "chzcrypt - Modify the zcrypt configuration" on page 556.

Adding or removing of cryptographic adapters to or from an LPAR is transparent to applications that use clear key functions. If a cryptographic adapter is removed while cryptographic requests are being processed, zcrypt automatically resubmits lost requests to the remaining adapters. Special handling is required for secure key.

Secure key requests are submitted to a dedicated cryptographic coprocessor. If this coprocessor is removed or lost, new requests cannot be submitted to a different coprocessor. Therefore, dynamically adding and removing adapters with a secure key application requires support within the application. For more information about secure key cryptography, see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294. You can obtain this book at www.ibm.com/security/cryptocards/pciecc/library.shtml.

Alternatively, you can read or set the configuration timer through the `config_time` sysfs attribute. This read-write attribute can be found at the AP bus level. Valid values for the `config_time` sysfs attribute are in the range 5 - 120 seconds.

Procedure

You can work with cryptographic adapters in the following ways:

- Add or remove cryptographic adapters by using the SE or HMC. After the configuration timer expires, the cryptographic adapter is added to or removed from Linux, and the corresponding sysfs entries are created or deleted.
- Enable or disable a cryptographic adapter by using the **chzcrypt** command. The cryptographic adapter is only set online or offline in sysfs. The sysfs entries for the cryptographic adapter are retained. Use the **lszcrypt** command to check the results of the **chzcrypt** command.

Examples

- To use the **lszcrypt** and **chzcrypt** commands to find out the current configuration timer setting, issue:

```
# lszcrypt -b
...
config_time=30 (seconds)
...
```

  In the example, the timer is set to 30 seconds.
- To set the configuration timer to 60 seconds, issue:

```
# chzcrypt -c 60
```

To use sysfs to find out the current configuration timer setting, issue:
- To read the configuration timer setting, issue:

```
# cat  /sys/bus/ap/config_time
```

- To set the configuration timer to 60 seconds, issue:

```
# echo 60 > /sys/bus/ap/config_time
```

Displaying information about the AP bus

Use the **lszcrypt -b** command to display status information about the AP bus; alternatively, you can use sysfs.

About this task

For information about **lszcrypt -b**, see "lszcrypt - Display zcrypt devices" on page 637.

The AP bus is represented in sysfs as a directory of the form
`/sys/bus/ap`

The sysfs directory contains a number of attributes with information about the AP bus.

Table 59. AP bus attributes

Attribute	Explanation
ap_domain	Read-only attribute that represents the domain. By default the kernel selects a domain. Alternatively, you can select the domain using a kernel parameter, or a module parameter during module load. See "Module parameters" on page 486.
ap_ctrl_domain_mask	Read-only attribute that represents the installed control domain facilities as a 32-byte field in hexadecimal notation. A maximun number of 256 domains can be addressed. Each bit position represents a dedicated control domain.
ap_interrupts	Read-only attribute that indicates whether interrupt handling for the AP bus is enabled.
config_time	Read-write attribute that represents a time interval in seconds used to detect new crypto devices.

Table 59. AP bus attributes (continued)

Attribute	Explanation
poll_thread	Read-write attribute that represents the polling thread for the AP bus.
poll_timeout	Read-write attribute that represents the time interval of the poll thread in nanoseconds.

Example

```
# lszcrypt -b
ap_domain=5
ap_interrupts are enabled
config_time=30 (seconds)
poll_thread is disabled
poll_timeout=250000 (nanoseconds)
```

Unloading the cryptographic device driver

You can use **rmmod** or **modprobe** to unload the cryptographic device driver modules.

Before you begin

The use count of the modules must be zero before you can unload them.

Procedure

- To unload the entire zcrypt device driver, explicitly unload each module. For example:

|
```
# rmmod zcrypt_cex4 zcrypt_cex2a zcrypt_pcicc zcrypt_msgtype50 zcrypt_msgtype6 zcrypt_api ap
```

- Alternatively, unload all unused modules that are related to zcrypt_api. You must unload only modules that were actually loaded. For example, if only the zcrypt_cex4 and zcrypt_msgtype50 modules are loaded in addition to zcrypt_api and ap, use:

```
# rmmod zcrypt_cex4 zcrypt_msgtype50 zcrypt_api ap
```

  List the arguments in the order given.

External programming interfaces

Applications can directly access the zcrypt device driver through an API.

Programmers: This information is intended for those who want to program against the cryptographic device driver or against the available cryptographic libraries.

For information about the library APIs, see the following files in the Linux source tree:

- The libica library `/usr/include/ica_api.h`
- The openCryptoki library `/usr/include/opencryptoki/pkcs11.h`
- The CCA library `/opt/IBM/`*<prod>*`/include/csulincl.h`, where *<prod>* is specific to the particular hardware product.

`ica_api.h`, `pkcs11.h`, and `csulincl.h` are present after their libraries are installed.

Clear key cryptographic functions

The libica library provides a C API to clear-key cryptographic functions that are supported by System z hardware. You can configure both openCryptoki (by using the icatoken) and openssl (by using the ibmca engine) to use System z clear-key cryptographic hardware support through libica. See *libica Programmer's Reference*, SC34-2602 for details about the libica functions.

If you must circumvent libica and access the zcrypt device driver directly, your user space program must open the z90crypt device node, and submit the cryptographic request with an IOCTL. The IOCTL subfunction ICARSAMODEXPO performs RSA modular exponent encryption and decryption. The IOCTL ICARSACRT performs RSA CRT decryption. See the cryptographic device driver header file in the Linux source tree:
`/usr/include/asm-s390/zcrypt.h`

Ensuring the correct length for RSA encryption requests: Cryptographic CCA coprocessors might reject RSA encryption requests for which the numerical value of the data to be encrypted is greater than the modulus.

Ensuring correct data padding when using PCICC

Correct padding can be important when using a PCICC or PCIXCC coprocessor.

If you have a PCICC coprocessor only, or are attempting to use a CRT key on a system with a PCIXCC coprocessor (MCL2) only, you must ensure that your data is PKCS-1.2 padded. PKCS-1.2 padding means that the key is formatted as defined in the RSA PKCS#1 v2.0 standard for the RSAES-PKCS1-v1.5 encryption and decryption scheme. In this case, the zcrypt device driver or the cryptographic hardware might change the padding. If the requests are not padded correctly, the cryptographic operations are performed in software.

If you have at least one PCICA, PCIXCC (MCL3), CEX2C, CEX2A, CEX3C, CEX3A, CEX4C, or CEX4A card, or if you are using only Mod-Expo keys with a PCIXCC (MCL2) coprocessor, you do not need to ensure that your data is PKCS-1.2 padded. In this case, the padding remains unchanged.

Secure key cryptographic functions

To use clear key cryptographic functions in your user space program, use the CCA host library. The CCA host library opens the z90zcrypt device node and submits the cryptographic requests by using the ZSECSENDCPRB IOCTL subfunction.

See *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294, for installation and setup instructions, feature coexistence information, and how to use CCA functions. You can obtain this publication at www.ibm.com/security/cryptocards/pciecc/library.shtml.

Reading true random numbers

To read true random numbers, a user space program must open the `hwrng` device and read as many bytes as needed from the device.

Tip: Using the output of the `hwrng` device to periodically reseed a pseudo-random number generator might be an efficient use of the random numbers.

Chapter 43. Pseudo-random number device driver

The pseudo-random number device driver provides user-space applications with pseudo-random numbers generated by the System z CP Assist for Cryptographic Function (CPACF).

The pseudo-random number device provides pseudo-random numbers similar to the Linux pseudo-random number device `/dev/urandom` but offers a better performance.

Building a kernel with the pseudo-random number device driver

Select option CONFIG_S390_PRNG in the kernel configuration menu to build a kernel with the pseudo-random number device driver.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

```
Cryptographic API --->
   ...
   Hardware crypto devices  --->
      ...
      Pseudo random number generator device driver            (CONFIG_S390_PRNG)
```

Figure 106. Pseudo-random number kernel configuration menu options

The pseudo-random number device driver can be compiled into the kernel or as a separate module, prng.

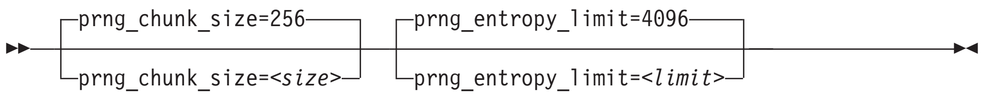
Setting up the pseudo-random number device driver

Unless your distribution performs these tasks for you, you must load the device driver module, create a device node and, optionally, make it available to non-root users.

Kernel parameters

You can configure the pseudo-random number device driver if the device driver was compiled into the kernel. You configure the device driver by adding parameters to the kernel parameter line.

Kernel parameter syntax



where:

<size>
: is the number of bytes that the Linux kernel reads from the mainframe hardware in a single read operation. This value is independent from the number of bytes a user space program might read from the device. The default is 256 bytes.

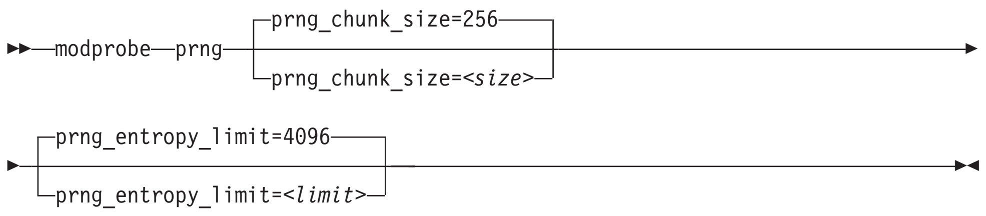
<limit>
: is the number of bytes read from the mainframe hardware after which extra randomness is added to the state of the pseudo random number generator. The default is 4096 bytes.

The defaults were chosen for good results with most workloads. Changing these settings might be detrimental to cryptographic performance.

Module parameters

You can load and configure the pseudo-random number device driver if it was compiled as a separate module.

Module parameter syntax



where:

<size>
: is the number of bytes that the Linux kernel reads from the mainframe hardware in a single read operation. This value is independent from the number of bytes a user space program might read from the device. The default is 256 bytes.

<limit>
: is the number of bytes read from the mainframe hardware after which extra randomness is added to the state of the pseudo random number generator. The default is 4096 bytes.

Device node

User-space programs access the pseudo-random-number device through a device node, `/dev/prandom`.

Your distribution might create this device node for you or provide udev to create it (see "Device nodes provided by udev" on page 4).

If no device node is created for you, you must create it yourself, for example, with the **mknod** command. See the **mknod** man page for further details.

The pseudo-random-number device requires a misc character device node. During load, a sysfs directory, `/sys/devices/virtual/misc/prandom`, is created. This directory contains a `dev` attribute with 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. Your distribution might extend the access to other users.

If access to the device is restricted to root on your system and your distribution uses udev, add the following udev rule. It automatically extends access to the device to other users.

```
KERNEL=="prandom",            MODE="0444", OPTIONS="last_rule"
```

If access to the device is restricted to root on your system and your distribution does not use udev, use the **chmod** command to make the device available to other users.

Reading pseudo-random numbers

The pseudo-random number device is read-only. Use the read function, cat program, or dd program to obtain random numbers.

Example

In this example `bs` specifies the block size in bytes for transfer, and `count` specifies the number of records with block size. The bytes are written to the output file.

```
dd if=/dev/prandom of=<output file name> bs=<xxxx> count=<nnnn>
```

Part 8. Performance measurement using hardware facilities

The System z hardware provides performance data that can be accessed by Linux on System z.

Gathering performance data constitutes an additional load on the Linux instance on which the application to be analyzed runs. Hardware support for data gathering can reduce the extra load and can yield more accurate data.

For the performance measurement facilities of z/VM, see “Performance monitoring for z/VM guest virtual machines” on page 413.

Other performance relevant information is provided in the context of the respective device driver or feature. For example, see “Working with DASD statistics in debugfs” on page 172 for DASD performance and “Starting and stopping collection of QETH performance statistics” on page 285 for qeth group devices.

Newest version

You can find the newest version of this publication at
www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Restrictions

For prerequisites and restrictions see
www.ibm.com/developerworks/linux/linux390/development_technical.html
www.ibm.com/developerworks/linux/linux390/development_restrictions.html

Chapter 44. 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.

The channel measurement facility provides the following features:
- Basic channel measurement format for concurrently collecting data on up to 4096 devices. (Specifying 4096 or more channels causes high memory consumption, and enabling data collection might not succeed.)
- Extended channel measurement format for concurrently collecting data on an unlimited number of devices.
- Data collection for all channel-attached devices, except those using QDIO (that is, except qeth and SCSI-over-Fibre channel attached devices)

Kernel builders: The channel measurement facility is always included in the Linux kernel. You do not need to select any options in the kernel configuration menu.

Setting up the channel measurement facility

Configure the channel measurement facility by adding parameters to the kernel parameter file.

You can configure the channel measurement facility by adding parameters to the kernel parameter file.

Channel measurement facility kernel parameters

```
    ┌─cmf.format=-1──────┐   ┌─cmf.maxchannels=1024───────────┐  (1)
►►──┼────────────────────┼───┼────────────────────────────────┼──────────────────►◄
    └─cmf.format=──┬─0─┬─┘   └─cmf.maxchannels=<no_channels>──┘
                   └─1─┘
```

Notes:

1 If you specify both parameter=value pairs, separate them with a blank.

where:

`cmf.format`
defines the format, `0` for basic and `1` for extended, of the channel measurement blocks. The default, `-1`, uses the extended format for z990 and later mainframes and the basic format for earlier mainframes.

`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

Typical tasks that you need to perform when you work with the channel measurement facility is controlling data collection and reading data.

Enabling, resetting, and switching off data collection

Control data collection through the `cmb_enable` sysfs attribute of the device.

Procedure

- To enable data collection, write 1 to the `cmb_enable` attribute. If data collection was already enabled, writing 1 to the attribute resets all collected data to zero.

  Issue a command of this form:

  ```
  # echo 1 > /sys/bus/ccw/devices/<device_bus_id>/cmb_enable
  ```

  where `/sys/bus/ccw/devices/<device_bus_id>` represents the device in sysfs.

  When data collection is enabled for a device, a subdirectory `/sys/bus/ccw/devices/<device_bus_id>/cmf` is created that contains several attributes. These attributes contain the collected data (see "Reading data").
- To switch off data collection issue a command of this form:

  ```
  # echo 0 > /sys/bus/ccw/devices/<device_bus_id>/cmb_enable
  ```

  When data collection for a device is switched off, the subdirectory `/sys/bus/ccw/devices/<device_bus_id>/cmf` and its content are deleted.

Example

In this example, data collection for a device `/sys/bus/ccw/devices/0.0.b100` is already active and reset:

```
# cat /sys/bus/ccw/devices/0.0.b100/cmb_enable
1
# echo 1 > /sys/bus/ccw/devices/0.0.b100/cmb_enable
```

Reading data

Read the sysfs attributes with collected I/O data, for example with the **`cat`** command.

Procedure

While data collection is enabled for a device, the directories that represent it in sysfs contain a subdirectory, `cmf`, with several read-only attributes. These attributes hold the collected data.

To read one of the attributes issue a command of this form:

```
# cat /sys/bus/ccw/devices/<device_bus_id>/cmf/<attribute>
```

where /sys/bus/ccw/devices/*<device_bus_id>* is the directory that represents the device, and <*attribute*> the attribute to be read. Table 60 summarizes the available attributes.

Table 60. Attributes with collected I/O data

Attribute	Value
ssch_rsch_count	An integer that represents the ssch rsch count value.
sample_count	An integer that represents the sample count value.
avg_device_connect_time	An integer that represents the average device connect time, in nanoseconds, per sample.
avg_function_pending_time	An integer that represents the average function pending time, in nanoseconds, per sample.
avg_device_disconnect_time	An integer that represents the average device disconnect time, in nanoseconds, per sample.
avg_control_unit_queuing_time	An integer that represents the average control unit queuing time, in nanoseconds, per sample.
avg_initial_command_response_time	An integer that represents the average initial command response time, in nanoseconds, per sample.
avg_device_active_only_time	An integer that represents the average device active only time, in nanoseconds, per sample.
avg_device_busy_time	An integer that represents the average value device busy time, in nanoseconds, per sample.
avg_utilization	A percent value that represents the fraction of time that has been spent in device connect time plus function pending time plus device disconnect time during the measurement period.
avg_sample_interval	An integer that represents the average time, in nanoseconds, between two samples during the measurement period. Can be "-1" if no measurement data has been collected.
avg_initial_command_response_time	An integer that represents the average time in nanoseconds between the first command of a channel program being sent to the device and the command being accepted. Available in extended format only.
avg_device_busy_time	An integer that represents the average time in nanoseconds of the subchannel being in the "device busy" state when initiating a start or resume function. Available in extended format only.

Example

To read the avg_device_busy_time attribute for a device /sys/bus/ccw/devices/0.0.b100:

```
# cat /sys/bus/ccw/devices/0.0.b100/cmf/avg_device_busy_time
21
```

Chapter 45. OProfile hardware sampling support

OProfile is a performance analysis tool for Linux that can use hardware sampling support to capture performance data for processes, shared libraries, the kernel, and device drivers.

For general information about OProfile, see sourceforge.net/projects/oprofile.

OProfile hardware sampling can be used for Linux instances in LPAR mode. The hardware sampling support that is used by OProfile was introduced for System z10 in October 2008.

As of kernel 3.3, the OProfile user interface that is described here can be used with OProfile 0.9.8.

The previous user interface is still available to make OProfile compatible with earlier versions. Use the previous interface with OProfile versions older than 0.9.8. For a description of earlier versions, see previous versions of this publication in the archive section of www.ibm.com/developerworks/linux/linux390/documentation_dev.html.

Note: OProfile and perf-based sampling tools use the CPU-measurement sampling facility and, therefore, cannot simultaneously collect sample data.

Building a kernel with OProfile support

There are specific options in the kernel configuration menu that you must select to compile a kernel with OProfile support.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

It describes the options that you must select in the Linux configuration menu to include OProfile support.

Figure 107 summarizes the required kernel configuration menu options:

```
General setup --->
   ...
   Profiling support                                        (CONFIG_PROFILING)
   OProfile system profiling                                (CONFIG_OPROFILE)
```

Figure 107. Kernel configuration menu options for OProfile

CONFIG_PROFILING
enables the extended profiling support mechanisms that are used by profilers.

CONFIG_OPROFILE
enables the OProfile profiler. You can compile OProfile support into the kernel or as a separate module, `oprofile`.

Setting up OProfile support

After you install the OProfile package, you must initialize OProfile on your Linux instance and enable hardware sampling for the LPAR in which the Linux instance runs.

Initializing OProfile

Before initialization, the `/dev/oprofile` file system is not available and commands that act on files within this file system fail.

Issue:

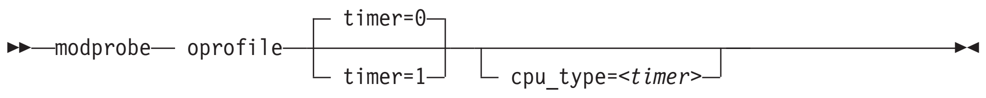
```
# opcontrol --init
```

This command loads the `oprofile` module if it was not compiled into the kernel image and initializes the OProfile support. You can load the module with module parameters, see “OProfile module parameters.” For more information, see oprofile.sourceforge.net/docs.

OProfile module parameters

If the OProfile support was built as a separate module, you can specify the timer switch and the compatibility mode as parameters when you load the module with the **modprobe** command.

OProfile module parameter syntax



where:

timer
: `timer=1` forces timer-interrupt based sampling.
`timer=0` is the default, and uses hardware-based sampling, if possible.

cpu_type=<*timer*>
: specifies that OProfile uses the old virtual file system from versions earlier than 0.9.8. Hardware sampling is nevertheless used if available. The only valid value for the `cpu_type` parameter is **timer**.

The kernel module provides the following virtual file system:

```
/dev/oprofile/0/enabled
/dev/oprofile/0/event
/dev/oprofile/0/count
/dev/oprofile/0/unit_mask
/dev/oprofile/0/kernel
/dev/oprofile/timer/enabled
```

The `event` file supports only one event: HWSAMPLING with the value 0. The `unit_mask` file must always be 0. You use the other files when you work with OProfile. See "Working with OProfile" on page 510.

Example

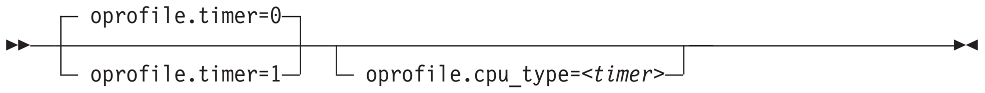
To force the use of timer-based sampling, load the module as follows:

```
modprobe oprofile timer=1
```

OProfile kernel parameters

If the OProfile support was compiled into the kernel, you specify the timer switch and the compatibility mode by adding parameters to the kernel parameter line.

OProfile kernel parameter syntax



where the parameters have the same meaning as described in "OProfile module parameters" on page 508.

Setting up an LPAR for hardware sampling

To enable hardware sampling for an LPAR, you must activate the LPAR with authorization for basic sampling control.

For more information, see the *Support Element Operations Guide* for your mainframe system. Also see "Authorizing an LPAR" on page 514.

To check whether hardware sampling is enabled, read the `hwsampler` attribute:

```
# cat /dev/oprofile/hwsampling/hwsampler
1
```

If hardware sampling is enabled, the value is 1.

If the value is `0`, timer-interrupt based sampling is used. The reason might be that your System z hardware does not support hardware sampling, that your LPAR was not set up for hardware sampling, or that your Linux instance runs as a z/VM guest.

You can disable hardware sampling by writing `0` to the `hwsampler` attribute:

```
# echo 0 > /dev/oprofile/hwsampling/hwsampler
```

Working with OProfile

You might have to set up resources for sampling and the rate of sampling, enable and start sampling, and filter samples.

This section describes typical tasks that you need to perform when working with OProfile.

- "Starting and stopping sampling"
- "Setting the sampler memory"
- "Enabling and disabling hardware sampling"
- "Setting the hardware sampling rate" on page 511
- "Filtering the hardware samples" on page 511

Starting and stopping sampling

You start and stop sampling as you would on any hardware platform.

See oprofile.sourceforge.net/docs for details.

Setting the sampler memory

Set the sampler memory size with the `opcontrol` command.

About this task

The best size for the sampler memory depends on the particular system and the workload to be measured. Providing the sampler with too little memory results in lost samples. Reserving too much system memory for the sampler impacts the overall performance and, hence, also the workload to be measured.

Procedure

To set the size of the memory that is reserved for sampled data, issue a command of this form:

```
# opcontrol --s390hwsampbufsize=<num>
```

where <*num*> is the memory size in multiples of 2 MB. The default is 1.

Example

This example shows how to set the memory size to 4 MB (2x2 MB):

```
# opcontrol --s390hwsampbufsize=2
```

Enabling and disabling hardware sampling

Enable or disable hardware sampling through the `/dev/oprofile/0/enabled` attribute in the `/dev/oprofile` file system.

About this task

If hardware sampling is available and the compatibility mode (cpu_type=timer) is not used, the `/dev/oprofile/0` directory is present.

Hardware sampling might be disabled for the following reasons:

- The System z hardware does not support hardware sampling.
- The LPAR is not set up for hardware sampling.
- The Linux instance runs as a z/VM guest.

Procedure

You can control hardware sampling by issuing a command of the form:

```
# echo <value> > /dev/oprofile/0/enabled
```

where *<value>* is 1 for enabled hardware sampling, and 0 for disabled hardware sampling. By default, hardware sampling is turned on, if it is available.
If hardware sampling is available (`/dev/oprofile/0` is present), you can switch back to timer based sampling by issuing:

```
# echo "0" > /dev/oprofile/0/enabled
```

Alternatively, you can use the `/dev/oprofile/timer/enabled` file to work with timer-based sampling. Specifying the following command switches timer-based sampling on:

```
# echo "1" > /dev/oprofile/timer/enabled
```

Setting the hardware sampling rate

Set the hardware sampling rate through the `/dev/oprofile/0/count` attribute in the `/dev/oprofile` file system.

Procedure

Issue a command of this form to set the sampling rate:

```
# echo  <value> > /dev/oprofile/0/count
```

where *<value>* is the sampling rate in cycles between samples. The sampling rate is capped to the range valid for the hardware. The default is 4127518. Using low values might considerably impact the workload to be measured.

Example

This example sets the sampling rate to twice the default rate:

```
# echo 8255036 > /dev/oprofile/0/count
```

or, using the OProfile 0.9.8 command-line tool:

```
# opcontrol --event HWSAMPLING:8255036
```

Filtering the hardware samples

When you use hardware sampling, you can filter the samples by user-space or kernel events to make the data sample smaller.

Procedure

- Filter the samples by issuing a command of the form:

  ```
  # echo <value1> > /dev/oprofile/0/kernel
  ```

  where *<value1>* is 1 to include kernel samples, and 0 to omit them. The default is 1.
- Alternatively, issue:

  ```
  # echo <value2> > /dev/oprofile/0/user
  ```

  where *<value2>* is 1 to include user-space samples, and 0 to omit them. The default is 1.
- You can achieve the same results with the OProfile 0.9.8 command-line tools:

  ```
  # opcontrol --event HWSAMPLING:<count>:0:<value1>:<value2>
  ```

Example

To get kernel samples only:

```
# echo "1" > /dev/oprofile/0/kernel
# echo "0" > /dev/oprofile/0/user
```

or with the OProfile 0.9.8 command-line tools:

```
# opcontrol --event HWSAMPLING:8255036:0:1:0
```

Chapter 46. Using the CPU-measurement facilities

Use the CPU-measurement counter facility and sampling facility to obtain performance data for Linux in LPAR mode.

The z/Architecture CPU-measurement facilities were introduced for System z10 in October 2008.

Counter facility
The hardware counters are grouped into the following counter sets:
- Basic counter set
- Problem-state counter set
- Crypto-activity counter set
- Extended counter set

A further common counter set, the Coprocessor group counter set, cannot be accessed from Linux on System z.

Sampling facility
The sampling facility includes the following sampling modes:
- Basic-sampling mode
- Diagnostic-sampling mode

The diagnostic-sampling mode is intended for use by IBM support only.

Conflict with OProfile: Perf-based sampling tools and OProfile use the CPU-measurement sampling facility and, therefore, cannot simultaneously collect sample data.

The number and type of individual counters and the details of the sampling facility depend on your System z hardware model. Use the **`lscpumf`** command to find out what is available for your hardware (see "lscpumf - Display information about the CPU-measurement facilities" on page 614). For details, see *IBM The CPU-Measurement Facility Extended Counters Definition for z10, z196, z114 and zEC12*, SA23-2261.

You can use the perf tool on Linux to access the hardware counters and sample data of the CPU-measurement facilities. If you want to write your own application for analyzing counter or sample data, you can use the `libpfm4` library. This library is available on sourceforge at perfmon2.sourceforge.net.

Building a kernel with the CPU-measurement facilities

Select the common code kernel configuration option CONFIG_PERF_EVENTS to build a kernel that supports the z/Architecture CPU-measurement facilities.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Figure 108 on page 514 shows where to find the CONFIG_PERF_EVENTS option in the kernel configuration menu.

```
General setup --->
   ...
   Kernel Performance Events And Counters --->
      Kernel performance events and counters          (CONFIG_PERF_EVENTS)
```

Figure 108. Kernel configuration menu options for performance events

The CONFIG_PERF_EVENTS option enables kernel support for the performance events that are provided by various hardware and software platforms, including the z/Architecture hardware.

Building and installing the perf tool

You build the perf tool separately from the kernel.

Procedure

Perform these steps to build and install the perf tool:

1. Go to the root of the Linux source tree.
2. Build the tool by issuing this command:
   ```
   # make -C tools/perf/
   ```
3. Install the tool by issuing this command:
   ```
   # make -C tools/perf/ install
   ```

Working with the CPU-measurement facilities

You can use the perf tool to work with the CPU-measurement facilities for authorized LPARs.

- "Authorizing an LPAR"
- "Reading CPU-measurement counters" on page 515
- "Collecting CPU-measurement sample data" on page 516
- "Setting limits for the sampling facility buffer" on page 517
- "Obtaining details about the CPU-measurement facilities" on page 518

Authorizing an LPAR

The LPAR within which the Linux instance runs must be authorized to use the CPU-measurement counter sets or sampling modes. Use the HMC or SE to authorize the LPAR.

About this task

The details of the steps in this task can differ, depending on your hardware. For more information, see the *Support Element Operations Guide* for your mainframe system.

Procedure

Perform these steps on the HMC or SE to grant authorizations:

1. Navigate to the LPAR for which you want to grant authorizations.

2. Within the LPAR profile, select the **Security** page.
3. Within the counter facility options, select each counter set you want to use. The coprocessor group counter set is not supported by Linux on System z.
4. If you want to use the sampling facility, select the basic sampling mode within the sampling facility options.

   Note: You cannot enable the diagnostic mode unless it has been enabled for you by IBM support.
5. Click **Save**.

What to do next

Deactivate, activate, and IPL the LPAR to make the authorization take effect. For more information, see the *Support Element Operations Guide* for your mainframe system.

When your Linux instance is available again, you can use the **`lscpumf`** command to confirm that the authorizations are in place (see “lscpumf - Display information about the CPU-measurement facilities” on page 614).

Reading CPU-measurement counters

Use the perf tool to read CPU-measurement counters.

Before you begin

You must know the symbolic name for the counter or the hexadecimal value of the counter number. Issue **lscpumf -c** to obtain a list of counters, their symbolic names, and their numbers (see “lscpumf - Display information about the CPU-measurement facilities” on page 614).

You can also find the decimal values in *z/Architecture The Load-Program-Parameter and the CPU-Measurement Facilities*, SA23-2260 and in *IBM The CPU-Measurement Facility Extended Counters Definition for z10, z196, z114 and zEC12*, SA23-2261.

Procedure

Issue a **`perf`** command to read a counter.
- Using symbolic names:

```
# perf stat -e cpum_cf/event=<symbolic_name>/ -- <path_to_app>
```

- Using raw events:

```
# perf stat -e r<hex_counter_number> -- <path_to_app>
```

Where:

`-e cpum_cf/event=`*`<symbolic_name>`***`/`**
: specifies a counter through a symbolic name. Symbolic names are lengthy but meaningful and the same for all mainframes models that support the counter.

`-e r`*`<hex_counter_number>`*
: specifies the hexadecimal value for the counter number as a raw event. This specification is short but abstract and the numbers can differ between mainframe models.

<*path_to_app*>
: specifies the path to the application to be evaluated. The counters are incremented for all threads that belong to the specified application. If you specify `-a` instead of the double hyphen and path, system-wide counter data is read.

Tip: You can read multiple counters by specifying a comma-separated list of counters, for example, `-e r20,r21`.
For more information about the **perf** command, see the **perf** or **perf-stat** man page.

Examples

Issue one of the following commands to read the problem-state cycle count counter (symbolic name PROBLEM_STATE_CPU_CYCLES; hexadecimal value 20) and the problem-state instruction count counter (symbolic name PROBLEM_STATE_INSTRUCTIONS; hexadecimal value 21) for an application /bin/df.

- Using symbolic names:

```
# perf stat -e cpum_cf/event=PROBLEM_STATE_CPU_CYCLES/,cpum_cf/event=PROBLEM_STATE_INSTRUCTIONS/ -- /bin/df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/dasda1            6967656   3360888   3229780  51% /
none                    942956        88    942868   1% /dev/shm
/dev/dasdb1            6967656   4135792   2471260  63% /root

 Performance counter stats for '/bin/df':

        1,258,624      PROBLEM_STATE_CPU_CYCLES
          341,792      PROBLEM_STATE_INSTRUCTIONS

      0.002676094 seconds time elapsed
```

- Using raw events:

```
# perf stat -e r20,r21 -- /bin/df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/dasda1            6967656   3360884   3229784  51% /
none                    942956        88    942868   1% /dev/shm
/dev/dasdb1            6967656   4135792   2471260  63% /root

 Performance counter stats for '/bin/df':

        1,233,295      r20
          341,792      r21

      0.002526281 seconds time elapsed
```

Collecting CPU-measurement sample data

Use the perf tool to read CPU-measurement sample data.

Procedure

Issue a command of this form to read sample data:

```
# perf record -e cpum_sf/event=SF_CYCLES_BASIC/ -- <path_to_app>
```

Where <*path_to_app*> is the path to the application for which you want to collect sample data. If you specify `-a` instead of the double hyphen and path, system-wide sample data is collected. Instead of the symbolic name, you can also specify the raw event name `rB0000`.

Example

```
# perf record -e cpum_sf/event=SF_CYCLES_BASIC/ -- /bin/df
Filesystem          1K-blocks      Used Available Use% Mounted on
/dev/dasda1           6967656   3360508   3230160  51% /
none                   942956        88    942868   1% /dev/shm
/dev/dasdb1           6967656   4132924   2474128  63% /root
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.001 MB perf.data (~29 samples) ]
```

What to do next

You can now display the sample data by issuing the following command:

```
# perf report
```

For more information about collecting and displaying sample data with the **`perf`** command, see the **`perf-record`** and the **`perf-report`** man pages.

Hint: You can use the **`perf record -F`** option to collect sample data at a high frequency or the **`perf record -c`** option to collect sample data for corresponding short sampling intervals. Specified values must be supported by both the CPU-measurement sampling facility and perf. Issue **`lscpumf -i`** to find out the maximum and minimum values for the CPU-measurement sampling facility. If perf fails at a high sampling frequency, you might have to adjust the `kernel.perf_event_max_sample_rate` system control to override default perf limitations.

Setting limits for the sampling facility buffer

Use the **`chcpumf`** command to set the minimum and maximum buffer size for the CPU-measurement sampling facility.

See "chcpumf - Set limits for the CPU measurement sampling facility buffer" on page 547.

Before you begin

For each CPU, the CPU-measurement sampling facility has a buffer for writing sample data. The required buffer size depends on the sampling function and the sampling interval that is used by the perf tool. The sampling facility starts with an initial buffer size that depends on the expected requirements, your System z hardware, and the available hardware resources. During the sampling process, the sampling facility increases the buffer size if required.

The sampling facility is designed for autonomous buffer management, and you do not usually need to intervene. You might want to change the minimum or maximum buffer size, for example, for one of the following reasons:
- There are considerable resource constraints on your system that cause perf sampling to malfunction and sample data to be lost.
- As an expert user of perf and the sampling facility, you want to explore results with particular buffer settings.

Procedure

Use the **chcpumf** command to set the minimum and maximum buffer sizes.

1. Optional: Specify the **lscpumf** command with the `-i` parameter to display the current limits for the buffer size (see "lscpumf - Display information about the CPU-measurement facilities" on page 614).
2. Optional: Specify the **chcpumf** command with the `-m` parameter to set the minimum buffer size.

   Example:

   ```
   # chcpumf -m 500
   ```

   The value that you specify with `-m` is the minimum buffer size in multiples of sample-data-blocks. A sample-data-block occupies approximately 4 KB. The specified minimum value is compared with the initial buffer size that is calculated by the sampling facility. The greater value is then used as the initial size when the sampling facility is started.
3. Optional: Specify the **chcpumf** command with the `-x` parameter to set the maximum buffer size.

   Example:

   ```
   # chcpumf -x 1000
   ```

   The value that you specify with `-x` is the maximum buffer size in multiples of sample-data-blocks. A sample-data-block occupies approximately 4 KB. The specified maximum is the upper limit to which the sampling facility can adjust the buffer.

Example

Tips:

- You can specify both, the minimum and the maximum buffer size with a single command.
- Use the `-V` parameter to to display the minimum and maximum buffer settings that apply as a result of the command.

Example: To change the minimum buffer size to 500 times the size of a sample-data-block and the maximum buffer size to 1000 times the size of a sample-data-block, issue:

```
# chcpufm -V -m 500 -x 1000
Sampling buffer sizes:
    Minimum:    500 sample-data-blocks
    Maximum:   1000 sample-data-blocks
```

Obtaining details about the CPU-measurement facilities

You can obtain version information for the CPU-measurement counter and sampling facility and check which counter sets are authorized on your LPAR.

Procedure

1. Issue the **lscpumf** command with the -i parameter to display detailed information and debug data about the CPU-measurement facilities.

   Example:

```
# lscpumf -i
CPU-measurement counter facility
---------------------------------------------------------------------------
Version: 1.2

Authorized counter sets:
    Basic counter set
    Problem-State counter set

Linux perf event support: Yes (PMU: cpum_cf)

CPU-measurement sampling facility
---------------------------------------------------------------------------
Sampling Interval:
     Minimum:      18228 cycles (approx.   285714 Hz)
     Maximum:  170650536 cycles (approx.       30 Hz)

Authorized sampling modes:
     basic      (sample size:  32 bytes)


Linux perf event support: Yes (PMU: cpum_sf)

Current sampling buffer settings for cpum_sf:
    Basic-sampling mode
        Minimum:     15 sample-data-blocks (  64KB)
        Maximum:   8176 sample-data-blocks (  32MB)
```

2. Optional: For more detailed information, including debug information, use the magic sysrequest function with character p. This function triggers kernel messages.

   For example, trigger the messages from procfs:

```
# echo p > /proc/sysrq-trigger
```

   Note: If you call magic sysrequest functions with a method other than through the procfs, you might need to activate them first. For more information about the magic sysrequest functions, see "Using the magic sysrequest feature" on page 58.

   Find the messages by issuing the **dmesg** command and looking for output lines that include CPUM_CF or CPUM_SF.

   More information: For details about the information in the messages, see *z/Architecture The Load-Program-Parameter and the CPU-Measurement Facilities*, SA23-2260, and the perf section in *Kernel Messages*, SC34-2599.

Part 9. Diagnostics and troubleshooting

These resources are useful when diagnosing and solving problems for Linux on System z.

Newest version

You can find the newest version of this publication at
www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Restrictions

For prerequisites and restrictions see
www.ibm.com/developerworks/linux/linux390/development_technical.html
www.ibm.com/developerworks/linux/linux390/development_restrictions.html

When reporting a problem to IBM support, you might be asked to supply a kernel dump. See *Using the Dump Tools*, SC33-8412 for information about how to create dumps.

Chapter 47. Logging I/O subchannel status information

When investigating I/O subchannels, support specialists might request operation status information for the subchannel.

About this task

The channel subsystem offers a logging facility that creates a set of log entries with such information. From Linux, you can trigger this logging facility through sysfs.

The log entries are available through the SE Console Actions Work Area with the View Console Logs function. The entries differ dependent on the device and model that is connected to the subchannel. On the SE, the entries are listed with a prefix that identifies the model. The content of the entries is intended for support specialists.

Procedure

To create a log entry, issue a command of this form:

```
# echo 1 > /sys/devices/css0/<subhannel-bus-id>/logging
```

where *<subchannel-bus-id>* is the bus ID of the I/O subchannel that corresponds to the I/O device for which you want to create a log entry.
To find out how your I/O devices map to subchannels you can use, for example, the **lscss** command.

Example

In this example, first the subchannel for an I/O device with bus ID `0.0.3d07` is identified, then logging is initiated.

```
# lscss -d 0.0.3d07
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.3d07 0.0.000c  1732/01 1731/01      80  80  ff   05000000 00000000
# echo 1 > /sys/devices/css0/0.0.000c/logging
```

Chapter 48. Control program identification

If your Linux instance runs in LPAR mode, you can provide the names of the Linux instance and, if applicable, sysplex to the control program identification (CPI) feature.

You can use one of these interfaces to specify the names:
- The sysfs interface `/sys/firmware/cpi`
- The control program identification module, sclp_cpi

The names are used, for example, to identify the Linux instance or the sysplex on the HMC.

Building a kernel with CPI support

If you want to use the CPI module rather than the sysfs interface, you need to select the kernel configuration option CONFIG_SCLP_CPI.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

You compile the CPI support as a separate module, sclp_cpi.

```
Device Drivers --->
   ...
   Character devices --->
      ...
      --- S/390 character device drivers (depends on S390) ---
      ...
      Control-Program Identification                      (CONFIG_SCLP_CPI)
```

Figure 109. Kernel configuration menu option for control program identification

Working with the CPI support

Typical tasks that you perform when you work with CPI support are setting and displaying system information.
- "Loading the CPI module"
- "Defining a sysplex name" on page 526
- "Defining a system name" on page 526
- "Defining a system type" on page 527
- "Displaying the system level" on page 527
- "Sending system data to the SE" on page 527

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.

About this task

When you load the CPI module, the following is sent to the SE:

- System name (if provided)
- Sysplex name (if provided)
- System type (automatically set to "LINUX")
- System level (automatically set to the value of LINUX_VERSION_CODE)

CPI module parameter syntax

```
►►──modprobe── sclp_cpi──┬─────────────────────────┬──┬───────────────────────────┬──────►◄
                         └─ system_name=<system>──┘  └─ sysplex_name=<sysplex>──┘
```

where:

`system_name` `=` *<system>*
: specifies an eight-character system name of the following set: A-Z, 0-9, $, @, #, and blank. The specification is converted to uppercase.

`sysplex_name` `=` *<sysplex>*
: specifies an eight-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 `system_name` attribute in the `/sys/firmware/cpi` directory in sysfs to specify a system name.

About this task

The system name is a string that consists of up to eight characters of the following set: A-Z, 0-9, $, @, #, and blank.

The `system_name` attribute is intended for setting the name only. To confirm the current system name, check the HMC.

Example

```
# echo LPAR12 > /sys/firmware/cpi/system_name
```

Defining a sysplex name

Use the `sysplex_name` attribute in the `/sys/firmware/cpi` directory in sysfs to specify a sysplex name.

About this task

The sysplex name is a string that consists of up to eight characters of the following set: A-Z, 0-9, $, @, #, and blank.

This attribute is intended for setting the name only. To confirm the current sysplex name, check the HMC.

Example

```
# echo SYSPLEX1 > /sys/firmware/cpi/sysplex_name
```

Defining a system type

Use the `system_type` attribute in the `/sys/firmware/cpi` directory in sysfs to specify LINUX as the system type.

Example

```
# echo LINUX > /sys/firmware/cpi/system_type
```

Displaying the system level

Read version information about your Linux instance from the `system_level` attribute in the `/sys/firmware/cpi` directory in sysfs.

About this task

The information is displayed in the format:

`0x0000000000aabbcc`

where:

aa kernel version

bb kernel patch level

cc kernel sublevel

Example

Linux kernel 3.5 displays as

```
# cat /sys/firmware/cpi/system_level
0x0000000000030500
```

Sending system data to the SE

Use the `set` attribute in the `/sys/firmware/cpi` directory in sysfs to send data to the service element.

About this task

To send the data in attributes `sysplex_name`, `system_level`, `system_name`, and `system_type` to the SE, write an arbitrary string to the set attribute.

Example

```
# echo 1 > /sys/firmware/cpi/set
```

Chapter 49. Activating automatic problem reporting

You can activate automatic problem reporting for situations where Linux experiences a kernel panic.

Before you begin

- The Linux instance must run in an LPAR.
- You need a hardware support agreement with IBM to report problems to RETAIN.
- The Linux kernel must be compiled with the SCLP_ASYNC device driver either as a module or built in.

About this task

Linux uses the Call Home function to send automatically collected problem data to the IBM service organization through the Service Element. Hence a system crash automatically leads to a new Problem Management Record (PMR) which can be processed by IBM service.

Building a kernel with the SCLP_ASYNC device driver

Control the build options for the SCLP_ASYNC device driver through the kernel configuration menu.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

You need to select the option CONFIG_SCLP_ASYNC in the Linux configuration menu to enable automatic problem reporting (see Figure 110).

```
Device Drivers --->
   ...
   Character devices --->
      ...
      --- S/390 character device drivers (depends on S390) ---
      ...
      Support for Call Home via Asynchronous SCLP Records   (CONFIG_SCLP_ASYNC)
```

Figure 110. Call Home kernel configuration menu option

The CONFIG_SCLP_ASYNC option can be compiled into the kernel or as a separate module, sclp_async.

Setting up the SCLP_ASYNC device driver

If the SCLP_ASYNC component is compiled as a separate module, you need to load it before you can work with it.

About this task

There are no kernel or module parameters for the SCLP_ASYNC device driver.

Procedure

Load the sclp_async module with the **modprobe** command to ensure that any other required modules are loaded in the correct order:

```
# modprobe sclp_async
```

Activating the Call Home support

When the SCLP_ASYNC device driver is compiled into the kernel or loaded as a module, you can control it through the sysctl interface or through procfs.

Procedure

To activate the support, set the `callhome` attribute to `1`. To deactivate the support, set the `callhome` attribute to `0`. Issue a command of this form:

```
# echo <flag> > /proc/sys/kernel/callhome
```

This command is equivalent to the following:

```
# sysctl -w  kernel.callhome=<flag>
```

Linux cannot check whether the Call Home function is supported by the hardware.

Examples

- To activate the Call Home support, issue:

  ```
  # echo 1 > /proc/sys/kernel/callhome
  ```

- To deactivate the Call Home support, issue:

  ```
  # echo 0 > /proc/sys/kernel/callhome
  ```

Chapter 50. Avoiding common pitfalls

Common problems and how to avoid them.

Ensuring correct channel path status

Ensure that you varied the path offline before you perform a planned task on it.

Tasks that require the channel path to be offline include:
- Pulling out or plugging in a cable on a path.
- Configuring a path off or on at the SE.

To vary the path offline, issue a command of the form:

```
# chchp -v 0 <chpid>
```

where *<chpid>* is the channel path ID.

After the operation completed and the path is available again, vary the path online by using a command of the form:

```
# chchp -v 1 <chpid>
```

Alternatively, you can write `on` or `off` to the channel path `status` attribute in sysfs to vary the path online or offline.

```
# echo on|off > /sys/devices/css0/chp0.<chpid>/status
```

An unplanned change in path availability can occur due to, for example, unplanned cable pulls or a temporary path malfunction. Then, the PIM/PAM/POM values (as obtained through **`lscss`**) might not be as expected. To update the PIM/PAM/POM values, vary one of the paths that lead to the affected devices.

Example:

```
# chchp -v 0 chchp -v 0 0.12
# chchp -v 1 chchp -v 0 0.12
```

Rationale: Linux does not always receive a notification (machine check) when the status of a path changes (especially for a path that comes online again). To make sure Linux has up-to-date information about the usable paths, path verification is triggered through the Linux vary operation.

Determining channel path usage

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 16 for details.

Configuring LPAR I/O devices

An Linux LPAR should contain only those I/O devices that it uses.

Limit the I/O devices 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. Use the cio_ignore kernel parameter or the `/proc/cio_ignore` dynamic control to remove devices, see "cio_ignore - List devices to be ignored" on page 704 and "Changing the exclusion list" on page 705.

Rationale: Numerous unused devices can cause:
- Unnecessary high memory usage due to allocation of device structures.
- Unnecessary high load on status changes because hot-plug handling must be done for every device found.

Using cio_ignore

With cio_ignore, essential devices might be hidden.

For example, if Linux does not boot under z/VM and does not show any message except:

```
HCPGIR450W CP entered; disabled wait PSW 00020001 80000000 00000000 00144D7A
```

Check if cio_ignore is used and verify that the console device, which is typically device number 0.0.0009, is not ignored.

Excessive guest swapping

Avoid excessive guest swapping by using the timed page pool size and the static page pool size attributes.

An instance of Linux on z/VM might be swapping and stalling. Setting the timed page pool size and the static page pool size to zero might solve the problem:

```
# echo 0 > /proc/sys/vm/cmm_timed_pages
# echo 0 > /proc/sys/vm/cmm_pages
```

If you see a temporary relief, the guest does not have enough memory. Try increasing the guest memory.

If the problem persists, z/VM might be out of memory.

If you are using cooperative memory management (CMM), unload the cooperative memory management module:

```
# modprobe -r cmm
```

See Chapter 41, "Cooperative memory management," on page 475 for more details about CMM.

Including service levels of the hardware and the hypervisor

The service levels of the different hardware cards, the LPAR level, and the z/VM service level are valuable information for problem analysis.

If possible, include this information with any problem you report to IBM service.

A `/proc` interface that provides a list of service levels is available. To see the service levels issue:

```
# cat /proc/service_levels
```

Example for a z/VM system with a QETH adapter:

```
# cat /proc/service_levels
VM: z/VM Version 5 Release 2.0, service level 0801 (64-bit)
qeth: 0.0.f5f0 firmware level 087d
```

Booting stops with disabled wait state

An automatic processor type check might stop the boot process with a disabled wait PSW.

On some distributions, a processor type check is automatically run at every kernel startup. If the check determines that the distribution used is not compatible with the hardware, it stops the boot process with a disabled wait PSW.

If this problem occurs, ensure that you are using a distribution that is supported on your hardware.

If you are using an SCLP console, you might get a message that indicates the problem.

Preparing for dump-on-panic

You might want to consider setting up your system to automatically create a memory dump after a kernel panic.

Configuring and using dump-on-panic is a good idea for several reasons:
- You have a memory dump disk that is prepared ahead of time.
- You do not have to reproduce the problem since a memory dump will be triggered automatically immediately after the failure.

See Chapter 8, "Shutdown actions," on page 117 for details.

Chapter 51. Kernel messages

System z specific kernel modules issue messages on the console and write them to the syslog. You can configure your system to issue these messages with message numbers.

Based on these message numbers, you can create man pages that users can call to obtain message details.

The message numbers consist of a module identifier, a dot, and a hash value that is generated from the message text. For example, `xpram.ab9aa4` is a message number.

Kernel Messages, SC34-2599 summarizes the messages that are issued by the System z specific kernel modules. You can find this documentation on developerWorks at

www.ibm.com/developerworks/linux/linux390/documentation_dev.html

| and on the IBM knowledge Center at

| ibm.com/support/knowledgecenter/linuxonibm/com.ibm.linux.l0kmsg.doc/
| l0km_plugin_top.html

Note: Some messages are issued with message numbers although there is no message explanation. These messages are considered self-explanatory and they are not included in this documentation. If you find an undocumented message with a message text that needs further explanation, complete a Readers' Comment Form or send a request to eservdoc@de.ibm.com.

Building a kernel with message documentation support

Select option CONFIG_KMSG_IDS in the Linux configuration menu to include message documentation support.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

You need to patch the 3.16 kernel source with additions from developerWorks at www.ibm.com/developerworks/linux/linux390/kernel.html to include message documentation support.

With the kernel patches in place, you can find option CONFIG_KMSG_IDS as the menu item **Kernel message numbers** at the top level of the kernel configuration menu.

Generating the message man pages

You can generate the message man pages from the root of the Linux source tree after you compile the kernel.

About this task

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

Procedure

Generate the man pages by entering:

```
# make D=2
```

After you run this command, you will find a man page for each message in a `man` directory that is located in the root of your Linux source tree.

Displaying a message man page

Man page names for System z specific kernel messages match the corresponding message numbers.

Before you begin

Copy the message man pages as generated in "Generating the message man pages" on page 535 to `/man/man9`.

Procedure

For example, the following message has the message number `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_number>
```

Example

Enter the following command to display the man page for message `xpram.ab9aa4`:

```
# man xpram.ab9aa4
```

The corresponding man page looks like this example:

```
xpram.ab9aa4(9)                                              xpram.ab9aa4(9)

Message
       xpram.ab9aa4: %d 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 parame-
       ter 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 par-
       titions that are probably not configured as intended.

User action
       If the XPRAM device driver has been  complied  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)
```

Part 10. Reference

Use these commands, kernel parameters, kernel options to configure Linux on System z. Be aware of the z/VM DIAG calls required by Linux on System z.

Newest version

You can find the newest version of this publication at www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Restrictions

For prerequisites and restrictions see
www.ibm.com/developerworks/linux/linux390/development_technical.html
www.ibm.com/developerworks/linux/linux390/development_restrictions.html

Chapter 52. Commands for Linux on System z

You can use System z specific commands to configure and work with the Linux on System z device drivers and features.

Most of the commands described in this section are included in the s390-tools package (version 1.26) that is available at www.ibm.com/developerworks/linux/linux390/s390-tools.html.

Some commands come with an init script or a configuration file or both. It is assumed that init scripts are installed in `/etc/init.d/` and configuration files are installed in `/etc/`, but this might vary depending on your distribution. You can extract any missing files from the `etc` subdirectory in the s390-tools package.

Commands described elsewhere

- For the **`zipl`** command, see Chapter 5, "Initial program loader for System z - zipl," on page 65.
- For the **`snipl`** command, see Chapter 9, "Remotely controlling virtual hardware - snipl," on page 121.
- For commands and tools related to creating and analyzing system dumps, see *Using the Dump Tools*, SC33-8412.
- For commands related to terminal access over IUCV connections, see *How to Set up a Terminal Server Environment on z/VM*, SC34-2596.
- The **`icainfo`** and **`icastats`** commands are provided with the libica package and described in *libica Programmer's Reference*, SC34-2602.

Generic command options

There are common command options that, for simplicity, have been omitted from some of the syntax diagrams.

`-h or --help`
to display help information for the command.

`--version`
to display version information for the command.

The syntax for these options is:

Common command options

```
►►──<command>──┬─┤ Other command options ├─┬──────────────────►◄
               ├─-h────────────────────────┤
               ├─--help────────────────────┤
               └─--version─────────────────┘
```

where command can be any of the commands described in this section.

See Appendix B, “Understanding syntax diagrams,” on page 745 for general information about reading syntax diagrams.

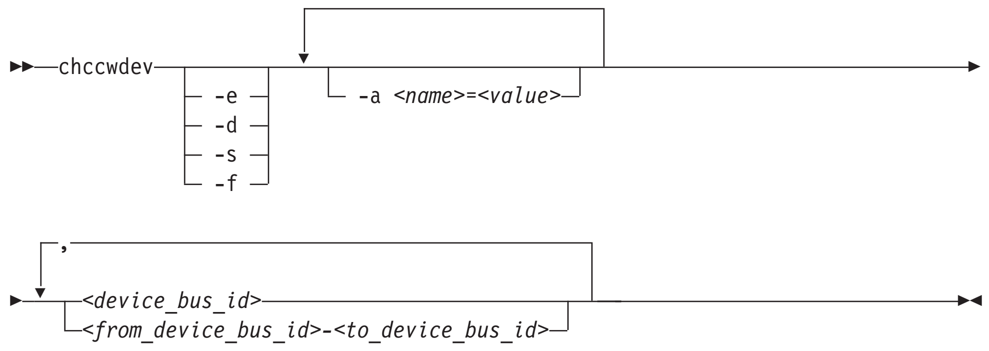
chccwdev - Set CCW device attributes

Use the **chccwdev** command to set attributes for CCW devices and to set CCW devices online or offline.

Use “znetconf - List and configure network devices” on page 699 to work with CCW_GROUP devices. For more information about CCW devices and CCW group devices, see “Device categories” on page 9.

The **chccwdev** command uses cio_settle before it changes anything to ensure that sysfs reflects the latest device status information and includes newly available devices.

chccwdev syntax



Where:

-e or --online
sets the device online.

-d or --offline
sets the device offline.

-s or --safeoffline
waits until all outstanding I/O requests complete, and then tries to set the device offline. Valid for DASDs only.

-f or --forceonline
forces a boxed device online, if this action is supported by the device driver.

-a or --attribute *<name>*=*<value>*
sets the *<name>* attribute to *<value>*.

The available attributes depend on the device type. See the chapter for your device for details about the applicable attributes and values.

Setting the `online` attribute has the same effect as using the **-e** or **-d** options.

<device_bus_id>
identifies a device. Device bus-IDs are of the form 0.*<n>*.*<devno>*, where *<n>* is a subchannel set ID and *<devno>* is a device number. Input is converted to lowercase.

<from_device_bus_id>-<to_device_bus_id>
identifies a range of devices. If not all devices in the given range exist, the command is limited to the existing ones. If you specify a range with no existing devices, you get an error message.

-h or --help
displays help information for the command. To view the man page, enter **man chccwdev**.

-v or --version
displays version information for the command.

Examples

- To set a CCW device 0.0.b100 online issue:

```
# chccwdev -e 0.0.b100
```

- Alternatively, use **-a** to set a CCW device 0.0.b100 online. Issue:

```
# chccwdev -a online=1 0.0.b100
```

- To set all CCW devices in the range 0.0.b200 through 0.0.b2ff online, issue:

```
# chccwdev -e 0.0.b200-0.0.b2ff
```

- To set a CCW device 0.0.b100 and all CCW devices in the range 0.0.b200 through 0.0.b2ff offline, issue:

```
# chccwdev -d 0.0.b100,0.0.b200-0.0.b2ff
```

- To set several CCW devices in different ranges and different subchannel sets offline, issue:

```
# chccwdev -d 0.0.1000-0.0.1100,0.1.7000-0.1.7010,0.0.1234,0.1.4321
```

- To set devices with bus ID 0.0.0192, and 0.0.0195 through 0.0.0198 offline after completing all outstanding I/O requests:

```
# chccwdev -s 0.0.0192,0.0.0195-0.0.0198
```

  If an outstanding I/O request is blocked, the command might wait forever. Reasons for blocked I/O requests include reserved devices that can be released or disconnected devices that can be reconnected.

  1. Try to resolve the problem that blocks the I/O request and wait for the command to complete.
  2. If you cannot resolve the problem, issue **chccwdev -d** to cancel the outstanding I/O requests. The data is lost.

- To set an ECKD DASD 0.0.b100 online and to enable extended error reporting and logging issue:

```
# chccwdev -e -a eer_enabled=1 -a erplog=1 0.0.b100
```

chchp - Change channel path status

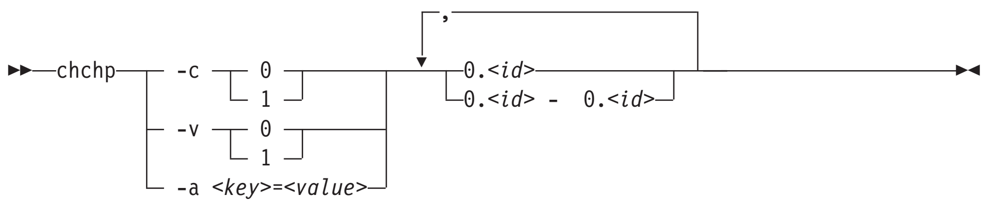
Use the **chchp** command to set channel paths online or offline.

The actions are equivalent to performing a Configure Channel Path Off or Configure Channel Path On operation on the Hardware Management Console.

The channel path status that results from a configure operation is persistent across IPLs.

Note: Changing the configuration state of an I/O channel path might affect the availability of I/O devices. It can also trigger associated functions (such as channel-path verification or device scanning), which in turn can result in a temporary increase in processor, memory, and I/O load.

chchp syntax



Where:

-c or --configure *<value>*
: sets the device to configured (1) or standby (0).

  Note: Setting the configured state to standby can stop running I/O operations.

-v or --vary *<value>*
: changes the logical channel-path state to online (1) or offline (0).

  Note: Setting the logical state to offline can stop running I/O operations.

-a or --attribute *<key>*=*<value>*
: changes the channel-path sysfs attribute *<key>* to *<value>*. The *<key>* can be the name of any available channel-path sysfs attribute (that is, `configure` or `status`). *<value>* can take any valid value that can be written to the attribute (for example, `0` or `offline`). Using `-a` is a generic way of writing to the corresponding sysfs attribute. It is intended for cases where sysfs attributes or attribute values are available in the kernel but not in **chchp**.

0.*<id>* **and 0.***<id>* **- 0.***<id>*
: where *<id>* is a hexadecimal, two-digit, lowercase identifier for the channel path. An operation can be performed on more than one channel path by specifying multiple identifiers as a comma-separated list, or a range, or a combination of both.

--version
: displays the version number of **chchp** and exits.

-h or --help
displays a short help text. To view the man page, enter **man chchp**.

Examples

- To set channel path 0.19 into standby state issue:

```
# chchp -a configure=0 0.19
```

- To set the channel path with the channel path ID 0.40 to the standby state, write `0` to the configure file with the **chchp** command:

```
# chchp --configure 0 0.40
Configure standby 0.40... done.
```

- To set a channel-path to the configured state, write 1 to the configure file with the **chchp** command:

```
# chchp --configure 1 0.40
Configure online 0.40... done.
```

- To set channel-paths 0.65 to 0.6f to the configured state issue:

```
# chchp -c 1 0.65-0.6f
```

- To set channel-paths 0.12, 0.7f and 0.17 to 0.20 to the logical offline state issue:

```
# chchp -v 0 0.12,0.7f,0.17-0.20
```

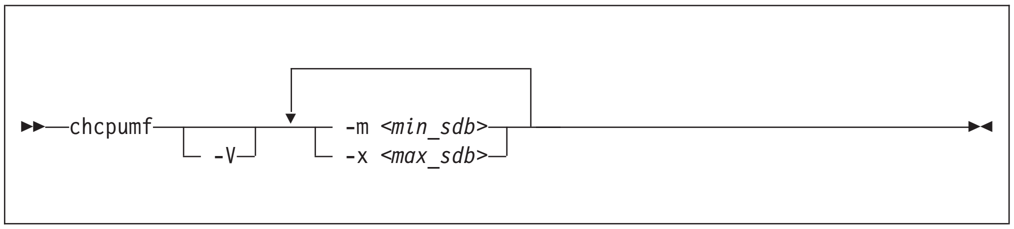
chcpumf - Set limits for the CPU measurement sampling facility buffer

Use the **chcpumf** command to set limits for the CPU measurement sampling facility buffer.

The sampling facility is designed for autonomous buffer management, and you do not usually need to intervene. However, you might want to change the minimum or maximum size, for example, for one of the following reasons:

- There are considerable resource constraints on your system, and the sampling facility stops because it tries to allocate more buffer space than is available.
- As an expert user of perf and the sampling facility, you want to explore results with particular buffer settings.

chcpumf syntax



where:

-m *<min_sdb>* **or --min** *<min_sdb>*
: specifies the minimum sampling facility buffer size in sample-data-blocks. A sample-data-block occupies approximately 4 KB. The sampling facility starts with this buffer size if it exceeds the initial buffer size that is calculated by the sampling facility.

-x *<max_sdb>* **or --max** *<max_sdb>*
: specifies the maximum sampling facility buffer size in sample-data-blocks. A sample-data-block occupies approximately 4 KB. While it is running, the sampling facility dynamically adjusts the buffer size to a suitable value, but cannot exceed this limit.

-V or --verbose
: displays the buffer size settings after the changes.

-v or --version
: displays the version number of **chcpumf** and exits.

-h or --help
: displays out a short help text, then exits. To view the man page, enter **man chcpumf**.

Example

To change the minimum buffer size to 500 times the size of a sample-data-block and the maximum buffer size to 1000 times the size of a sample-data-block , issue:

```
# chcpufm -V -m 500 -x 1000
Sampling buffer sizes:
    Minimum:    500 sample-data-blocks
    Maximum:   1000 sample-data-blocks
```

chmem - Set memory online or offline

Use the **chmem** command to set a particular size or range of memory online or offline.

Setting memory online can fail if the hypervisor does not have enough memory left, for example because memory was overcommitted. Setting memory offline can fail if Linux cannot free the memory. If only part of the requested memory can be set online or offline, a message informs you how much memory was set online or offline instead of the requested amount.

chmem syntax

```
►►──chmem──┬─ -e─┬──┬─ <size>──────────┬───────────────────────────────────────►◄
           └─ -d─┘  └─ <start>-<end>───┘
```

Where:

-e or --enable
: sets the specified memory online.

-d or --disable
: sets the specified memory offline.

<size>
: specifies an amount of memory to be set online or offline. A numeric value without a unit or a numeric value immediately followed by `m` or `M` is interpreted as MB (1024 x 1024 bytes). A numeric value immediately followed by `g` or `G` is interpreted as GB (1024 x 1024 x 1024 bytes).

  The size must be aligned to the memory block size, as shown in the output of the **lsmem** command.

<start>-<end>
: specifies a memory range to be set online or offline. *<start>* is the hexadecimal address of the first byte and *<end>* is the hexadecimal address of the last byte in the memory range.

  The range must be aligned to the memory block size, as shown in the output of the **lsmem** command.

-v or --version
: displays the version number of **chmem**, then exits.

-h or --help
: displays a short help text, then exits. To view the man page, enter **man chmem**.

Examples

- This command requests 1024 MB of memory to be set online.

```
# chmem --enable 1024
```

- This command requests 2 GB of memory to be set online.

```
# chmem --enable 2g
```

- This command requests the memory range that starts with `0x00000000e4000000` and ends with `0x00000000f3ffffff` to be set offline.

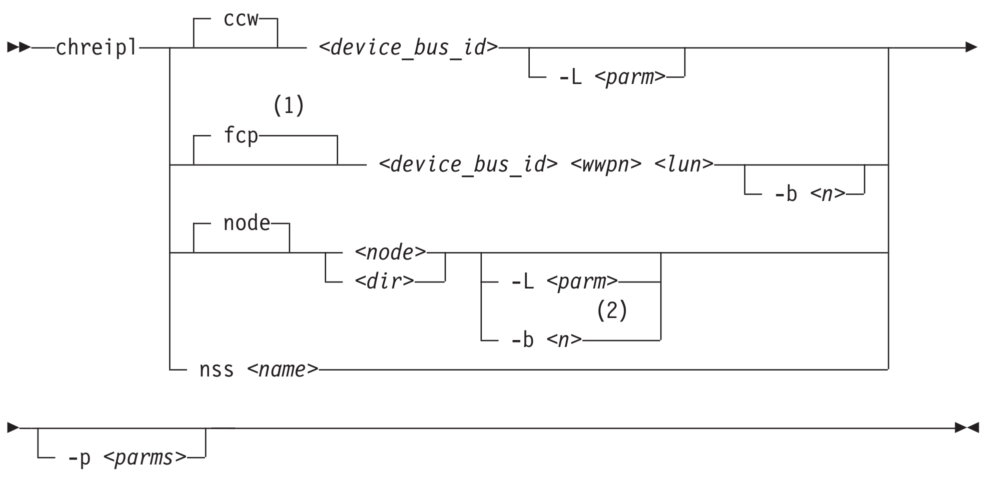
```
# chmem --disable 0x00000000e4000000-0x00000000f3ffffff
```

chreipl - Modify the re-IPL configuration

Use the **chreipl** tool to modify the re-IPL configuration for Linux on System z.

You can configure a particular device as the reboot device. For **zipl** boot menu configurations, you can set the boot menu entry to be used for the next reboot. You can also specify additional kernel parameters for the next reboot.

chreipl syntax



Notes:

1 You can specify the *<device_bus_id>*, *<wwpn>*, and *<lun>* in any order if you use the corresponding command options.

2 **-L** can be used if the device node or directory maps to a DASD. **-b** can be used if the device node or directory maps to a SCSI disk.

Where:

<device_bus_id> **or -d** *<device_bus_id>* **or --device** *<device_bus_id>*
: specifies the device bus-ID of a CCW re-IPL device or of the FCP device through with a SCSI re-IPL device is attached.

<wwpn> **or -w** *<wwpn>* **or --wwpn** *<wwpn>*
: specifies the worldwide port name (WWPN) of a SCSI re-IPL device.

<lun> **or -l** *<lun>* **or --lun** *<lun>*
: specifies the logical unit number (LUN) of a SCSI re-IPL device.

<node>
: specifies a device node of a DASD, SCSI, or logical device mapper re-IPL device. For more information about logical boot devices, see "Preparing a logical device as a boot device" on page 71.

<dir>
: specifies a directory in the Linux file system on the re-IPL device.

nss
: declares that the following parameters refer to a z/VM named saved system (NSS).

<name> **or -n** *<name>* **or --name** *<name>*
: specifies the name of an NSS as defined on the z/VM system.

-L or --loadparm *<parameter>*
: specifies the entry in the boot menu to be used for the next reboot. This parameter applies only if the re-IPL device is a DASD with a **zipl** boot menu configuration.

  Omitting this parameter eliminates an existing selection in the boot configuration. Depending on your boot menu configuration, a **zipl** interactive boot menu might be displayed during the re-IPL process or the default configuration is used. See "Example for a DASD menu configuration on z/VM" on page 94, "Example for a DASD menu configuration (LPAR)" on page 101, and "Menu configurations" on page 84 for details.

-b or --bootprog *<n>*
: specifies the entry in the boot menu to be used for the next reboot. This parameter applies only if the re-IPL device is a SCSI disk with a **zipl** boot menu configuration.

  Omitting this parameter eliminates an existing selection in the boot configuration and the default boot configuration is used.

-p or --bootparms
: specifies boot parameters for the next reboot. The boot parameters, which typically are kernel parameters, are appended to the kernel parameter line in the boot configuration. The number of characters you can specify depends on your environment and re-IPL device as shown in Table 61.

Table 61. Maximum characters for additional kernel parameters

Virtual hardware where Linux runs	DASD re-IPL device	SCSI re-IPL device	NSS re-IPL device
z/VM guest virtual machine	64	3452	56
LPAR	none	3452	n/a

  If you omit this parameter, the existing boot parameters in the next boot configuration are used without any changes.

-h or --help
: displays help information for the command. To view the man page, enter **man chreipl**.

-v or --version
: displays version information.

For disk-type re-IPL devices, the command accepts but does not require an initial statement:

ccw
: declares that the following parameters refer to a DASD re-IPL device.

fcp
: declares that the following parameters refer to a SCSI re-IPL device.

node
: declares that the following parameters refer to a disk re-IPL device that is

identified by a device node or by a directory in the Linux file system on that device. The disk device can be a DASD or a SCSI disk.

Examples

These examples illustrate common uses for **chreipl**.

- The following commands all configure the same DASD as the re-IPL device, assuming that the device bus-ID of the DASD is `0.0.7e78`, that the standard device node is `/dev/dasdc`, that udev creates an alternative device node `/dev/disk/by-path/ccw-0.0.7e78`, that `/mnt/boot` is located on the Linux file system in a partition of the DASD.
  - Using the bus ID:
    ```
    # chreipl 0.0.7e78
    ```
  - Using the bus ID and the optional `ccw` statement:
    ```
    # chreipl ccw 0.0.7e78
    ```
  - Using the bus ID, the optional statement and the optional **`--device`** keyword:
    ```
    # chreipl ccw --device 0.0.7e78
    ```
  - Using the standard device node:
    ```
    # chreipl /dev/dasdc
    ```
  - Using the udev-created device node:
    ```
    # chreipl /dev/disk/by-path/ccw-0.0.7e78
    ```
  - Using a directory within the file system on the DASD:
    ```
    # chreipl /mnt/boot
    ```
- The following commands all configure the same SCSI disk as the re-IPL device, assuming that the device bus-ID of the FCP device through which the device is attached is `0.0.1700`, the WWPN of the storage server is `0x500507630300c562`, and the LUN is `0x401040b300000000`. Further it is assumed that the standard device node is `/dev/sdb`, that udev creates an alternative device node `/dev/disk/by-id/scsi-36005076303ffc56200000000000010b4`, and that `/mnt/fcpboot` is located on the Linux file system in a partition of the SCSI disk.
  - Using bus ID, WWPN, and LUN:
    ```
    # chreipl 0.0.1700 0x500507630300c562 0x401040b300000000
    ```
  - Using bus ID, WWPN, and LUN with the optional `fcp` statement:
    ```
    # chreipl fcp 0.0.1700 0x500507630300c562 0x401040b300000000
    ```
  - Using bus ID, WWPN, LUN, the optional statement, and keywords for the parameters. When you use the keywords, the parameters can be specified in any order:
    ```
    # chreipl fcp --wwpn 0x500507630300c562 -d 0.0.1700 --lun 0x401040b300000000
    ```
  - Using the standard device node:

```
# chreipl /dev/sdb
```

– Using the udev-created device node:

```
# chreipl /dev/disk/by-id/scsi-36005076303ffc56200000000000010b4
```

– Using a directory within the file system on the SCSI disk:

```
# chreipl /mnt/fcpboot
```

- To configure a DASD with bus ID `0.0.7e78` as the re-IPL device, using the first entry of the **`zipl`** boot menu:

```
# chreipl 0.0.7e78 -L 1
Re-IPL type:  ccw
Device:       0.0.7e78
Loadparm:     "1"
Bootparms:    ""
```

- To configure a DASD with bus ID `0.0.7e78` as the re-IPL device and adding `mem=512M` to the existing kernel parameters in the boot configuration:

```
# chreipl 0.0.7e78 -p "mem=512M"
Re-IPL type:  ccw
Device:       0.0.7e78
Loadparm:     ""
Bootparms:    "mem=512M"
```

- To configure an NSS `LINUX1` as the re-IPL device:

```
# chreipl nss LINUX1
```

chshut - Control the system shutdown actions

Use the **chshut** command to change the shutdown actions for specific shutdown triggers.

The shutdown triggers are:
- halt
- poff
- reboot

The shutdown triggers restart and panic are handled by the dumpconf service script, see *Using the Dump Tools*, SC33-8412 for details.

Linux on System z performs shutdown actions according to sysfs attribute settings within the /sys/firmware directory structure. The **chshut** command sets a shutdown action for a shutdown trigger by changing the corresponding sysfs attribute setting. For more information about the sysfs attributes and the shutdown actions, see Chapter 8, “Shutdown actions,” on page 117.

chshut syntax

```
►►──chshut──┬─ halt───┬──┬─ ipl──────────────────────┬───────────────────►◄
            ├─ poff───┤  ├─ reipl────────────────────┤
            └─ reboot─┘  ├─ stop─────────────────────┤
                         │  ┌──────────────────────┐ │
                         │  ▼                      │ │
                         └──── vmcmd "<cp_command>"─┴─┘
```

Where:

halt
sets an action for the halt shutdown trigger.

If your distribution maps halt to power off, set the poff shutdown action instead of the halt action.

poff
sets an action for the poff shutdown trigger.

reboot
sets an action for the reboot shutdown trigger.

ipl
sets IPL as the action to be taken.

reipl
sets re-IPL as the action to be taken.

stop
sets “stop” as the action to be taken.

vmcmd "<*cp_command*>"
sets the action to be taken to issuing a z/VM CP command. The command must be specified in uppercase characters and enclosed in quotation marks. To issue multiple commands, repeat the vmcmd attribute with each command.

-h or --help
displays help information for the command. To view the man page, enter **man chshut**.

-v or --version
displays version information.

Examples

These examples illustrate common uses for **chshut**.

- To make the system start again after a power off:

```
# chshut poff ipl
```

- To log off the z/VM guest virtual machine if the Linux **poweroff** command was run successfully:

```
# chshut poff vmcmd LOGOFF
```

- To send a message to z/VM user ID OPERATOR and automatically log off the z/VM guest virtual machine if the Linux **poweroff** command is run:

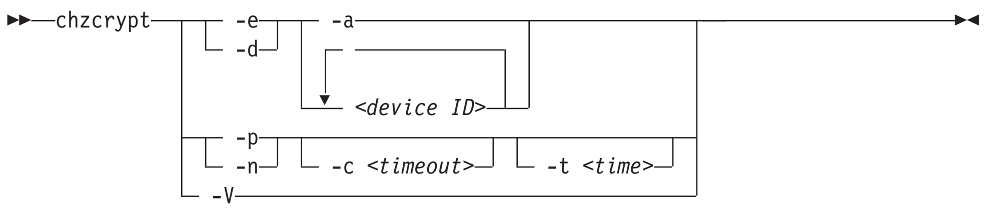
```
# chshut poff vmcmd "MSG OPERATOR Going down" vmcmd "LOGOFF"
```

chzcrypt - Modify the zcrypt configuration

Use the **chzcrypt** command to configure cryptographic adapters that are managed by zcrypt and modify zcrypt's AP bus attributes.

To display the attributes, use "lszcrypt - Display zcrypt devices" on page 637.

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 that is to be set online or offline. A cryptographic adapter can be specified either in decimal notation or hexadecimal notation with 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 rescanning the AP bus to <*timeout*> seconds.

-t *<time>***or --poll-timeout=***<time>*
: sets the high-resolution polling timer to <*time*> nanoseconds. To display the value, use **lszcrypt -b**.

-V or --verbose
: displays verbose messages.

-h or --help
: displays help information for the command. To view the man page, enter **man chzcrypt**.

-v or --version
: displays version information.

Examples

These examples illustrate common uses for **chzcrypt**.

- To set the cryptographic adapters 0, 1, 4, 5, and 12 online (in decimal notation):

```
chzcrypt -e 0 1 4 5 12
```

- To set all available cryptographic adapters offline:

```
chzcrypt -d -a
```

- To set the configuration timer for rescanning the AP bus to 60 seconds and disable zcrypt's poll thread:

```
chzcrypt -c 60 -n
```

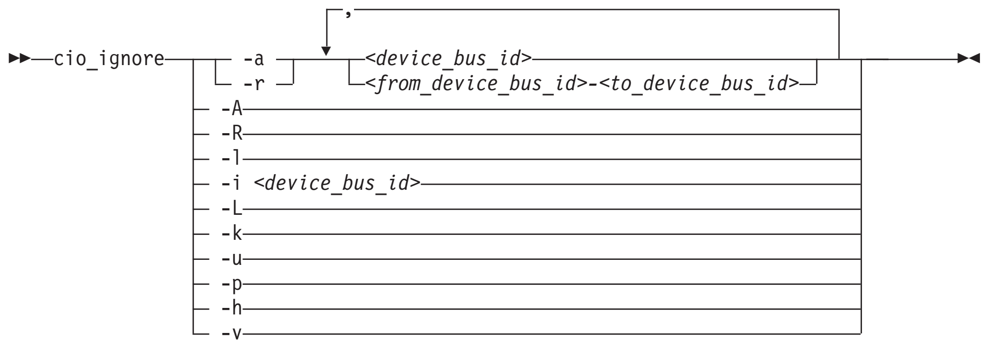
cio_ignore - Manage the I/O exclusion list

Use the **cio_ignore** command to specify I/O devices that are to be ignored by Linux.

When a Linux on System z instance boots, it senses and analyzes all available I/O devices. You can use the cio_ignore kernel parameter (see “cio_ignore - List devices to be ignored” on page 704) to specify devices that are to be ignored. This exclusion list can cover all possible devices, even devices that do not actually exist.

The **cio_ignore** command manages this exclusion list on a running Linux instance. You can change the exclusion list and display it in different formats.

cio_ignore syntax



Where:

-a or --add
: adds one or more device specifications to the exclusion list.

When you add specifications for a device that is already sensed and analyzed, there is no immediate effect of adding it to the exclusion list. For example, the device still appears in the output of the **lscss** command and can be set online. However, if the device subsequently becomes unavailable, it is ignored when it reappears. For example, if the device is detached in z/VM, it is ignored when it is attached again.

See the **-p** option about making devices that are already sensed and analyzed unavailable to Linux.

-r or --remove
: removes one or more device specifications from the exclusion list.

When you remove device specifications from the exclusion list, the corresponding devices are sensed and analyzed if they exist. Where possible, the corresponding device driver is informed, and the devices become available to Linux.

<device_bus_id>
: identifies a single device.

Device bus-IDs are of the form 0.<*n*>.<*devno*>, where <*n*> is a subchannel set ID and <*devno*> is a device number. If the subchannel set ID is 0, you can abbreviate the specification to the device number, with or without a leading 0x.

Example: The specifications 0.0.0190, 190, 0190, and 0x190 are all equivalent. There is no short form of 0.1.0190.

`<from_device_bus_id>-<to_device_bus_id>`
identifies a range of devices. <*from_device_bus_id*> and <*to_device_bus_id*> have the same format as <*device_bus_id*>.

`-A or --add-all`
adds the entire range of possible devices to the exclusion list.

When you add specifications for a device that is already sensed and analyzed, there is no immediate effect of adding it to the exclusion list. For example, the device still appears in the output of the **`lscss`** command and can be set online. However, if the device subsequently becomes unavailable, it is ignored when it reappears. For example, if the device is detached in z/VM, it is ignored when it is attached again.

See the **`-p`** option about making devices that are already sensed and analyzed unavailable to Linux.

`-R or --remove-all`
removes all devices from the exclusion list.

When you remove device specifications from the exclusion list, the corresponding devices are sensed and analyzed if they exist. Where possible, the corresponding device driver is informed, and the devices become available to Linux.

`-l or --list`
displays the current exclusion list.

`-i or --is-ignored`
checks if the specified device is on the exclusion list. The command prints an information message and completes with exit code `0` if the device is on the exclusion list. The command completes with exit code `2` if the device is not on the exclusion list.

`-L or --list-not-blacklisted`
displays specifications for all devices that are not in the current exclusion list.

`-k or --kernel-param`
returns the current exclusion list in kernel parameter format.

You can make the current exclusion list persistent across rebooting Linux by using the output of the **`cio_ignore`** command with the **`-k`** option as part of the Linux kernel parameter. See Chapter 3, "Kernel and module parameters," on page 25.

`-u or --unused`
discards the current exclusion list and replaces it with a specification for all devices that are not online. This includes specification for possible devices that do not actually exist.

`-p or --purge`
makes all devices that are in the exclusion list and that are currently offline unavailable to Linux. This option does not make devices unavailable if they are online.

`-h or --help`
displays help information for the command. To view the man page, enter **`man cio_ignore`**.

-v or --version
displays version information.

Examples

These examples illustrate common uses for **cio_ignore**.

- The following command shows the current exclusion list:

```
# cio_ignore -l
Ignored devices:
=================
0.0.0000-0.0.7e8e
0.0.7e94-0.0.f4ff
0.0.f503-0.0.ffff
0.1.0000-0.1.ffff
0.2.0000-0.2.ffff
0.3.0000-0.3.ffff
```

- The following command shows specifications for the devices that are not on the exclusion list:

```
# cio_ignore -L
Accessible devices:
===================
0.0.7e8f-0.0.7e93
0.0.f500-0.0.f502
```

  The following command checks if `0.0.7e8f` is on the exclusion list:

```
# cio_ignore -i 0.0.7e8f
Device 0.0.7e8f is not ignored.
```

- The following command adds, `0.0.7e8f`, to the exclusion list:

```
# cio_ignore -a 0.0.7e8f
```

  The previous example then becomes:

```
# cio_ignore -L
Accessible devices:
===================
0.0.7e90-0.0.7e93
0.0.f500-0.0.f502
```

  And for `0.0.7e8f` in particular:

```
# cio_ignore -i 0.0.7e8f
Device 0.0.7e8f is ignored.
```

- The following command shows the current exclusion list in kernel parameter format:

```
# cio_ignore -k
cio_ignore=all,!7e90-7e93,!f500-f502
```

cmsfs-fuse - Mount a z/VM CMS file system

Use the **cmsfs-fuse** command to mount the enhanced disk format (EDF) file system on a z/VM minidisk.

In Linux, the minidisk is represented as a DASD and the file system is mounted as a cmsfs-fuse file system. The cmsfs-fuse file system translates the record-based file system on the minidisk into Linux semantics.

Through the cmsfs-fuse file system, the files on the minidisk become available to applications on Linux. Applications can read from and write to files on minidisks. Optionally, the cmsfs-fuse file system converts text files between EBCDIC on the minidisk and ASCII within Linux.

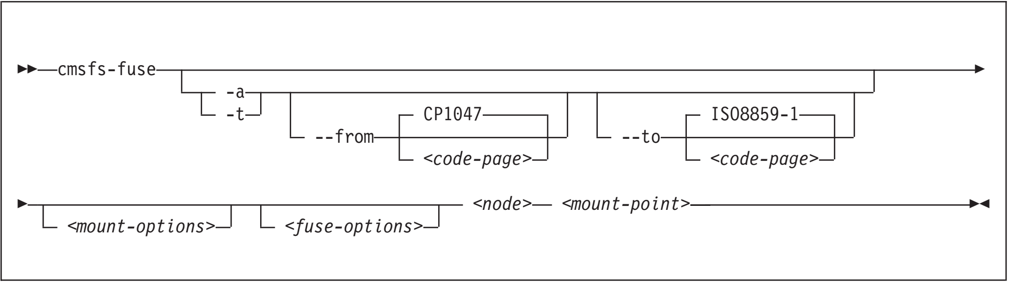
Attention: You can inadvertently damage files and lose data when directly writing to files within the cmsfs-fuse file system. To avoid problems when you write, multiple restrictions must be observed, especially regarding linefeeds (see restrictions for write).

Tip: If you are unsure about how to safely write to a file on the cmsfs-fuse file system, copy the file to a location outside the cmsfs-fuse file system, edit the file, and then copy it back to its original location.

Use **fusermount** to unmount file systems that you mounted with **cmsfs-fuse**. See the **fusermount** man page for details.

Before you begin:
- You need a kernel that was built with the common code option CONFIG_FUSE_FS to include FUSE support.
- FUSE support must be compiled into the kernel or the fuse module must be loaded, for example, with **modprobe fuse**.
- The FUSE library must be installed on your system. If the library is not included in your distribution, you can obtain it from sourceforge at sourceforge.net/projects/fuse.
- The DASD must be online.
- Depending whether you intend to read, write, or both, you must have the appropriate permissions for the device node.

cmsfs-fuse syntax



where:

-a or --ascii
treats all files on the minidisk as text files and converts them from EBCDIC to ASCII.

-t or --filetype
treats files with extensions as listed in the **cmsfs-fuse** configuration file as text files and converts them from EBCDIC to ASCII.

By default, the cmsfs-fuse command uses `/etc/cmsfs-fuse/filetypes.conf` as the configuration file. You can replace the list in this default file by creating a file `.cmsfs-fuse/filetypes.conf` in your home directory.

The `filetypes.conf` file lists one file type per line. Lines that start with a number sign (#) followed by a space are treated as comments and are ignored.

--from *<code-page>*
specifies the encoding of the files on the z/VM minidisk. If this option is not specified, code page CP1047 is used. Enter **iconv --list** to display a list of all available code pages.

--to *<code-page>*
specifies the encoding to which the files on the z/VM minidisk are converted in Linux. If this option is not specified, code page ISO-8859-1 is used. Enter **iconv --list** to display a list of all available code pages.

<mount-options>
options as available for the **mount** command. See the **mount** man page for details.

<fuse-options>
options for FUSE. The following options are supported by the **cmsfs-fuse** command. To use an option, it must also be supported by the version of FUSE that you have.

-d or -o debug
enables debug output (implies **-f**).

-f runs the command as a foreground operation.

-o allow_other
allows access to other users.

-o allow_root
allows access to root.

-o nonempty
allows mounts over files and non-empty directories.

-o default_permissions
enables permission checking by the kernel.

-o max_read=*<n>*
sets maximum size of read requests.

-o kernel_cache
caches files in the kernel.

-o [no]auto_cache
enables or disables caching based on modification times.

-o umask=*<mask>*
sets file permissions (octal).

-o uid=*<n>*
sets the file owner.

-o gid=*<n>*
sets the file group.

-o max_write=*<n>*
sets the maximum size of write requests.

-o max_readahead=*<n>*
sets the maximum readahead value.

-o async_read
performs reads asynchronously (default).

-o sync_read
performs reads synchronously.

-o big_writes
enables write operations with more than 4 KB.

<node>
the device node for the DASD that represents the minidisk in Linux.

<mount-point>
the mount point in the Linux file system where you want to mount the CMS file system.

-h or --help
displays help information for the command. To view the man page, enter **man cmsfs-fuse**.

-v or --version
displays version information for the command.

Extended attributes

You can use the following extended attributes to handle the CMS characteristics of a file:

user.record_format
specifies the format of the file. The format is F for fixed record length files and V for variable record length files. This attribute can be set only for empty files. The default file format for new files is V.

user.record_lrecl
specifies the record length of the file. This attribute can be set only for an empty fixed record length file. A valid record length is an integer in the range 1-65535.

user.file_mode
specifies the CMS file mode of the file. The file mode consists of a mode letter from A-Z and mode number from 0 - 6. The default file mode for new files is A1.

You can use the following system calls to work with extended attributes:

listxattr
to list the current values of all extended attributes.

getxattr
to read the current value of a particular extended attribute.

setxattr
to set a particular extended attribute.

You can use these system calls through the **getfattr** and **setfattr** commands. For more information, see the man pages of these commands and of the listxattr, getxattr, and setxattr system calls.

Restrictions

When you work with files in the cmsfs-fuse file system, restrictions apply for the following system calls:

`write`
Be aware of the following restrictions when you write to a file on the cmsfs-fuse file system:

Write location
Writing is supported only at the end of a file.

Padding
For fixed-length record files, the last record is padded to make up a full record length. The padding character is zero in binary mode and the space character in ASCII mode.

Sparse files
Sparse files are not supported. To prevent the **`cp`** tool from writing in sparse mode, specify `-sparse=never`.

Records and linefeeds with ASCII conversion (-a and -t)

In the ASCII representation of an EBCDIC file, a linefeed character determines the end of a record. Follow these rules about linefeed characters requirements when you write to EBCDIC files in ASCII mode:

For fixed record length files
Use linefeed characters to separate character strings of the fixed record length.

For variable record length files
Use linefeed characters to separate character strings. The character strings must not exceed the maximum record length.

The CMS file system does not support empty records. cmsfs-fuse adds a space to records that consist of a linefeed character only.

`rename` and `creat`
Uppercase file names are enforced.

`truncate`
Only shrinking of a file is supported. For fixed-length record files, the new file size must be a multiple of the record length.

Examples

- To mount the CMS file system on the minidisk represented by the file node `/dev/dasde` at `/mnt`:

```
# cmsfs-fuse /dev/dasde /mnt
```

- To mount the CMS file system on the minidisk represented by the file node `/dev/dasde` at `/mnt` and enable EBCDIC to ASCII conversion for text files with extensions as specified in `~/.cmsfs-fuse/filetypes.conf` or `/etc/cmsfs-fuse/filetypes.conf` if the former does not exist:

```
# cmsfs-fuse -t /dev/dasde /mnt
```

- To mount the CMS file system on the minidisk represented by the file node `/dev/dasde` at `/mnt` and allow root to access the mounted file system:

```
# cmsfs-fuse -o allow_root /dev/dasde /mnt
```

- To unmount the CMS file system that was mounted at `/mnt`:

```
# fusermount -u /mnt
```

- To show the record format of a file, PROFILE.EXEC, on a z/VM minidisk that is mounted on `/mnt`:

```
# getfattr -n user.record_format /mnt/PROFILE.EXEC
F
```

- To set record length 80 for an empty fixed record format file, PROFILE.EXEC, on a z/VM minidisk that is mounted on `/mnt`:

```
# setfattr -n user.record_lrecl -v 80 /mnt/PROFILE.EXEC
```

cpuplugd - Control CPUs and memory

Use the **cpuplugd** command and a set of rules in a configuration file to dynamically enable or disable CPUs. For Linux on z/VM, you can also dynamically add or remove memory.

When Linux is running in an LPAR, setting a CPU offline can result in the LPAR status "Exceptions" in the HMC or SE. With one or more CPUs offline, this status does not necessarily indicate a problem.

Rules that are tailored to a particular system environment and the associated workload can increase performance. The rules can include various system load variables.

cpuplugd syntax

```
►►──cpuplugd──┬────┬──┬────┬── -c <config file>──────────────────────────►◄
              └─-f─┘  └─-V─┘
```

Where:

-c or --config *<config file>*
specifies the path to the configuration file with the rule (see "Configuration file structure" on page 567). You can find a sample configuration file at `/etc/sysconfig/cpuplugd`.

-f or --foreground
runs cpuplugd in the foreground and not as a daemon. If this option is omitted, cpuplugd runs as a daemon in the background.

-V or --verbose
displays verbose messages to stdout when cpuplugd is running in the foreground or to syslog when cpuplugd is running as a daemon in the background. This option can be useful for debugging.

-h or --help
displays help information for the command. To view the command man page, enter **man cpuplugd**. To view the man page for the configuration file, enter **man cpuplugd.conf**.

-v or --version
displays version information for cpuplugd.

Examples

- To start cpuplugd in daemon mode with a configuration file `/etc/sysconfig/cpuplugd`:

```
# cpuplugd -c /etc/sysconfig/cpuplugd
```

- To run cpuplugd in the foreground with verbose messages and with a configuration file `/etc/sysconfig/cpuplugd`:

```
# cpuplugd -V -f -c /etc/sysconfig/cpuplugd
```

Configuration file structure

The cpuplugd configuration file can specify rules for controlling the number of active CPUs and for controlling the amount of memory.

The configuration file contains:

- *<variable>*="*<value>*" pairs

  These pairs must be specified within one line. The maximum valid line length is 2048 characters. The values can be decimal numbers or algebraic or Boolean expressions.
- Comments

  Any part of a line that follows a number sign (#) is treated as a comment. There can be full comment lines with the number sign at the beginning of the line or comments can begin in mid-line.
- Empty lines

Attention: These configuration file samples illustrate the syntax of the configuration file. Do not use the sample rules on production systems. Useful rules differ considerably, depending on the workload, resources, and requirements of the system for which they are designed.

Basic configuration file for CPU control

A configuration file for dynamically enabling or disabling CPUs has several required specifications.

The configuration file sample of Figure 111 has been reduced to the specifications that are required for dynamically enabling or disabling CPUs.

```
UPDATE="10"
CPU_MIN="2"
CPU_MAX="10"

HOTPLUG = "idle < 10.0"
HOTUNPLUG = "idle > 100"
```

Figure 111. Simplified configuration file with CPU hotplug rules

In the configuration file:

`UPDATE`
: specifies the time interval, in seconds, at which cpuplugd evaluates the rules and, if a rule is met, enables or disables CPUs. This variable is also required for controlling memory (see "Basic configuration file for memory control" on page 568).

  In the example, the rules are evaluated every 10 seconds.

`CPU_MIN`
: specifies the minimum number of CPUs. Even if the rule for disabling CPUs is met, cpuplugd does not reduce the number of CPUs to less than this number.

  In the example, the number of CPUs cannot become less than 2.

`CPU_MAX`
: specifies the maximum number of CPUs. Even if the rule for enabling CPUs is

met, cpuplugd does not increase the number of CPUs to more than this number. If 0 is specified, the maximum number of CPUs is the number of CPUs available on the system.

In the example, the number of CPUs cannot become more than 10.

HOTPLUG
specifies the rule for dynamically enabling CPUs. The rule resolves to a boolean true or false. Each time this rule is true, cpuplugd enables one CPU, unless the number of CPUs has already reached the maximum specified with CPU_MAX.

Setting HOTPLUG to `0` disables dynamically adding CPUs.

In the example, a CPU is enabled when the idle times of all active CPUs sum up to less than 10.0%. See "Keywords for CPU hotplug rules" on page 570 for information about available keywords.

HOTUNPLUG
specifies the rule for dynamically disabling CPUs. The rule resolves to a boolean true or false. Each time this rule is true, cpuplugd disables one CPU, unless the number of CPUs has already reached the minimum specified with CPU_MIN.

Setting HOTUNPLUG to `0` disables dynamically removing CPUs.

In the example, a CPU is disabled when the idle times of all active CPUs sum up to more than 100%. See "Keywords for CPU hotplug rules" on page 570 for information about available keywords.

If one of these variables is set more than once, only the last occurrence is used. These variables are not case sensitive.

If both the HOTPLUG and HOTUNPLUG rule are met simultaneously, HOTUNPLUG is ignored.

Basic configuration file for memory control

For Linux on z/VM, you can also use cpuplugd to dynamically add or take away memory. There are several required specifications for memory control.

The configuration file sample of Figure 112 was reduced to the specifications that are required for dynamic memory control.

```
UPDATE="10"
CMM_MIN="0"
CMM_MAX="131072"   # 512 MB
CMM_INC="10240"    #  40 MB

MEMPLUG = "swaprate > 250"
MEMUNPLUG = "swaprate < 10"
```

Figure 112. Simplified configuration file with memory hotplug rules

In the configuration file:

UPDATE
specifies the time interval, in seconds, at which cpuplugd evaluates the rules

and, if a rule is met, adds or removes memory. This variable is also required for controlling CPUs (see "Basic configuration file for CPU control" on page 567).

In the example, the rules are evaluated every 10 seconds.

`CMM_MIN`
specifies the minimum amount of memory, in 4 KB pages, that Linux surrenders to the CMM static page pool (see "Cooperative memory management background" on page 415). Even if the MEMPLUG rule for taking memory from the CMM static page pool and adding it to Linux is met, cpuplugd does not decrease this amount.

In the example, the amount of memory that is surrendered to the static page pool can be reduced to 0.

`CMM_MAX`
specifies the maximum amount of memory, in 4 KB pages, that Linux surrenders to the CMM static page pool (see "Cooperative memory management background" on page 415). Even if the MEMUNPLUG rule for removing memory from Linux and adding it to the CMM static page pool is met, cpuplugd does not increase this amount.

In the example, the amount of memory that is surrendered to the static page pool cannot become more than 131072 pages of 4 KB (512 MB).

`CMM_INC`
specifies the amount of memory, in 4 KB pages, that is removed from Linux when the MEMUNPLUG rule is met. Removing memory from Linux increases the amount that is surrendered to the CMM static page pool.

In the example, the amount of memory that is removed from Linux is 10240 pages of 4 KB (40 MB) at a time.

`CMM_DEC`
Optional: specifies the amount of memory, in 4 KB pages, that is added to Linux when the MEMPLUG rule is met. Adding memory to Linux decreases the amount that is surrendered to the CMM static page pool.

If this variable is omitted, the amount of memory that is specified for CMM_INC is used.

In the example, CMM_DEC is omitted and the amount of memory added to Linux is 10240 pages of 4 KB (40 MB) at a time, as specified with CMM_INC.

`MEMPLUG`
specifies the rule for dynamically adding memory to Linux. The rule resolves to a Boolean true or false. Each time this rule is true, cpuplugd adds the number of pages that are specified by CMM_DEC, unless the CMM static page pool already reached the minimum that is specified with CMM_MIN.

Setting MEMPLUG to `0` disables dynamically adding memory to Linux.

In the example, memory is added to Linux if there are more than 250 swap operations per second. See "Keywords for memory hotplug rules" on page 571 for information about available keywords.

`MEMUNPLUG`
specifies the rule for dynamically removing memory from Linux. The rule resolves to a Boolean true or false. Each time this rule is true, cpuplugd removes the number of pages specified by CMM_INC, unless the CMM static page pool already reached the maximum that is specified with CMM_MAX.

Setting MEMUNPLUG to `0` disables dynamically removing memory from Linux.

In the example, memory is removed from Linux when there are less than 10 swap operations per second. See "Keywords for memory hotplug rules" on page 571 for information about available keywords.

If any of these variables are set more than once, only the last occurrence is used. These variables are not case-sensitive.

If both the MEMPLUG and MEMUNPLUG rule are met simultaneously, MEMUNPLUG is ignored.

CMM_DEC and CMM_INC can be set to a decimal number or to a mathematical expression that uses the same algebraic operators and variables as the MEMPLUG and MEMUNPLUG hotplug rules (see "Keywords for memory hotplug rules" on page 571 and "Writing more complex rules" on page 572).

Predefined keywords

There is a set of predefined keywords that you can use for CPU hotplug rules and a set of keywords that you can use for memory hotplug rules.

All predefined keywords are case sensitive.

Keywords for CPU hotplug rules:

There are predefined keywords for use in the CPU hotplug rules, HOTPLUG and HOTUNPLUG.

The following keywords are available:

`loadavg`
is the current load average.

`onumcpus`
is the current number of online CPUs.

`runnable_proc`
is the current number of runnable processes.

`user`
is the current CPU user percentage.

`nice`
is the current CPU nice percentage.

`system`
is the current CPU system percentage.

`idle`
is the current CPU idle percentage.

`iowait`
is the current CPU iowait percentage.

`irq`
is the current CPU irq percentage.

`softirq`
is the current CPU softirq percentage.

`steal`
is the current CPU steal percentage.

`guest`
is the current CPU guest percentage.

`guest_nice`
is the current CPU guest_nice percentage.

`cpustat.`*<name>*
is data from `/proc/stat` and `/proc/loadavg`. In the keyword, *<name>* can be any of the previously listed keywords, for example, `cpustat.idle`. See the proc man page for more details about the data that is represented by these keywords.

With this notation, the keywords resolve to raw timer ticks since system start, not to current percentages. For example, `idle` resolves to the current idle percentage and `cpustat.idle` resolves to the total timer ticks spent idle. See "Using historical data" on page 572 about how to obtain average and percentage values.

`loadavg`, `onumcpus`, and `runnable_proc` are not percentages and resolve to the same values as `cpustat.loadavg`, `cpustat.onumcpus`, and `cpustat.runnable_proc`.

`cpustat.total_ticks`
is the total number of timer ticks since system start.

`time`
is the UNIX epoch time in the format "seconds.microseconds".

Percentage values are accumulated for all online CPUs. Hence, the values for the percentages range from 0 to 100 × (number of online CPUs). To get the average percentage per CPU device, divide the accumulated value by the number of CPUs. For example, `idle / onumcpus` yields the average idle percentage per CPU.

Keywords for memory hotplug rules:

There are predefined keywords for use in the memory hotplug rules, MEMPLUG and MEMUNPLUG.

The following keywords are available:

`apcr`
is the number of page cache operations, pgpin + pgpout, from `/proc/vmstat` in 512-byte blocks per second.

`freemem`
is the amount of free memory in MB.

`swaprate`
is the number of swap operations, pswpin + pswpout, from `/proc/vmstat` in 4 KB pages per second.

`meminfo.`*<name>*
is the value for the symbol *<name>* as shown in the output of **`cat /proc/meminfo`**. The values are plain numbers but refer to the same units as those used in `/proc/meminfo`.

`vmstat.`*<name>*
is the value for the symbol *<name>* as shown in the output of **`cat /proc/vmstat`**.

Using historical data:

Historical data is available for the keyword `time` and the sets of keywords `cpustat.`*<name>*, `meminfo.`*<name>*, and `vmstat.`*<name>*.

See "Keywords for CPU hotplug rules" on page 570 and "Keywords for memory hotplug rules" on page 571 for details about these keywords.

Use the suffixes [*<n>*] to retrieve the data of *<n>* intervals in the past, where *<n>* can be in the range 0 - 100.

Examples

cpustat.idle
yields the current value for the counted idle ticks.

cpustat.idle[1]
yields the idle ticks as counted one interval ago.

cpustat.idle[5]
yields the idle ticks as counted five intervals ago.

cpustat.idle - cpustat.idle[5]
yields the idle ticks during the past five intervals.

time - time[1]
yields the length of an update interval in seconds.

cpustat.total_ticks - cpustat.total_ticks[5]
yields the total number of ticks during the past five intervals.

(cpustat.idle - cpustat.idle[5]) / (cpustat.total_ticks - cpustat.total_ticks[5])
yields the average ratio of idle ticks to total ticks during the past five intervals.

Multiplying this ratio with 100 yields the percentage of idle ticks during the last five intervals.

Multiplying this ratio with 100 * `onumcpus` yields the accumulated percentage of idle ticks for all processors during the last five intervals.

Writing more complex rules

In addition to numbers and keywords, you can use mathematical and Boolean operators, and you can use user-defined variables to specify rules.

- The keywords of "Predefined keywords" on page 570
- Decimal numbers
- The mathematical operators

+	addition
-	subtraction
*****	multiplication
/	division
<	less than
>	greater than

- Parentheses (and) to group mathematical expressions
- The Boolean operators

&	and
|	or
!	not

- User-defined variables

You can specify complex calculations as user-defined variables, which can then be used in expressions. User-defined variables are case-sensitive and must not match a pre-defined variable or keyword. In the configuration file, definitions for user-defined variables must precede their use in expressions.

Variable names consist of alphanumeric characters and the underscore (_) character. An individual variable name must not exceed 128 characters. All user-defined variable names and values, in total, must not exceed 4096 characters.

Examples

- `HOTPLUG = "loadavg > onumcpus + 0.75"`
- `HOTPLUG = "(loadavg > onumcpus + 0.75) & (idle < 10.0)"`
-
  ```
  my_idle_rate = "(cpustat.idle - cpustat.idle[5]) / (cpustat.total_ticks - cpustat.total_ticks[5])"
  my_idle_percent_total = "my_idle_rate * 100 * onumcpus"
  ...
  HOTPLUG = "(loadavg > onumcpus + 0.75) & (my_idle_percent_total < 10.0)"
  ```

Sample configuration file

A typical configuration file includes multiple user-defined variables and values from procfs, for example, to calculate the page scan rate or the cache size.

```
# Required static variables

CPU_MIN="1"
CPU_MAX="0"
UPDATE="1"
CMM_MIN="0"
CMM_MAX="131072" # 512 MB

# User-defined variables

pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] + vmstat.pgscan_direct_movable[1]"
# page scan rate in pages / timer tick
pgscanrate="(pgscan_d - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
# cache usage in kilobytes
avail_cache="meminfo.Cached - meminfo.Shmem"

user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"
user_2="(cpustat.user[2] - cpustat.user[3])"
nice_2="(cpustat.nice[2] - cpustat.nice[3])"
system_2="(cpustat.system[2] - cpustat.system[3])"
CP_Active0="(user_0 + nice_0 + system_0) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_Active2="(user_2 + nice_2 + system_2) / (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
idle_2="(cpustat.idle[2] - cpustat.idle[3])"
iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"
CP_idle0="(idle_0 + iowait_0) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle2="(idle_2 + iowait_2) / (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_idleAVG="(CP_idle0 + CP_idle2) / 2"

# More required variables

# cmm_inc: 10% of free memory, in 4K pages
CMM_INC="meminfo.MemFree / 40"
# cmm_dec: 10% of total memory, in 4K pages
CMM_DEC="meminfo.MemTotal / 40"

# Hotplug rules
HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"
MEMPLUG="pgscanrate > 20"
MEMUNPLUG="(meminfo.MemFree + avail_cache) > (meminfo.MemTotal / 10)"
```

Figure 113. Sample configuration file for CPU and memory hotplug

After you install cpuplugd with the s390-tools RPM, a commented sample configuration file is available at `/etc/sysconfig/cpuplugd`.

Attention: These configuration file samples illustrate the syntax of the configuration file. Do not use the sample rules on production systems. Useful rules differ considerably, depending on the workload, resources, and requirements of the system for which they are designed.

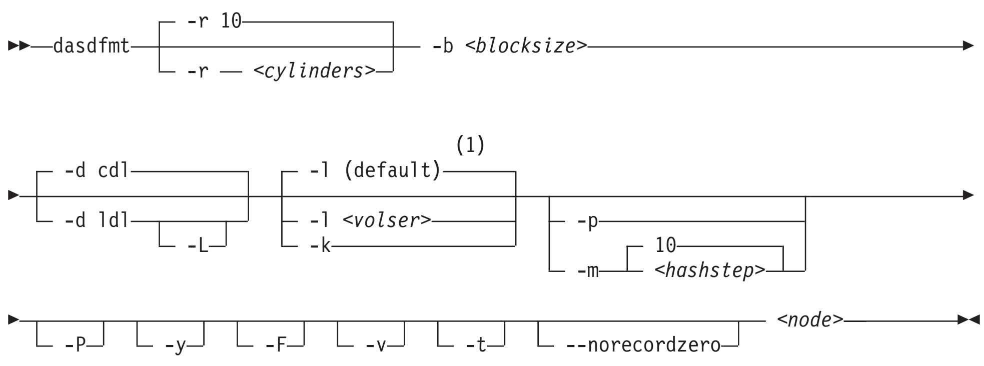
dasdfmt - Format a DASD

Use the **dasdfmt** command to low-level format ECKD-type direct access storage devices (DASD).

dasdfmt uses an ioctl call to the DASD driver to format tracks. A block size (hard sector size) can be specified. The formatting process can take quite a long time (hours for large DASD). Use the **-p** option to monitor the progress.

CAUTION:
As on any platform, formatting irreversibly destroys data on the target disk. Be sure not to format a disk with vital data unintentionally.

dasdfmt syntax



```
                 ┌─-r 10──────────────┐
►►──dasdfmt──────┼────────────────────┼── -b <blocksize>──────────────────────►
                 └─-r ── <cylinders>──┘

                                 (1)
 ┌─-d cdl───────────┐   ┌─-l (default)───┐
►┼──────────────────┼───┼────────────────┼───┬────────────────────────┬──────►
 └─-d ldl──┬─────┬──┘   ├─-l <volser>────┤   ├─-p─────────────────────┤
           └─-L──┘      └─-k─────────────┘   │      ┌─10─────────┐    │
                                             └─-m───┴─<hashstep>─┴────┘

►──┬────┬──┬────┬──┬────┬──┬────┬──┬────┬──┬────────────────┬── <node>──────►◄
   └─-P─┘  └─-y─┘  └─-F─┘  └─-v─┘  └─-t─┘  └─--norecordzero─┘
```

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:

-r *<cylinders>* **or --requestsize=***<cylinders>*
specifies the number of cylinders to be processed in one formatting step. The value must be an integer in the range 1 - 255. The default is 10 cylinders. Use this parameter to use any available PAV devices. Ideally, the number of cylinders matches the number of associated devices, counting the base device and all alias devices.

-b *<block_size>* **or --blocksize=***<block_size>*
specifies one of the following block sizes in bytes: 512, 1024, 2048, or 4096.

If you do not specify a value for the block size, you are prompted. You can then press Enter to accept 4096 or specify a different value.

Tip: Set *<block_size>* as large as possible (ideally 4096); the net capacity of an ECKD DASD decreases for smaller block sizes. For example, a DASD formatted with a block size of 512 byte has only half of the net capacity of the same DASD formatted with a block size of 4096 byte.

<node>
specifies the device node of the device to be formatted, for example, `/dev/dasdzzz`. See "DASD naming scheme" on page 151 for more details about device nodes.

-d *<disklayout>* **or --disk_layout=***<disklayout>*
formats the device with the compatible disk layout (cdl) or the Linux disk layout (ldl). If the parameter is not specified, the default (cdl) is used.

-L or --no_label
valid for `-d ldl` only, where it suppresses the default LNX1 label.

-l *<volser>* **or --label=***<volser>*
specifies the volume serial number (see VOLSER) to be written to the disk. If the VOLSER contains special characters, it must be enclosed in single quotation marks. In addition, any '$' character in the VOLSER must be preceded by a backslash ('\').

-k or --keep_volser
keeps the volume serial number when writing the volume label (see VOLSER). Keeping the volume serial number is useful, for example, if the volume serial number was written with a z/VM tool and should not be overwritten.

-p or --progressbar
displays a progress bar. Do not use this option if you are using a line-mode terminal console driver. For example, if you are using a 3215 terminal device driver or a line-mode hardware console device driver.

-P or --percentage
displays one line for each formatted cylinder. The line shows the number of the cylinder and percentage of formatting process. Intended for use by higher level interfaces.

-m *<hashstep>* **or --hashmarks=***<hashstep>*
displays a number sign (#) after every *<hashstep>* cylinders are formatted. *<hashstep>* must be in the range 1 - 1000. The default is 10.

The **-m** option is useful where the console device driver is not suitable for the progress bar (**-p** option).

-y starts formatting immediately without prompting for confirmation.

-F or --force
formats the device without checking whether it is mounted.

-v displays extra information messages (verbose).

-t or --test
runs the command in test mode. Analyzes parameters and prints what would happen, but does not modify the disk.

-- norecordzero
prevents a format write of record zero. This option is intended for experts: Subsystems in DASD drivers are by default granted permission to modify or add a standard record zero to each track when needed. Before you revoke the permission with this option, you must ensure that the device contains standard record zeros on all tracks.

-V or --version
displays the version number of **dasdfmt** and exits.

`-h or --help`
displays an overview of the syntax. Any other parameters are ignored. To view the man page, enter **`man dasdfmt`**.

Examples

- To format a 100 cylinder z/VM minidisk with the standard Linux disk layout and a 4 KB blocksize with device node `/dev/dasdc`:

```
# dasdfmt -b 4096 -d ldl -p /dev/dasdc
Drive Geometry: 100 Cylinders * 15 Heads =  1500 Tracks

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 (accepting 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
#
```

- To format with the **`-P`** option:

```
# dasdfmt -P /dev/dasde

cyl     1 of   500 |     0%
cyl     2 of   500 |     0%
cyl     3 of   500 |     0%
cyl     4 of   500 |     0%
cyl     5 of   500 |     1%

...
cyl   496 of   500 |    99%
cyl   497 of   500 |    99%
cyl   498 of   500 |    99%
cyl   499 of   500 |    99%
cyl   500 of   500 |   100%
```

- To make best use of PAV when formatting a DASD that has one base device and four alias devices, specify five cylinders:

```
# dasdfmt /dev/dasdd -y -b 4096 -d cdl -r 5
Finished formatting the device.
Rereading the partition table... ok
```

dasdstat - Display DASD performance statistics

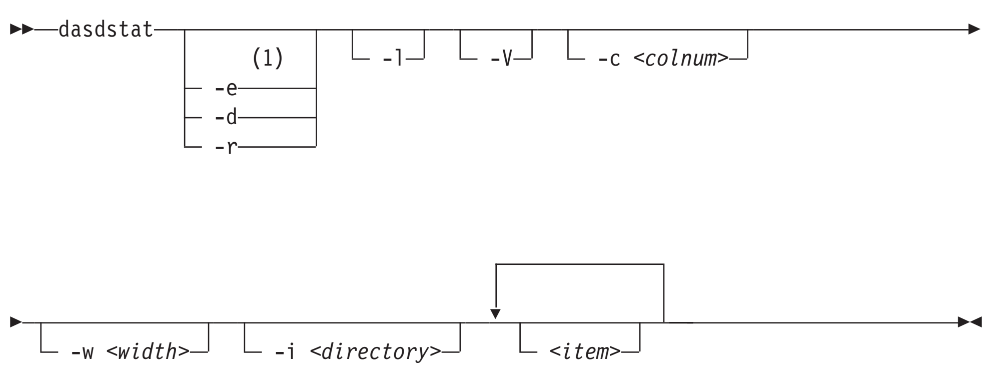
Use the **dasdstat** command to display DASD performance statistics, including statistics about Parallel Access Volume (PAV) and High Performance Ficon.

This command includes and extends the performance statistics that is also available through the **tunedasd** command.

Before you begin:

- Your DASD device driver must be compiled with profiling support (see "Building a kernel with the DASD device driver" on page 155).
- You need a kernel that supports debugfs and debugfs must be mounted.

dasdstat syntax



Notes:

1 Omit the **-e**, **-d**, and **-r** options to read statistics.

Where:

-e or --enable
: starts statistics data collection.

-d or --disable
: stops statistics data collection.

-r or --reset
: sets the statistics counters to zero.

-l or --long
: displays more detailed statistics information, for example, differentiates between read and write requests.

-V or --verbose
: displays more verbose command information.

-c *<colnum>* **or --columns** *<colnum>*
: formats the command output in a table with the specified number of columns. The default is 16. Each row gets wrapped after the specified number of lines.

-w *<width>* **or --column-width** *<width>*
: sets the minimum width, in characters, of a column in the output table.

-i *<directory>* **or --directory** *<directory>*
specifies the directory that contains the statistics. The default is *<mountpoint>*/dasd, where *<mountpoint>* is the mount point of debugfs. You need to specify this parameter if the **dasdstat** command cannot determine this mount point or if the statistics are copied to another location.

<item>
limits the command to the specified items. For *<item>* you can specify:

- global for summary statistics for all available DASDs.
- The block device name by which a DASD is known to the DASD device driver.
- The bus ID by which a DASD is known as a CCW device. DASDs that are set up for PAV or HyperPAV have a CCW base device and, at any one time, can have one or more CCW alias devices for the same block device. Alias devices are not permanently associated with the same block device. Statistics that are based on bus ID, therefore, show additional detail for PAV and HyperPAV setups.

If you do not specify any individual item, the command applies to all DASD block devices, CCW devices, and to the summary.

-v or --version
displays the version number of **dasdstat**, then exits.

-h or --help
displays help information for the command.

Examples

- This command starts data collection for dasda, 0.0.b301, and for a summary of all available DASDs.

```
# dasdstat -e dasda 0.0.b301 0.0.b302 global
```

- This command resets the statistics counters for dasda.

```
# dasdstat -r dasda
```

- This command reads the summary statistics:

```
statistics data for statistic: global
start time of data collection: Wed Aug 17 09:52:47 CEST 2011

3508 dasd I/O requests
with 67616 sectors(512B each)
0 requests used a PAV alias device
3458 requests used HPF
 __<4 ___8 __16 __32 __64 _128 _256 _512 __1k __2k __4k __8k _16k _32k _64k 128k
 _256 _512 __1M __2M __4M __8M _16M _32M _64M 128M 256M 512M __1G __2G __4G _>4G
Histogram of sizes (512B secs)
    0    0 2456  603  304  107   18    9    3    8    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
Histogram of I/O times (microseconds)
    0    0    0    0    0    0  100 1738  813  725   30   39   47   15    1    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
Histogram of I/O time till ssch
    0    0  901  558  765   25   28  288  748  161   17   16    1    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
Histogram of I/O time between ssch and irq
    0    0    0    0    0    0  316 2798  283   13   19   22   41   15    1    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
Histogram of I/O time between irq and end
    0 3023  460    8    4    9    4    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
# of req in chanq at enqueuing (0..31)
 ___0 ___1 ___2 ___3 ___4 ___5 ___6 ___7 ___8 ___9 __10 __11 __12 __13 __14 __15
 __16 __17 __18 __19 __20 __21 __22 __23 __24 __25 __26 __27 __28 __29 __30 __31
    0 2295  319  247  647    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

For details about the data items, see "Interpreting the data rows" on page 174.

dasdview - Display DASD structure

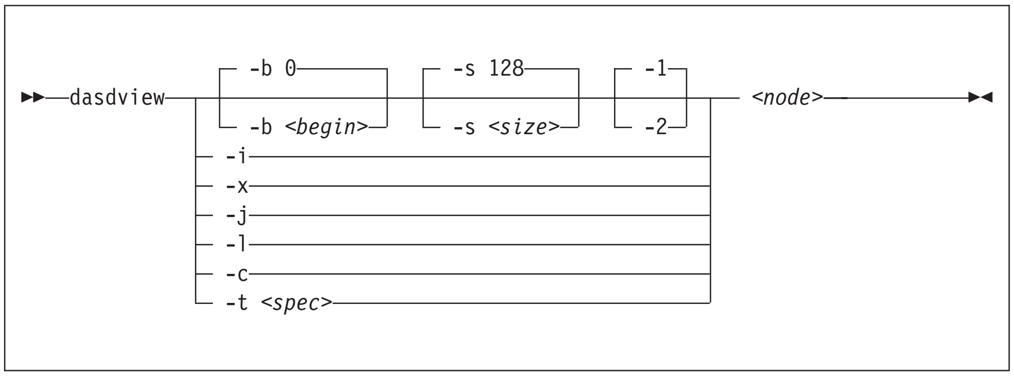
Use the **dasdview** command to display DASD information.

dasdview displays:
- The volume label.
- VTOC details (general information, and the DSCBs of format 1, format 3, format 4, format 5, format 7, format 8, and format 9).
- The content of the DASD, by specifying:
  - Starting point
  - Size

  You can display these values in hexadecimal, EBCDIC, and ASCII format.
- Device characteristics, such as:
  - Whether the data on the DASD is encrypted.
  - Whether the disk is a solid-state device.

If you specify a start point and size, you can also display the contents of a disk dump. For more information about partitioning, see "The IBM label partitioning scheme" on page 146.

dasdview syntax



Where:

-b *<begin>* **or --begin=***<begin>*
: displays disk content on the console, starting from *<begin>*. The contents of the disk are displayed as hexadecimal numbers, ASCII text, and EBCDIC text. If *<size>* is not specified, **dasdview** takes the default size (128 bytes). You can specify the variable *<begin>* as:

  ```
  <begin>[k|m|b|t|c]
  ```

  If the disk is in raw-track access mode, you can specify only track (t) or cylinder (c) entities.

  The default for *<begin>* is 0.

  dasdview displays a disk dump on the console by using the DASD driver. The DASD driver might suppress parts of the disk, or add information that is not relevant. Such discrepancies might occur, for example, when **dasdview** displays the first two tracks of a disk that was formatted as **cdl**. In this situation, the DASD driver pads shorter blocks with zeros to maintain a constant blocksize. All Linux applications (including **dasdview**) process according to this rule.

  Here are some examples of how this option can be used:

```
-b 32   (start printing at Byte 32)
-b 32k  (start printing at kByte 32)
-b 32m  (start printing at MByte 32)
-b 32b  (start printing at block 32)
-b 32t  (start printing at track 32)
-b 32c  (start printing at cylinder 32)
```

-s *<size>* **or --size=***<size>*
displays a disk dump on the console, starting at *<begin>*, and continuing for **size**=*<size>*. The contents of the dump are displayed as hexadecimal numbers, ASCII text, and EBCDIC text. If a start value, *<begin>*, is not specified, **dasdview** takes the default. You can specify the variable *<size>* as:

```
size[k|m|b|t|c]
```

If the disk is in raw-track access mode, you can specify only track (t) or cylinder (c) entities.

The default for *<size>* is 128 bytes. Here are some examples of how this option can be used:

```
-s 16   (use a 16 Byte size)
-s 16k  (use a 16 kByte size)
-s 16m  (use a 16 MByte size)
-s 16b  (use a 16 block size)
-s 16t  (use a 16 track size)
-s 16c  (use a 16 cylinder size)
```

-1 displays the disk dump with format 1 (as 16 Bytes per line in hexadecimal, ASCII and EBCDIC). A line number is not displayed. You can use option **-1** only together with **-b** or **-s**.

Option **-1** is the default.

-2 displays the disk dump with format 2 (as 8 Bytes per line in hexadecimal, ASCII and EBCDIC). A decimal and hexadecimal byte count are also displayed. You can use option **-2** only together with **-b** or **-s**.

-i or --info
displays basic information such as device node, device bus-ID, device type, or geometry data.

-x or --extended
displays the information that is obtained by using **-i** option, but also open count, subchannel identifier, and so on.

-j or --volser
displays volume serial number (volume identifier).

-l or --label
displays the volume label.

The `-l` option displays all known label fields. The fields that are shown depend on the label, which is identified by the 'volume label identifier'. The most important differences are:

volume label key
is only valid for 'VOL1' labels (used for ECKD CDL format)

VTOC pointer
is only valid for 'VOL1' labels

ldl_version
is only valid for 'LNX1' labels (used for ECKD LDL format)

formatted_blocks
is only valid for 'LNX1' labels and when the (EBCDIC) ldl_version is 2 or higher

-c or --characteristics
displays model-dependent device characteristics, for example disk encryption status or whether the disk is a solid-state device.

-t *<spec>* **or --vtoc=***<spec>*
displays the VTOC's table-of-contents, or a single VTOC entry, on the console. The variable *<spec>* can take these values:

info	displays overview information about the VTOC, such as a list of the data set names and their sizes.
f1	displays the contents of all *format 1* data set control blocks (DSCBs).
f3	displays the contents of all (z/OS-specific) *format 3* DSCBs.
f4	displays the contents of all *format 4* DSCBs.
f5	displays the contents of all *format 5* DSCBs.
f7	displays the contents of all *format 7* DSCBs.
f8	displays the contents of all *format 8* DSCBs.
f9	displays the contents of all *format 9* DSCBs.
all	displays the contents of *all* DSCBs.

<node>
specifies the device node of the device for which you want to display information, for example, `/dev/dasdzzz`. See "DASD naming scheme" on page 151 for more details about device nodes).

-h or --help
displays short usage text on console. To view the man page, enter **man dasdview**.

-v or --version
displays version number on console, and exit.

Examples

- To display basic information about a DASD:

```
# dasdview -i /dev/dasdzzz
```

This example displays:

```
--- general DASD information --------------------------------------------------
device node            : /dev/dasdzzz
busid                  : 0.0.0193
type                   : ECKD
device type            : hex 3390       dec 13200

--- DASD geometry -------------------------------------------------------------
number of cylinders    : hex 64         dec 100
tracks per cylinder    : hex f          dec 15
blocks per track       : hex c          dec 12
blocksize              : hex 1000       dec 4096
#
```

- To display device characteristics:

```
# dasdview -c /dev/dasda
```

This example displays:

```
encrypted disk         : no
solid state device     : no
```

- To include extended information:

```
# dasdview -x /dev/dasdzzz
```

This example displays:

```
--- general DASD information --------------------------------------------------
device node            : /dev/dasdzzz
busid                  : 0.0.0193
type                   : ECKD
device type            : hex 3390       dec 13200

--- DASD geometry -------------------------------------------------------------
number of cylinders    : hex 64         dec 100
tracks per cylinder    : hex f          dec 15
blocks per track       : hex c          dec 12
blocksize              : hex 1000       dec 4096

--- extended DASD information -------------------------------------------------
real device number     : hex 452bc08    dec 72530952
subchannel identifier  : hex e          dec 14
CU type  (SenseID)     : hex 3990       dec 14736
CU model (SenseID)     : hex e9         dec 233
device type  (SenseID) : hex 3390       dec 13200
device model (SenseID) : hex a          dec 10
open count             : hex 1          dec 1
req_queue_len          : hex 0          dec 0
chanq_len              : hex 0          dec 0
status                 : hex 5          dec 5
label_block            : hex 2          dec 2
FBA_layout             : hex 0          dec 0
characteristics_size   : hex 40         dec 64
confdata_size          : hex 100        dec 256

characteristics        : 3990e933 900a5f80  dff72024 0064000f
                         e000e5a2 05940222  13090674 00000000
                         00000000 00000000  24241502 dfee0001
                         0677080f 007f4a00  1b350000 00000000

configuration_data     : dc010100 4040f2f1  f0f54040 40c9c2d4
                         f1f3f0f0 f0f0f0f0  f0c6c3f1 f1f30509
                         dc000000 4040f2f1  f0f54040 40c9c2d4
                         f1f3f0f0 f0f0f0f0  f0c6c3f1 f1f30500
                         d4020000 4040f2f1  f0f5c5f2 f0c9c2d4
                         f1f3f0f0 f0f0f0f0  f0c6c3f1 f1f3050a
                         f0000001 4040f2f1  f0f54040 40c9c2d4
                         f1f3f0f0 f0f0f0f0  f0c6c3f1 f1f30500
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         800000a1 00001e00  51400009 0909a188
                         0140c009 7cb7efb7  00000000 00000800
#
```

- To display volume label information for a cdl formatted disk:

```
# dasdview -l /dev/dasdzzz
```

This example displays:

```
--- volume label --------------------------------------------------------------
volume label key        : ascii  'åÖÓñ'
                        : ebcdic 'VOL1'
                        : hex    e5d6d3f1

volume label identifier : ascii  'åÖÓñ'
                        : ebcdic 'VOL1'
                        : hex    e5d6d3f1

volume identifier       : ascii  'ðçðñùó'
                        : ebcdic '0X0193'
                        : hex    f0e7f0f1f9f3

security byte           : hex    40


VTOC pointer            : hex    0000000101
                                 (cyl 0, trk 1, blk 1)

reserved                : ascii  '@@@@@'
                        : ebcdic '     '
                        : hex    4040404040

CI size for FBA         : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

blocks per CI (FBA)     : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

labels per CI (FBA)     : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

reserved                : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

owner code for VTOC     : ascii  '@@@@@@@@@@@@@@'
                          ebcdic '              '
                          hex    40404040 40404040  40404040 4040

reserved                : ascii  '@@@@@@@@@@@@@@@@@@@@@@@@@@@@'
                          ebcdic '                            '
                          hex    40404040 40404040  40404040 40404040
                                 40404040 40404040  40404040

ldl_version             : ascii  '@'
                        : ebcdic ' '
                        : hex    40

formatted_blocks        : dec 16565899579919558117
                        : hex e5e5e5e5e5e5e5e5#
```

- To display volume label information for an ldl formatted disk:

```
--- volume label ---------------------------------------------------------------
volume label key        : ascii  '    '
                        : ebcdic '    '
                        : hex    00000000

volume label identifier : ascii  'ÓÕçñ'
                        : ebcdic 'LNX1'
                        : hex    d3d5e7f1

volume identifier       : ascii  'ðçðñùó'
                        : ebcdic '0X0193'
                        : hex    f0e7f0f1f9f3

security byte           : hex    40


VTOC pointer            : hex    4040404040
reserved                : ascii  '@@@@@'
                        : ebcdic '     '
                        : hex    4040404040

CI size for FBA         : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

blocks per CI (FBA)     : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

labels per CI (FBA)     : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

reserved                : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

owner code for VTOC     : ascii  '@@@@@@@@@@@@@@'
                          ebcdic '              '
                          hex    40404040 40404040  40404040 4040

reserved                : ascii  '@@@@@@@@@@@@@@@@@@@@@@@@@@@@'
                          ebcdic '                            '
                          hex    40404040 40404040  40404040 40404040
                                 40404040 40404040  40404040

ldl_version             : ascii  'ò'
                        : ebcdic '2'
                        : hex    f2

formatted_blocks        : dec 18000
                        : hex 0000000000004650
```

- To display partition information:

```
# dasdview -t info /dev/dasdzzz
```

This example displays:

```
--- VTOC info ---------------------------------------------------------------
The VTOC contains:
  3 format 1 label(s)
  1 format 4 label(s)
  1 format 5 label(s)
  0 format 7 label(s)
  0 format 8 label(s)
  0 format 9 label(s)
Other S/390 and zSeries operating systems would see the following data sets:
 +----------------------------------------------+--------------+--------------+
 | data set                                     | start        | end          |
 +----------------------------------------------+--------------+--------------+
 | LINUX.V0X0193.PART0001.NATIVE                |          trk |          trk |
 | data set serial number : '0X0193'            |            2 |          500 |
 | system code            : 'IBM LINUX    '     |      cyl/trk |      cyl/trk |
 | creation date          :  year 2009, day  55 |        0/  2 |       33/  5 |
 +----------------------------------------------+--------------+--------------+
 | LINUX.V0X0193.PART0002.NATIVE                |          trk |          trk |
 | data set serial number : '0X0193'            |          501 |          900 |
 | system code            : 'IBM LINUX    '     |      cyl/trk |      cyl/trk |
 | creation date          :  year 2009, day  55 |       33/  6 |       60/  0 |
 +----------------------------------------------+--------------+--------------+
 | LINUX.V0X0193.PART0003.NATIVE                |          trk |          trk |
 | data set serial number : '0X0193'            |          901 |         1499 |
 | system code            : 'IBM LINUX    '     |      cyl/trk |      cyl/trk |
 | creation date          :  year 2009, day  55 |       60/  1 |       99/ 14 |
 +----------------------------------------------+--------------+--------------+
#
```

- To display VTOC format 4 label information:

```
# dasdview -t f4 /dev/dasdzzz
```

This example displays:

```
--- VTOC format 4 label ------------------------------------------------------
DS4KEYCD    : 04040404040404040404040404040404040404040404040404040404040404040404...
DS4IDFMT    : dec 244, hex f4
DS4HPCHR    : 0000000105 (cyl 0, trk 1, blk 5)
DS4DSREC    : dec 7, hex 0007
DS4HCCHH    : 00000000 (cyl 0, trk 0)
DS4NOATK    : dec 0, hex 0000
DS4VTOCI    : dec 0, hex 00
DS4NOEXT    : dec 1, hex 01
DS4SMSFG    : dec 0, hex 00
DS4DEVAC    : dec 0, hex 00
DS4DSCYL    : dec 100, hex 0064
DS4DSTRK    : dec 15, hex 000f
DS4DEVTK    : dec 58786, hex e5a2
DS4DEVI     : dec 0, hex 00
DS4DEVL     : dec 0, hex 00
DS4DEVK     : dec 0, hex 00
DS4DEVFG    : dec 48, hex 30
DS4DEVTL    : dec 0, hex 0000
DS4DEVDT    : dec 12, hex 0c
DS4DEVDB    : dec 0, hex 00
DS4AMTIM    : hex 0000000000000000
DS4AMCAT    : hex 000000
DS4R2TIM    : hex 0000000000000000
res1        : hex 0000000000
DS4F6PTR    : hex 0000000000
DS4VTOCE    : hex 01000000000100000001
              typeind    : dec 1, hex 01
              seqno      : dec 0, hex 00
              llimit     : hex 00000001 (cyl 0, trk 1)
              ulimit     : hex 00000001 (cyl 0, trk 1)
res2        : hex 00000000000000000000
DS4EFLVL    : dec 0, hex 00
DS4EFPTR    : hex 0000000000 (cyl 0, trk 0, blk 0)
res3        : hex 00
DS4DCYL     : dec 100, hex 00000064
res4        : hex 0000
DS4DEVF2    : dec 64, hex 40
res5        : hex 00
#
```

- To print the contents of a disk to the console starting at block 2 (volume label):

```
# dasdview -b 2b -s 128 /dev/dasdzzz
```

This example displays:

```
+----------------------------------------+------------------+------------------+
| HEXADECIMAL                            | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1............16 | 1............16 |
+----------------------------------------+------------------+------------------+
|  E5D6D3F1 E5D6D3F1  F0E7F0F1 F9F34000  | VOL1VOL10X0193?. | ??????????????@. |
|  00000101 40404040  40404040 40404040  | ................ | ................ |
|  40404040 40404040  40404040 40404040  | ???????????????? | @@@@@@@@@@@@@@@@ |
|  40404040 40404040  40404040 40404040  | ???????????????? | @@@@@@@@@@@@@@@@ |
|  40404040 40404040  40404040 40404040  | ???????????????? | @@@@@@@@@@@@@@@@ |
|  40404040 88001000  10000000 00808000  | ????h........... | @@@@?........... |
|  00000000 00000000  00010000 00000200  | ................ | ................ |
|  21000500 00000000  00000000 00000000  | ?............... | !............... |
+----------------------------------------+------------------+------------------+
#
```

- To display the contents of a disk on the console starting at block 14 (first FMT1 DSCB) with format 2:

```
# dasdview -b 14b -s 128 -2 /dev/dasdzzz
```

This example displays:

```
+---------------+---------------+----------------------+----------+----------+
|     BYTE      |     BYTE      |     HEXADECIMAL      |  EBCDIC  |  ASCII   |
|    DECIMAL    |  HEXADECIMAL  |  1 2 3 4   5 6 7 8   | 12345678 | 12345678 |
+---------------+---------------+----------------------+----------+----------+
|         57344 |          E000 |  D3C9D5E4  E74BE5F0  | LINUX.V0 | ?????K?? |
|         57352 |          E008 |  E7F0F1F9  F34BD7C1  | X0193.PA | ?????K?? |
|         57360 |          E010 |  D9E3F0F0  F0F14BD5  | RT0001.N | ??????K? |
|         57368 |          E018 |  C1E3C9E5  C5404040  | ATIVE??? | ?????@@@ |
|         57376 |          E020 |  40404040  40404040  | ???????? | @@@@@@@@ |
|         57384 |          E028 |  40404040  F1F0E7F0  | ????10X0 | @@@@???? |
|         57392 |          E030 |  F1F9F300  0165013D  | 193.???? | ???.?e?= |
|         57400 |          E038 |  63016D01  0000C9C2  | ??_?..IB | c?m?..?? |
|         57408 |          E040 |  D440D3C9  D5E4E740  | M?LINUX? | ?@?????@ |
|         57416 |          E048 |  40404065  013D0000  | ??????.. | @@@e?=.. |
|         57424 |          E050 |  00000000  88001000  | ....h.?. | ....?.?. |
|         57432 |          E058 |  10000000  00808000  | ?....??. | ?....??. |
|         57440 |          E060 |  00000000  00000000  | ........ | ........ |
|         57448 |          E068 |  00010000  00000200  | .?....?. | .?....?. |
|         57456 |          E070 |  21000500  00000000  | ?.?..... | !.?..... |
|         57464 |          E078 |  00000000  00000000  | ........ | ........ |
+---------------+---------------+----------------------+----------+----------+
#
```

- To see what is at block 1234 (in this example there is nothing there):

```
# dasdview -b 1234b -s 128 /dev/dasdzzz
```

This example displays:

```
+----------------------------------------+------------------+------------------+
| HEXADECIMAL                            | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+----------------------------------------+------------------+------------------+
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
|  00000000 00000000  00000000 00000000  | ................ | ................ |
+----------------------------------------+------------------+------------------+
#
```

- To try byte 0 instead:

```
# dasdview -b 0 -s 64 /dev/dasdzzz
```

This example displays:

```
+----------------------------------------+------------------+------------------+
| HEXADECIMAL                            | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+----------------------------------------+------------------+------------------+
|  C9D7D3F1 000A0000  0000000F 03000000  | IPL1............ | ????............ |
|  00000001 00000000  00000000 40404040  | ................ | ................ |
|  40404040 40404040  40404040 40404040  | ???????????????? | @@@@@@@@@@@@@@@@ |
|  40404040 40404040  40404040 40404040  | ???????????????? | @@@@@@@@@@@@@@@@ |
+----------------------------------------+------------------+------------------+
#
```

- To display the contents of a disk on the console starting at cylinder 2 and printing one track of data:

```
# dasdview -b 2c -s 1t /dev/dasdk
```

This example displays:

```
+----------------------------------------+------------------+------------------+
| HEXADECIMAL                            | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1............16 | 1............16 |
+----------------------------------------+------------------+------------------+
|  52B7DBEE D6B9530B  0179F420 CB6EA95E  | ????O?????4??>z; | R?????S??y???n?^ |
|  EF49C03C 513542E7  D8F17D9D 06DC44F7  | ??{????XQ1'????7 | ?I?<Q5B???}???D? |

...
|  92963D5B 0200B0FA  53745C12 C3B45125  | ko?$?........... | ??=[?........... |
|  0D6040C2 F933381E  7A4C4797 F40FEDAB  | ?-?B9???:<?p4??? | ??@??38?zLG????? |
...
```

- To display the full record information of the same disk when it in raw-track access mode:

```
# dasdview -b 2c -s 1t /dev/dasdk
```

This example displays:

```
cylinder 2, head 0, record 0
+------------------------------------------------------------------------------+
| count area:                                                                  |
|          hex: 0002000000000008                                               |
|     cylinder:                2                                               |
|         head:                0                                               |
|       record:                0                                               |
|   key length:                0                                               |
|  data length:                8                                               |
+------------------------------------------------------------------------------+
| key area:                                                                    |
| HEXADECIMAL                             | EBCDIC            | ASCII           |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+----------------------------------------+------------------+------------------+
+----------------------------------------+------------------+------------------+
| data area:                                                                   |
| HEXADECIMAL                             | EBCDIC            | ASCII           |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+----------------------------------------+------------------+------------------+
|  00000000 00000000                     | ................ | ................ |
+----------------------------------------+------------------+------------------+

cylinder 2, head 0, record 1
+------------------------------------------------------------------------------+
| count area:                                                                  |
|          hex: 0002000001000200                                               |
|     cylinder:                2                                               |
|         head:                0                                               |
|       record:                1                                               |
|   key length:                0                                               |
|  data length:              512                                               |
+------------------------------------------------------------------------------+
| key area:                                                                    |
| HEXADECIMAL                             | EBCDIC            | ASCII           |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+----------------------------------------+------------------+------------------+
+----------------------------------------+------------------+------------------+
| data area:                                                                   |
| HEXADECIMAL                             | EBCDIC            | ASCII           |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+----------------------------------------+------------------+------------------+
|  52B7DBEE D6B9530B  0179F420 CB6EA95E  | ????O?????4??>z; | R?????S??y???n?^ |
|  EF49C03C 513542E7  D8F17D9D 06DC44F7  | ??{????XQ1'????7 | ?I?<Q5B???}???D? |
...
+----------------------------------------+------------------+------------------+

cylinder 2, head 0, record 2
+------------------------------------------------------------------------------+
| count area:                                                                  |
|          hex: 0002000002000200                                               |
|     cylinder:                2                                               |
|         head:                0                                               |
|       record:                2                                               |
|   key length:                0                                               |
|  data length:              512                                               |
+------------------------------------------------------------------------------+
| key area:                                                                    |
| HEXADECIMAL                             | EBCDIC            | ASCII           |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+----------------------------------------+------------------+------------------+
+----------------------------------------+------------------+------------------+
| data area:                                                                   |
| HEXADECIMAL                             | EBCDIC            | ASCII           |
|  01....04 05....08  09....12 13....16  | 1.............16 | 1.............16 |
+----------------------------------------+------------------+------------------+
|  92963D5B 0200B0FA  53745C12 C3B45125  | ko?$?.^???*?C??? | ??=[?.??St\???Q% |
|  0D6040C2 F933381E  7A4C4797 F40FEDAB  | ?-?B9???:<?p4??? | ??@??38?zLG????? |
...
```

- To display the contents of a disk, which is in raw-access mode, printing one track of data from the start of the disk:

```
# dasdview -s 1t /dev/dasdk
```

This example displays:

```
cylinder 0, head 0, record 0
+-------------------------------------------------------------------------------+
| count area:                                                                   |
|          hex: 0000000000000008                                                |
|     cylinder:                0                                                |
|         head:                0                                                |
|       record:                0                                                |
|   key length:                0                                                |
|  data length:                8                                                |
+-------------------------------------------------------------------------------+
| key area:                                                                     |
| HEXADECIMAL                             | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16   | 1.............16 | 1.............16 |
+-----------------------------------------+------------------+------------------+
+-----------------------------------------+------------------+------------------+
| data area:                                                                    |
| HEXADECIMAL                             | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16   | 1.............16 | 1.............16 |
+-----------------------------------------+------------------+------------------+
|  00000000 00000000                      | ................ | ................ |
+-----------------------------------------+------------------+------------------+

cylinder 0, head 0, record 1
+-------------------------------------------------------------------------------+
| count area:                                                                   |
|          hex: 0000000001040018                                                |
|     cylinder:                0                                                |
|         head:                0                                                |
|       record:                1                                                |
|   key length:                4                                                |
|  data length:               24                                                |
+-------------------------------------------------------------------------------+
| key area:                                                                     |
| HEXADECIMAL                             | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16   | 1.............16 | 1.............16 |
+-----------------------------------------+------------------+------------------+
|  C9D7D3F1                               | IPL1............ | ????............ |
+-----------------------------------------+------------------+------------------+
| data area:                                                                    |
| HEXADECIMAL                             | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16   | 1.............16 | 1.............16 |
+-----------------------------------------+------------------+------------------+
|  000A0000 0000000F  03000000 00000001   | .?.....??......? | .?.....??......? |
|  00000000 00000000                      | ................ | ................ |
+-----------------------------------------+------------------+------------------+
...
```

fdasd - Partition a DASD

Use the **fdasd** command to manage partitions on ECKD-type DASD that were formatted with the compatible disk layout.

See “dasdfmt - Format a DASD” on page 575 for information about formatting a DASD. With **fdasd** you can create, change, and delete partitions, and also change the volume serial number.

fdasd checks that the volume has a valid volume label and VTOC. If either is missing or incorrect, **fdasd** re-creates it. See “System z compatible disk layout” on page 147 for details about the volume label and VTOC.

Calling **fdasd** with a node, but without options, enters interactive mode. In interactive mode, you are given a menu through which you can display DASD information, add or remove partitions, or change the volume identifier. Your changes are not written to disk until you type the `write` option on the menu. You can quit without altering the disk at any time before this.

For more information about partitions, see “The IBM label partitioning scheme” on page 146.

Before you begin:
- To partition a SCSI disk, use **fdisk** rather than **fdasd**.
- The disk must be formatted with **dasdfmt**, using the compatible disk layout.

Attention: Careless use of **fdasd** can result in loss of data.

fdasd syntax

```
►►──fdasd──┬──────┬──┬───────────────────────────┬────────────────────────────►
           ├─ -s──┤  ├─ -a──┬──────────────────┬─┤
           └─ -r──┘  │      ├─ -k──────────────┤ │
                     │      │             (1)  │ │
                     │      └─ -l <volser>─────┘ │
                     ├─ -c <conf_file>───────────┤
                     ├─ -i───────────────────────┤
                     └─ -p───────────────────────┘

►──┬──────────────────────────────────────┬── <node>──────────────────────────►◄
   └─ -f──┬─3390,4096─────────────────┬───┘
          └─<device_type>,<blocksize>─┘
```

Notes:

1 If neither the **-l** option nor the **-k** option is specified, a VOLSER is generated from the device number through which the volume is accessed.

Where:

-s or --silent
suppresses messages.

-r or --verbose
displays additional messages that are normally suppressed.

-a or --auto
auto-creates one partition using the whole disk in non-interactive mode.

-k or --keep_serial
keeps the volume serial number when writing the volume label (see VOLSER). Keeping the serial number is useful, for example, if the volume serial number was written with a z/VM tool and should not be overwritten.

-l *<volser>* **or --label** *<volser>*
specifies the volume serial number (see VOLSER).

A volume serial consists of one through six alphanumeric characters or the following special characters:

$ # @ %

All other characters are ignored. Avoid using special characters in the volume serial. Special characters can cause problems when accessing a disk by VOLSER. If you must use special characters, enclose the VOLSER in single quotation marks. In addition, any '$' character in the VOLSER must be preceded by a backslash ('\').

For example, specify:

```
-l 'a@b\$c#'
```

to get:

```
A@B$C#
```

VOLSER is interpreted as an ASCII string and is automatically converted to uppercase, padded with blanks and finally converted to EBCDIC before it is written to disk.

Do not use the following reserved volume serials:
- SCRTCH
- PRIVAT
- MIGRAT
- Lnnnnn (L followed by a five-digit number)

These volume serials are used as keywords by other System z operating systems, such as z/OS.

Omitting this parameter causes **fdasd** to prompt for it, if it is needed.

-c *<conf_file>* **or --config** *<conf_file>*
creates partitions, in non-interactive mode, according to specifications in the configuration file *<conf_file>*.

For each partition you want to create, add one line of the following format to *<conf_file>*:

[*<first_track>*,*<last_track>*,*<type>*]

<first_track> and *<last_track>* are required and specify the first and last track of the partition. You can use the keyword **first** for the first possible track on the disk and the keyword **last** for the last possible track on the disk.

<type> describes the partition type and is one of:

native
for partitions to be used for Linux file systems.

gpfs
for partitions to be used as part of an Elastic Storage file system setup.

swap
for partitions to be used as swap devices.

raid
for partitions to be used as part of a RAID setup.

lvm
for partitions to be used as part of a logical volume group.

The type specification is optional. If the type is omitted, `native` is used.

The type describes the intended use of a partition to tools or other operating systems. For example, swap partitions could be skipped by backup programs. How Linux actually uses the partition depends on how the partition is formatted and set up. For example, a partition of type `native` can still be used in an LVM logical volume or in a RAID configuration.

Example: With the following sample configuration file, you can create three partitions:

```
[first,1000,raid]
[1001,2000,swap]
[2001,last]
```

-i or --volser
displays the volume serial number and exits.

-p or --table
displays the partition table and exits.

-f or --force
specifies values for the disk geometry instead of detecting them.

Note: Specifying incorrect values can render the disk unusable. Do not use this option if **fdasd** can automatically detect the disk geometry. For example, do not use the force option for native DASD or any disk with the disk geometry of a type 3390 DASD.

<device_type>,*<blocksize>*
specifies the disk device type. Valid device types are: 3390, 3380, and 9345. Valid block sizes are: 4096, 2048, 1024, and 512. The default specification is the combination of disk type 3390 with block size 4096.

For disks with the default geometry, you can omit the specifications for the device type and block size, and the following specifications are all valid:

- **-f**
- **-f***3390,4096*
- **--force**
- **--force**=*3390,4096*

For all other disks, you must specify both values with the command. The following specifications are all valid for a disk of type 3390 and block size 512:

- **-f***3390,512*
- **--force**=*3390,512*

Use the verbose option for information about the disk geometry as computed from the specified or default device type and block size.

<node>
specifies the device node of the DASD you want to partition, for example, `/dev/dasdzzz`. See "DASD naming scheme" on page 151 for more details about device nodes.

`-h or --help`
displays help on command-line arguments. To view the man page, enter **`man fdasd`**.

`-v or --version`
displays the version of **`fdasd`**.

fdasd menu

If you call **`fdasd`** in the interactive mode (that is, with just a node), a menu is displayed.

```
Command action
   m print this menu
   p print the partition table
   n add a new partition
   d delete a partition
   v change volume serial
   t change partition type
   r re-create VTOC and delete all partitions
   u re-create VTOC re-using existing partition sizes
   s show mapping (partition number - data set name)
   q quit without saving changes
   w write table to disk and exit

Command (m for help):
```

fdasd menu commands

Use the **`fdasd`** menu commands to modify or view information about DASDs

`m` re-displays the **`fdasd`** command menu.

`p` displays information about the DASD and the partitions.

DASD information:
- Number of cylinders
- Number of tracks per cylinder
- Number of blocks per track
- Block size
- Volume label
- Volume identifier
- Number of partitions defined

Partition information:
- Linux node
- Start track
- End track
- Number of tracks
- Partition ID
- Partition type

There is also information about the free disk space that is not used for a partition.

n adds a partition to the DASD. You are asked to give the start track and the length or end track of the new partition.

d deletes a partition from the DASD. You are asked which partition to delete.

v changes the volume identifier. You are asked to enter a new volume identifier. See VOLSER for the format.

t changes the partition type. You are prompted for the partition to be changed and for the new partition type.

Changing the type changes the disk description but does not change the disk itself. How Linux uses the partition depends on how the partition is formatted and set up. For example, as an LVM logical volume or in a RAID configuration.

The partition type describes the partition to other operating systems so that; for example, swap partitions can be skipped by backup programs.

r re-creates the VTOC and deletes all partitions.

u re-creates all VTOC labels without removing all partitions. Existing partition sizes are reused. This option is useful to repair damaged labels or migrate partitions that are created with older versions of **fdasd**.

s displays the mapping of partition numbers to data set names. For example:

```
Command (m for help): s

device .........: /dev/dasdzzz
volume label ...: VOL1
volume serial ..: 0X0193

WARNING: This mapping may be NOT up-to-date,
         if you have NOT saved your last changes!

/dev/dasdzzz1  -  LINUX.V0X0193.PART0001.NATIVE
/dev/dasdzzz2  -  LINUX.V0X0193.PART0002.NATIVE
/dev/dasdzzz3  -  LINUX.V0X0193.PART0003.NATIVE
```

q quits **fdasd** without updating the disk. Any changes that you have made (in this session) are discarded.

w writes your changes to disk and exits. After the data is written, Linux rereads the partition table.

Example using the menu

This example shows how to use **fdasd** to create two partitions on a z/VM minidisk, change the type of one of the partitions, save the changes, and check the results.

About this task

This example shows you how to format a z/VM minidisk with the compatible disk layout. The minidisk has device number 193.

Procedure

1. Call **fdasd**, specifying the minidisk:

```
# fdasd /dev/dasdzzz
```

fdasd reads the existing data and displays the menu:

```
reading volume label: VOL1
reading vtoc : ok

Command action
   m print this menu
   p print the partition table
   n add a new partition
   d delete a partition
   v change volume serial
   t change partition type
   r re-create VTOC and delete all partitions
   u re-create VTOC re-using existing partition sizes
   s show mapping (partition number - data set name)
   q quit without saving changes
   w write table to disk and exit
Command (m for help):
```

2. Use the p option to verify that no partitions are created yet on this DASD:

```
Command (m for help): p

Disk /dev/dasdzzz:
  cylinders ............: 100
  tracks per cylinder ..: 15
  blocks per track .....: 12
  bytes per block ......: 4096
  volume label .........: VOL1
  volume serial ........: 0X0193
  max partitions .......: 3

 ------------------------------- tracks -------------------------------
              Device      start      end   length   Id  System
                              2     1499     1498       unused
```

3. Define two partitions, one by specifying an end track and the other by specifying a length. (In both cases the default start tracks are used):

```
Command (m for help): n
First track (1 track = 48 KByte) ([2]-1499):
Using default value 2
Last track or +size[c|k|M] (2-[1499]): 700
You have selected track 700
```

```
Command (m for help): n
First track (1 track = 48 KByte) ([701]-1499):
Using default value 701
Last track or +size[c|k|M] (701-[1499]): +400
You have selected track 1100
```

4. Check the results by using the p option:

```
Command (m for help): p

Disk /dev/dasdzzz:
  cylinders ............: 100
  tracks per cylinder ..: 15
  blocks per track .....: 12
  bytes per block ......: 4096
  volume label .........: VOL1
  volume serial ........: 0X0193
  max partitions .......: 3

 ------------------------------- tracks -------------------------------
              Device      start      end   length   Id  System
       /dev/dasdzzz1          2      700      699    1  Linux native
       /dev/dasdzzz2        701     1100      400    2  Linux native
                           1101     1499      399       unused
```

5. Change the type of a partition:

```
Command (m for help): t

Disk /dev/dasdzzz:
  cylinders ............: 100
  tracks per cylinder ..: 15
  blocks per track .....: 12
  bytes per block ......: 4096
  volume label .........: VOL1
  volume serial ........: 0X0193
  max partitions .......: 3

 ------------------------------- tracks -------------------------------
              Device      start      end   length   Id  System
       /dev/dasdzzz1          2      700      699    1  Linux native
       /dev/dasdzzz2        701     1100      400    2  Linux native
                           1101     1499      399       unused

change partition type
partition id (use 0 to exit):
```

Enter the ID of the partition you want to change; in this example partition 2:

```
partition id (use 0 to exit): 2
```

6. Enter the new partition type; in this example type 2 for swap:

```
current partition type is: Linux native

    1 Linux native
    2 Linux swap
    3 Linux raid
    4 Linux lvm

new partition type: 2
```

7. Check the result:

```
Command (m for help): p


Disk /dev/dasdzzz:
  cylinders ............: 100
  tracks per cylinder ..: 15
  blocks per track .....: 12
  bytes per block ......: 4096
  volume label .........: VOL1
  volume serial ........: 0X0193
  max partitions .......: 3

 ------------------------------- tracks -------------------------------
              Device      start      end   length   Id  System
       /dev/dasdzzz1          2      700      699    1  Linux native
       /dev/dasdzzz2        701     1100      400    2  Linux swap
                           1101     1499      399       unused
```

8. Write the results to disk with the `w` option:

```
Command (m for help): w
writing VTOC...
rereading partition table...
#
```

Example using options

You can partition a DASD by using the **-a** or **-c** option without entering the menu mode.

This method is useful for partitioning with scripts, for example, if you need to partition several hundred DASDs.

With the `-a` parameter you can create one large partition on a DASD:

```
# fdasd -a /dev/dasdzzz
auto-creating one partition for the whole disk...
writing volume label...
writing VTOC...
rereading partition table...
#
```

This command creates a partition as follows:

```
        Device      start      end   length   Id  System
  /dev/dasdzzz1         2     1499     1498    1  Linux native
```

Using a configuration file, you can create several partitions. For example, the following configuration file, `config`, creates three partitions:

```
[first,500]
[501,1100,swap]
[1101,last]
```

Submitting the command with the `-c` option creates the partitions:

```
# fdasd -c config /dev/dasdzzz
parsing config file 'config'...
writing volume label...
writing VTOC...
rereading partition table...
#
```

This command creates partitions as follows:

```
        Device      start      end   length   Id  System
  /dev/dasdzzz1         2      500      499    1  Linux native
  /dev/dasdzzz2       501     1100      600    2  Linux swap
  /dev/dasdzzz3      1101     1499      399    3  Linux native
```

hyptop - Display hypervisor performance data

Use the **hyptop** command to obtain a dynamic real-time view of a hypervisor environment on System z.

It works with both the z/VM hypervisor and the LPAR hypervisor, Processor Resource/Systems Manager™ (PR/SM™). Depending on the available data, it shows, for example, CPU and memory information about LPARs or z/VM guest virtual machines. The **hyptop** command provides two main windows:

- A list of systems that the hypervisor is currently running (sys_list).
- One system in more detail (sys).

You can run **hyptop** in interactive mode (default) or in batch mode with the **-b** option.

Before you begin: The following things are required to run **hyptop**:

- The debugfs file system must be mounted.
- The Linux kernel must have the required support to provide the performance data. Check that *<debugfs mount point>*/s390_hypfs is available after you mount debugfs.
- The hyptop user must have read permission for the required debugfs files:
  - z/VM: *<debugfs mount point>*/s390_hypfs/diag_2fc
  - LPAR: *<debugfs mount point>*/s390_hypfs/diag_204
- To monitor all LPARs or z/VM guest virtual machines, your system must have additional permissions:
  - For z/VM: The guest virtual machine must be class B.
  - For LPAR: On the HMC or SE security menu of the LPAR activation profile, select the **Global performance data control** check box.

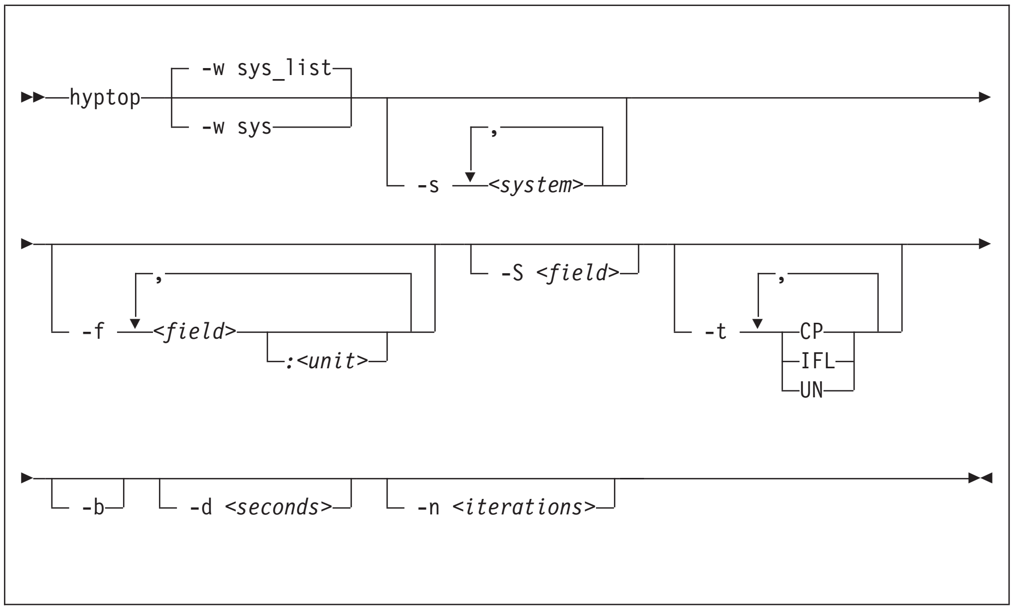
To mount debugfs, you can use this command:

```
# mount none -t debugfs /sys/kernel/debug
```

To make this setting persistent, add the following to /etc/fstab:

```
none     /sys/kernel/debug     debugfs defaults     0 0
```

hyptop syntax



Where:

-w *<window name>* **or --window=***<window name>*
: selects the window to display, either sys or sys_list. Use the options **--sys**, **--fields**, and **--sort** to modify the current window. The last window that is specified with the **--window** option is used as the start window. The default window is sys_list.

-s *<system>* **or --sys=***<system>*
: selects systems for the current window. If you specify this option, only the selected systems are shown in the window. For the sys window, you can specify only one system.

-f *<field>***[:***<unit>***] or --fields=***<field>***[:***<unit>***]**
: selects fields and units in the current window. The *<field>* variable is a one letter unique identifier for a field (for example "c" for CPU time). The *<unit>* variable specifies the unit that is used for the field (for example "us" for microseconds). See "Available fields and units" on page 605 for definitions. If the **--fields** option is specified, only the selected fields are shown.

-S *<field>* **or --sort=***<field>*
: selects the field that is used to sort the data in the current window. To reverse the sort order, specify the option twice. See "Available fields and units" on page 605 for definitions.

-t *<type>* **or --cpu_types=***<type>*
: selects CPU types that are used for CPU time calculations. See "CPU types" on page 608 for definitions.

-b or --batch_mode
: uses batch mode. Batch mode can be useful for sending output from **hyptop** to another program, a file, or a line mode terminal. In this mode no user input is accepted.

-d *<seconds>* **or --delay=***<seconds>*
: specifies the delay between screen updates.

-n *<iterations>* **or --iterations=***<iterations>*
specifies the maximum number of screen updates before the program ends.

-h or --help
prints usage information, then exits. To view the man page, enter **man hyptop**.

-v or --version
displays the version of **hyptop**, then exits.

Navigating between windows

Use letter or arrow keys to navigate between the windows.

When you start the **hyptop** command, the sys_list window opens in normal mode. Data is updated at regular intervals, and sorted by CPU time. You can navigate between the windows as shown in Figure 114.

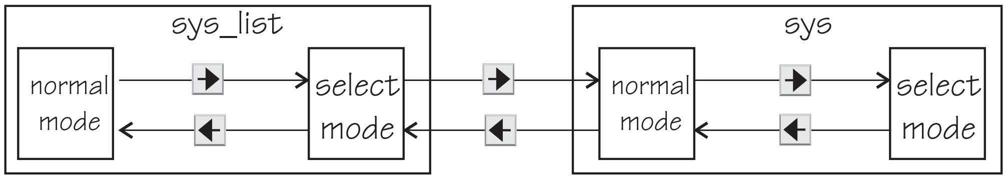


Figure 114. hyptop window navigation overview

To navigate between the windows, use the ← and → arrow keys. The windows have two modes, normal mode and select mode.

You can get online help for every window by pressing the ? key. Press Q in the sys_list window to exit hyptop.

Instead of using the arrow keys, you can use letter keys (equivalent to the vi editor navigation) in all windows as listed in Table 62.

Table 62. Using letter keys instead of arrow keys

Arrow key	**Letter key equivalent**
←	H
↓	J
↑	K
→	L

Selecting data

You can scroll windows and select data rows.

To enter select mode, press the → key. The display is frozen so that you can select rows. Select rows by pressing the ↑ and ↓ keys and mark the rows with the Spacebar. Marked rows are displayed in bold font. Leave the select mode by pressing the ← key.

To see the details of one system, enter select mode in the sys_list window, then navigate to the row for the system you want to look at, and press the → key. The sys window for the system opens. The ← key always returns you to the previous window.

To scroll any window, press the ↑ and ↓ keys or the Page Up and Page Down keys. Jump to the end of a window by pressing the Shift + G keys and to the beginning by pressing the G key.

Sorting data

You can sort data according to column.

The sys window or sys_list window table is sorted according to the values in the selected column. Select a column by pressing the hot key of the column. This key is underlined in the heading. If you press the hot key again, the sort order is reversed. Alternatively, you can select columns with the < and > keys.

Filtering data

You can filter the displayed data by CPU types and by data fields.

From the sys or sys_list window you can access the fields selection window and the CPU-type selection window as shown in Figure 115.

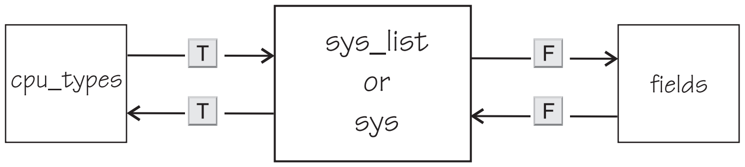


Figure 115. Accessing the fields and CPU-type selection windows

Use the T key to toggle between the CPU-type selection window and the main window. Use the F key to toggle between the fields selection window and the main window. You can also use the ← key to return to the main window from the CPU types and fields windows.

In the fields and CPU-type selection windows, press the field or CPU type identifier key (see “LPAR fields” on page 606, “z/VM fields” on page 606, and “CPU types” on page 608) to select or de-select. Selected rows are bold and de-selected rows are grey. When you return to the main window, the data is filtered according to your field and CPU type selections.

Available fields and units

Different fields are supported depending whether your hypervisor is LPAR PR/SM or z/VM.

The fields might also be different depending on machine type, z/VM version, and kernel version. Each field has a unique one letter identifier that can be used in interactive mode to enable the field in the field selection window. Also, use it to select the sort field in the sys or sys_list window. You can also select fields and sort data using the **--fields** and **--sort** command line options.

LPAR fields

Some fields for Linux in LPAR mode are available in both the sys_list and sys windows others are available only in the sys_list window or only in the sys window.

Identifier	Column label	Explanation
c	cpu	CPU time per second
m	mgm	Management time per second
C	Cpu+	Total CPU time
M	Mgm+	Total management time
o	online	Online time

In the sys_list window only:

Identifier	Column label	Explanation
y	system	Name of the LPAR (always shown)
#	#cpu	Number of CPUs

In the sys window only:

Identifier	Column label	Explanation
i	cpuid	CPU identifier (always shown)
p	type	CPU type. See “CPU types” on page 608
v	visual	Visualization of CPU time per second

z/VM fields

Some fields for Linux on z/VM are available in both the sys_list and sys windows. Others are available only in the sys_list window or only in the sys window.

In the sys_list and sys windows:

Identifier	Column label	Explanation
c	cpu	CPU time per second
C	Cpu+	Total CPU time
o	online	Online time

In the sys_list window only:

Identifier	Column label	Explanation
y	system	Name of the z/VM guest virtual machine (always shown)
#	#cpu	Number of CPUs
O	#cpuop	Number of operating CPUs
u	memuse	Used memory
a	memmax	Maximum memory

Identifier	Column label	Explanation
r	wcur	Current weight
x	wmax	Maximum weight

In the sys window only:

Identifier	Column label	Explanation
i	cpuid	CPU identifier (always shown)
v	visual	Visualization of CPU time per second

Units

Depending on the field type, the values can be displayed in different units.

In the sys_list and sys windows, the units are displayed under the column headings in parenthesis. Each unit can be specified through the **--fields** command line option. Units can also be selected interactively. To change a unit, enter select mode in the fields window. Then, select the field where you want to change the unit, and press the "+" or "-" keys to go through the available units. The following units are supported:

Units of time:

Unit	Explanation
us	Microseconds (10^{-6} seconds)
ms	Milliseconds (10^{-3} seconds)
%	Hundreds of a second (10^{-2} seconds) or percent
s	Seconds
m	Minutes
hm	Hours and minutes
dhm	Days, hours, and minutes

Units of memory:

Unit	Explanation
KiB	Kibibytes (1 024 bytes)
MiB	Mebibytes (1 048 576 bytes)
GiB	Gibibytes (1 073 741 824 bytes)

Other units:

Unit	Explanation
str	String
#	Count or number
vis	Visualization

CPU types

Enable or disable CPU types in interactive mode in the cpu_types window.

The CPU types can also be specified with the `--cpu_types` command line option.

The calculation of the CPU data uses CPUs of the specified types only. For example, if you want to see how much CPU time is consumed by your Linux systems, enable CPU type IFL.

On z/VM the processor type is always UN and you cannot select the type.

In an LPAR the following CPU types can be selected either interactively or with the `--cpu_types` command line option:

Identifier	Column label	Explanation
i	IFL	Integrated Facility for Linux. On older machines IFLs might be shown as CPs.
p	CP	CP processor type.
u	UN	Unspecified processor type (other than CP or IFL).

Examples

These examples show typical uses of **hyptop**.

- To start **hyptop** with the sys_list window in interactive mode, enter:

```
# hyptop
```

  – If your Linux instance is running in an LPAR that has permission to see the other LPARs, the output looks like the following example:

```
12:30:48 | CPU-T: IFL(18)  CP(3)  UN(3)                           ?=help
system   #cpu   cpu     mgm    Cpu+   Mgm+   online
(str)     (#)   (%)     (%)    (hm)   (hm)   (dhm)
S05LP30   10   461.14  10.18  1547:41 8:15 11:05:59
S05LP33    4   133.73   7.57  220:53  6:12 11:05:54
S05LP50    4    99.26   0.01  146:24  0:12 10:04:24
S05LP02    1    99.09   0.00  269:57  0:00 11:05:58
TRX2CFA    1     2.14   0.03    3:24  0:04 11:06:01
S05LP13    6     1.36   0.34    4:23  0:54 11:05:56
TRX1      19     1.22   0.14   13:57  0:22 11:06:01
TRX2      20     1.16   0.11   26:05  0:25 11:06:00
S05LP55    2     0.00   0.00    0:22  0:00 11:05:52
S05LP56    3     0.00   0.00    0:00  0:00 11:05:52
         413   823.39  23.86 3159:57 38:08 11:06:01
```

  – If your Linux instance runs in a z/VM guest virtual machine that has permission to see the other z/VM guest virtual machines, the output looks like the following example:

```
12:32:21 | CPU-T: UN(16)                                           ?=help
system    #cpu    cpu    Cpu+    online  memuse  memmax  wcur
(str)      (#)    (%)    (hm)     (dhm)   (GiB)   (GiB)   (#)
T6360004   6   100.31  959:47 53:05:20    1.56    2.00    100
DTCVSW1    1     0.00    0:00 53:16:42    0.01    0.03    100
T6360002   6     0.00  166:26 40:19:18    1.87    2.00    100
OPERATOR   1     0.00    0:00 53:16:42    0.00    0.03    100
T6360008   2     0.00    0:37 30:22:55    0.32    0.75    100
T6360003   6     0.00 3700:57 53:03:09    4.00    4.00    100
NSLCF1     1     0.00    0:02 53:16:41    0.03    0.25    500
PERFSVM    1     0.00    0:53 2:21:12     0.04    0.06      0
TCPIP      1     0.00    0:01 53:16:42    0.01    0.12   3000
DIRMAINT   1     0.00    0:04 53:16:42    0.01    0.03    100
DTCVSW2    1     0.00    0:00 53:16:42    0.01    0.03    100
RACFVM     1     0.00    0:00 53:16:42    0.01    0.02    100
          75   101.57 5239:47 53:16:42   15.46   22.50   3000
```

At the top of the sys and sys_list windows the CPU types currently used for CPU time calculation are displayed.

- To start **hyptop** with the sys window showing performance data for LPAR MYLPAR, enter:

```
# hyptop -w sys -s mylpar
```

The result looks like the following example:

```
11:18:50 MYLPAR CPU-T: IFL(0) CP(24) UN(2)                              ?=help
cpuid  type    cpu  mgm visual
(#)    (str)    (%)  (%) (vis)
0        CP  50.78 0.28 |#######################                       |
1        CP  62.76 0.17 |############################                  |
2        CP  71.11 0.48 |################################              |
3        CP  32.38 0.24 |###############                               |
4        CP  64.35 0.32 |#############################                 |
5        CP  67.61 0.40 |##############################                |
6        CP  70.95 0.35 |################################              |
7        CP  62.16 0.41 |############################                  |
8        CP  70.48 0.25 |################################              |
9        CP  56.43 0.20 |#########################                     |
10       CP   0.00 0.00 |                                              |
11       CP   0.00 0.00 |                                              |
12       CP   0.00 0.00 |                                              |
13       CP   0.00 0.00 |                                              |
=:V:N       609.02 3.10
```

- To start **hyptop** with the sys_list window in batch mode, enter:

```
# hyptop -b
```

- To start **hyptop** with the sys_list window in interactive mode, with the fields CPU time (in milliseconds), and online time (unit default), and sort the output according to online time, enter:

```
# hyptop -f c:ms,o -S o
```

- To start **hyptop** with the sys_list window in batch mode with update delay 5 seconds and 10 iterations, enter:

```
# hyptop -b -d 5 -n 10
```

- To start **hyptop** with the sys_list window and use only CPU types IFL and CP for CPU time calculation, enter:

```
# hyptop -t ifl,cp
```

Scenario

Perform the steps described in this scenario to start **hyptop** with the sys window with system MYLPAR with the fields CPU time (unit milliseconds) and Total CPU time (unit default), sort the output according to the Total CPU time, and then reverse the sort order.

Procedure

1. Start hyptop.

```
# hyptop
```

2. Go to select mode by pressing the → key. The display will freeze.
3. Navigate to the row for the system you want to look (in the example MYLPAR) at using the ↑ and ↓ keys.

```
12:15:00 | CPU-T: IFL(18)  CP(3)  UN(3)                                 ?=help
system   #cpu   cpu     mgm    Cpu+   Mgm+   online
(str)     (#)   (%)     (%)    (hm)   (hm)   (dhm)
 MYLPAR     4   199.69   0.04   547:41 8:15 11:05:59
S05LP33     4   133.73   7.57  220:53  6:12 11:05:54
S05LP50     4    99.26   0.01  146:24  0:12 10:04:24
S05LP02     1    99.09   0.00  269:57  0:00 11:05:58
...
S05LP56     3     0.00   0.00    0:00  0:00 11:05:52
          413   823.39  23.86 3159:57 38:08 11:06:01
```

4. Open the sys window for MYLPAR by pressing the → key.

```
12:15:51 MYLPAR CPU-T: IFL(18) CP(3) UN(2)                             ?=help
cpuid  type    cpu  mgm visual
(#)   (str)    (%)  (%) (vis)
0       IFL  99.84 0.02 |########################################
1       IFL  99.85 0.02 |########################################
2       IFL   0.00 0.00 |
3       IFL   0.00 0.00 |
=:V:N       199.69 0.04
```

5. Press the F key to go to the fields selection window:

```
Select Fields and Units                                                 ?=help
K S ID     UNIT AGG  DESCRIPTION
p * type   str  none CPU type
c * cpu    %    sum  CPU time per second
m * mgm    %    sum  Management time per second
C   cpu+   hm   sum  Total CPU time
M   mgm+   hm   sum  Total management time
o   online dhm  max  Online time
v * visual vis  none Visualization of CPU time per second
```

Ensure that CPU time per second and Total CPU time are selected and for CPU time microseconds are used as unit:

a. Press the P key, the M key, and the V key to disable `CPU type`, `Management time per second`, and `Visualization`.

b. Press the C key to enable `Total CPU time`.

c. Then select the `CPU time per second` row by pressing the → and ↓ keys.

d. Press the minus key (-) to switch from the percentage (%) unit to the microseconds (ms) unit.

```
Select Fields and Units ?=help
K S ID     UNIT AGG  DESCRIPTION
p   type   str  none CPU type
c * cpu    ms   sum  CPU time per second
m   mgm    %    sum  Management time per second
C * cpu+   hm   sum  Total CPU time
M   mgm+   hm   sum  Total management time
o   online dhm  max  Online time
v   visual vis  none Visualization of CPU time per second
```

Press the ← key twice to return to the sys window.

6. To sort by Total CPU time press the Shift + C keys:

```
13:44:41 MYLPAR CPU-T: IFL(18) CP(3) UN(2)               ?=help
cpuid    cpu    Cpu+
(#)     (ms)    (hm)
0      23.84  548:52
1      37.48  492:55
3       0.00    0:00
2       0.00    0:00
=:^:N  61.33 1041:47
```

To reverse the sort order, press the Shift + C keys again:

```
13:44:41 MYLPAR CPU-T: IFL(18) CP(3) UN(2)               ?=help
cpuid    cpu    Cpu+
(#)     (ms)    (hm)
2       0.00    0:00
3       0.00    0:00
1      37.48  492:55
0      23.84  548:52
=:^:N  61.33 1041:47
```

Results

You can do all of these steps in one by entering the command:

```
# hyptop -w sys -s mylpar -f c:ms,C -S C -S C
```

lschp - List channel paths

Use the **lschp** command to display information about channel paths.

lschp syntax

```
►►──lschp──┬───────────┬──────────────────────────────────────────►◄
           ├─ --help───┤
           └─ --version┘
```

where:

-v or --version
: displays the version number of **lschp** and exits.

-h or --help
: displays out a short help text, then exits. To view the man page, enter **man lschp**.

Output column description:

CHPID
: Channel-path identifier.

Vary
: Logical channel-path state:
  - 0 = channel-path is not used for I/O.
  - 1 = channel-path is used for I/O.

Cfg.
: Channel-path configure state:
  - 0 = stand-by
  - 1 = configured
  - 2 = reserved
  - 3 = not recognized

Type
: Channel-path type identifier.

Cmg
: Channel measurement group identifier.

Shared
: Indicates whether a channel path is shared between LPARs:
  - 0 = channel path is not shared
  - 1 = channel path is shared

PCHID
: Physical channel path identifier, or, if enclosed in brackets, internal channel identifier. The mapping might not be available to Linux when it is running as a z/VM guest. If so, use the CP command:

```
QUERY CHPID <num> PCHID
```

A column value of '-' indicates that a facility associated with the corresponding channel-path attribute is not available.

Examples

- To query the configuration status of channel path ID 0.40 issue:

```
# lschp

CHPID Vary Cfg. Type Cmg Shared  PCHID
======================================
...
...
0.40  1    1    1b   2   1       0580
...
...
```

The value under **Cfg.** shows that the channel path is configured (1).

lscpumf - Display information about the CPU-measurement facilities

Use the **lscpumf** command to display information about information about the CPU-measurement facilities.

lscpumf syntax

```
►►──lscpumf──┬─────┬──────────────────────────────────────────────────►◄
             ├─ -i─┤
             ├─ -c─┤
             ├─ -C─┤
             ├─ -s─┤
             ├─ -h─┤
             └─ -v─┘
```

where:

-i or --info
displays detailed information about available and supported CPU measurement facilities.

-c or --list-counters
lists counters that are provided by the CPU-measurement facility, omitting counters for which the LPAR is not authorized. For counter measurements with the perf program, the raw event identifier and symbolic counter name are displayed.

-C or --list-all-counters
lists all counters that are provided by the CPU-measurement counter facility, regardless of LPAR authorization. To list only those counters for which the LPAR is authorized, use the **-c** option. For counter measurements with the perf program, the raw event identifier and symbolic counter name are displayed.

-s or --list-sampling-events
lists perf raw events that activate the sampling facility.

-v or --version
displays the version number of **lscpumf** and exits.

-h or --help
displays out a short help text, then exits. To view the man page, enter **man lscpumf**.

Examples

- To display the supported facilities, issue:

```
# lscpumf
CPU-measurement Counter Facility
CPU-measurement Sampling Facility
```

- To display details about the facilities, issue:

```
# lscpumf -i
CPU-measurement counter facility
---------------------------------------------------------------------------
Version: 1.2

Authorized counter sets:
    Basic counter set
    Problem-State counter set

Linux perf event support: Yes (PMU: cpum_cf)


CPU-measurement sampling facility
---------------------------------------------------------------------------
Sampling Interval:
     Minimum:      18228 cycles (approx.   285714 Hz)
     Maximum:  170650536 cycles (approx.       30 Hz)

Authorized sampling modes:
     basic      (sample size:  32 bytes)


Linux perf event support: Yes (PMU: cpum_sf)

Current sampling buffer settings for cpum_sf:
    Basic-sampling mode
        Minimum:     15 sample-data-blocks (  64KB)
        Maximum:   8176 sample-data-blocks (  32MB)
```

- To display perf event information for authorized sampling functions, issue:

```
# lscpumf -s
Perf events for activating the sampling facility
===============================================================================

Raw
event   Name    Description
-------------------------------------------------------------------------------
rb0000  SF_CYCLES_BASIC

                Sample CPU cycles using basic-sampling mode.
                This event is not associated with a counter set.
```

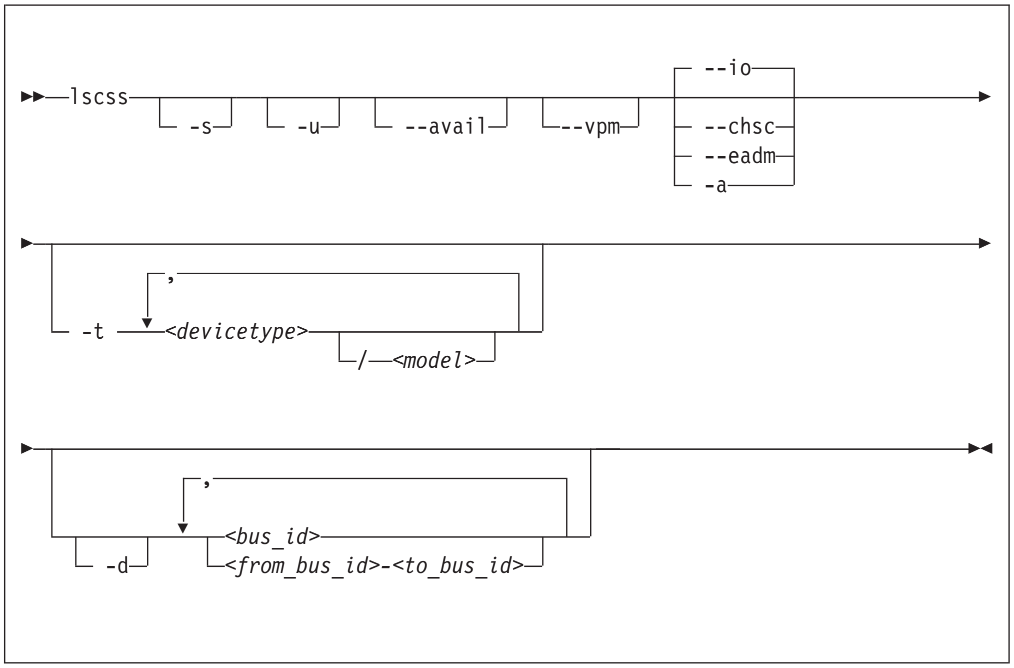
- To list all counters that are provided by your System z hardware, issue:

```
# lscpumf -C
Perf event counter list for IBM zEnterprise 196
==============================================================================

Raw
event   Name    Description
------------------------------------------------------------------------------
r0      CPU_CYCLES

                Cycle Count.
                Counter 0 / Basic Counter Set.

r1      INSTRUCTIONS

                Instruction Count.
                Counter 1 / Basic Counter Set.

r2      L1I_DIR_WRITES

                Level-1 I-Cache Directory Write Count.
                Counter 2 / Basic Counter Set.

r3      L1I_PENALTY_CYCLES

                Level-1 I-Cache Penalty Cycle Count.
                Counter 3 / Basic Counter Set.

r4      L1D_DIR_WRITES

                Level-1 D-Cache Directory Write Count.
                Counter 4 / Basic Counter Set.

r5      L1D_PENALTY_CYCLES

                Level-1 D-Cache Penalty Cycle Count.
                Counter 5 / Basic Counter Set.

r20     PROBLEM_STATE_CPU_CYCLES

                Problem-State Cycle Count.
                Counter 32 / Problem-State Counter Set.

r21     PROBLEM_STATE_INSTRUCTIONS

                Problem-State Instruction Count.
                Counter 33 / Problem-State Counter Set.
 ...
```

lscss - List subchannels

Use the **lscss** command to gather subchannel information from sysfs and display it in a summary format.

lscss syntax



Where:

-s or --short
: strips the 0.0. from the device bus-IDs in the command output.

  Note: This option limits the output to bus IDs that begin with 0.0.

-u or --uppercase
: displays the output with uppercase letters. The default is lowercase.

  Changed default: Earlier versions of **lscss** printed the command output in uppercase. Specify this option to obtain the former output style.

--avail
: includes the availability attribute of I/O devices.

--vpm
: shows verified paths in a mask. Channel paths that are listed in this mask are available to Linux device drivers for I/O. Reasons for a channel path to be unavailable include:

  - The corresponding bit is not set in at least one of the PIM, PAM, or POM masks.
  - The channel path is varied offline.
  - Linux received no interrupt to I/O when using this channel path.

--io
: limits the output to I/O subchannels and corresponding devices. This option is the default.

--chsc
: limits the output to CHSC subchannels.

--eadm
: limits the output to EADM subchannels.

-a or --all
: does not limit the output.

-t or --devtype
: limits the output to subchannels that correspond to devices of the specified device types and, if provided, the specified model.

<devicetype>
: specifies a device type.

<model>
: is a specific model of the specified device type.

-d or --devrange
: interprets bus IDs as specifications of devices. By default, bus IDs are interpreted as specifications of subchannels.

<bus_id>
: specifies an individual subchannel; if used with **-d** specifies an individual device. If you omit the leading 0.*<subchannel set ID>*., 0.0. is assumed.

  If you specify subchannels or devices, the command output is limited to these subchannels or devices.

<from_bus_id>-*<to_bus_id>*
: specifies a range of subchannels; if used with **-d** specifies a range of devices. If you omit the leading 0.*<subchannel set ID>*., 0.0. is assumed.

  If you specify subchannels or devices, the command output is limited to these subchannels or devices.

-h or --help
: displays help information for the command. To view the man page, enter **man lscss**.

-v or --version
: displays version information for the command.

Examples

- This command lists all subchannels, including subchannels that do not correspond to I/O devices:

```
# lscss -a
IO Subchannels and Devices:
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.f500 0.0.05cf  1732/01 1731/01 yes  80  80  ff   76000000 00000000
0.0.f501 0.0.05d0  1732/01 1731/01 yes  80  80  ff   76000000 00000000
0.0.f502 0.0.05d1  1732/01 1731/01 yes  80  80  ff   76000000 00000000
0.0.6194 0.0.36e0  3390/0c 3990/e9 yes  fc  fc  ff   32333435 40410000
0.0.6195 0.0.36e1  3390/0c 3990/e9 yes  fc  fc  ff   32333435 40410000
0.0.6196 0.0.36e2  3390/0c 3990/e9 yes  fc  fc  ff   32333435 40410000

CHSC Subchannels:
Device   Subchan.
-----------------
n/a      0.0.ff40

EADM Subchannels:
Device   Subchan.
-----------------
n/a      0.0.ff00
n/a      0.0.ff01
n/a      0.0.ff02
n/a      0.0.ff03
n/a      0.0.ff04
n/a      0.0.ff05
n/a      0.0.ff06
n/a      0.0.ff07
```

- This command limits the output to subchannels with attached DASD model 3390 type 0a:

```
# lscss -t 3390/0a
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.2f08 0.0.0a78  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000
0.0.2fe5 0.0.0b55  3390/0a 3990/e9      c0  c0  bf   34400000 00000000
0.0.2fe6 0.0.0b56  3390/0a 3990/e9      c0  c0  bf   34400000 00000000
0.0.2fe7 0.0.0b57  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000
```

- This command limits the output to the subchannel range 0.0.0b00-0.0.0bff:

```
# lscss 0.0.0b00-0.0.0bff
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.2fe5 0.0.0b55  3390/0a 3990/e9      c0  c0  bf   34400000 00000000
0.0.2fe6 0.0.0b56  3390/0a 3990/e9      c0  c0  bf   34400000 00000000
0.0.2fe7 0.0.0b57  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000
```

- This command limits the output to subchannels 0.0.0a78 and 0.0.0b57 and shows the availability:

```
# lscss --avail 0a78,0b57
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs           Avail.
-----------------------------------------------------------------------------
0.0.2f08 0.0.0a78  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000 good
0.0.2fe7 0.0.0b57  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000 good
```

- This command limits the output to subchannel 0.0.0a78 and prints uppercase output:

```
# lscss -u 0a78
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.2F08 0.0.0A78  3390/0A 3990/E9 YES  C0  C0  FF   34400000 00000000
```

- This command limits the output to subchannels that correspond to I/O device 0.0.7e10 and the device range 0.0.2f00-0.0.2fff:

```
# lscss -d 2f00-2fff,0.0.7e10
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.2f08 0.0.0a78  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000
0.0.2fe5 0.0.0b55  3390/0a 3990/e9      c0  c0  bf   34400000 00000000
0.0.2fe6 0.0.0b56  3390/0a 3990/e9      c0  c0  bf   34400000 00000000
0.0.2fe7 0.0.0b57  3390/0a 3990/e9 yes  c0  c0  ff   34400000 00000000
0.0.7e10 0.0.1828  3390/0c 3990/e9 yes  f0  f0  ef   34403541 00000000
```

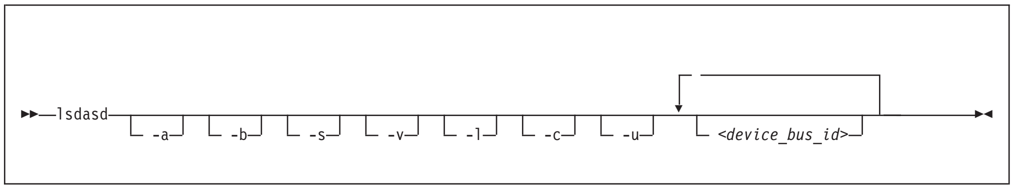
- This example shows a CHPID with PIM, PAM, and POM masks that are OK. However, the entry in the **vpm** column indicates that one of the paths, 0x41, is not usable for I/O.

```
# lscss --vpm
Device    Subchan. DevType  CU Type Use  PIM PAM POM VPM CHPIDs
-------------------------------------------------------------------------
0.0.f500  0.0.05cf  1732/01 1731/01 yes  80  80  ff  80  76000000 00000000
0.0.f501  0.0.05d0  1732/01 1731/01 yes  80  80  ff  80  76000000 00000000
0.0.f502  0.0.05d1  1732/01 1731/01 yes  80  80  ff  80  76000000 00000000
0.0.6194  0.0.3700  3390/0c 3990/e9 yes  fc  fc  ff  f8  32333435 40410000
0.0.6195  0.0.3701  3390/0c 3990/e9 yes  fc  fc  ff  f8  32333435 40410000
0.0.6196  0.0.3702  3390/0c 3990/e9 yes  fc  fc  ff  f8  32333435 40410000
0.0.6197  0.0.3703  3390/0c 3990/e9      fc  fc  ff  00  32333435 40410000
0.2.5600  0.2.0040  1732/03 1731/03      80  80  ff  00  5d000000 00000000
```

lsdasd - List DASD devices

Use the **lsdasd** command to gather information about DASD devices from sysfs and display it in a summary format.

lsdasd syntax



Where:

-a or --offline
includes devices that are currently offline.

-b or --base
omits PAV alias devices. Lists only base devices.

-s or --short
strips the "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.

<device_bus_id>
limits the output to information about the specified devices only.

--version
displays the version of the command.

-h or --help
displays out a short help text, then exits. To view the man page, enter **man lsdasd**.

Examples

- The following command lists all DASD (including offline DASDS):

```
# lsdasd -a
Bus-ID      Status     Name     Device     Type    BlkSz     Size       Blocks
==============================================================================
0.0.0190    offline
0.0.0191    offline
0.0.019d    offline
0.0.019e    offline
0.0.0592    offline
0.0.4711    offline
0.0.4712    offline
0.0.4f2c    offline
0.0.4d80    active    dasda     94:0       ECKD    4096     4695MB       1202040
0.0.4f19    active    dasdb     94:4       ECKD    4096    23034MB       5896800
0.0.4d81    active    dasdc     94:8       ECKD    4096     4695MB       1202040
0.0.4d82    active    dasdd     94:12      ECKD    4096     4695MB       1202040
0.0.4d83    active    dasde     94:16      ECKD    4096     4695MB       1202040
```

- The following command shows information only for the DASD with device number 0x4d80 and strips the “0.n.” from the bus IDs in the output:

```
# lsdasd -s 4d80
Bus-ID      Status    Name       Device    Type     BlkSz    Size       Blocks
==============================================================================
4d80        active    dasda      94:0      ECKD     4096     4695MB     1202040
```

- The following command shows only online DASDs in the format of **lsdasd** versions earlier than 1.7.0:

```
# lsdasd -c
0.0.4d80(ECKD) at ( 94: 0) is dasda : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4f19(ECKD) at ( 94: 4) is dasdb : active at blocksize 4096, 5896800 blocks, 23034 MB
0.0.4d81(ECKD) at ( 94: 8) is dasdc : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4d82(ECKD) at ( 94: 12) is dasdd : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4d83(ECKD) at ( 94: 16) is dasde : active at blocksize 4096, 1202040 blocks, 4695 MB
```

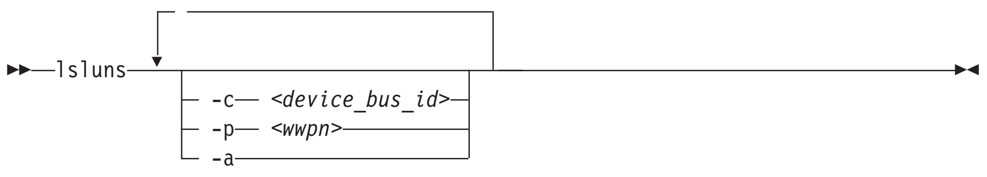
- The following command shows the device geometry of and some settings for the DASD with device bus-ID `0.0.4d82`:

```
lsdasd -l 0.0.4d82
0.0.4d82/dasdc/94:12
  status:       active
  type:         ECKD
  blksz:        4096
  size:         4695MB
  blocks:       1202040
  use_diag:     0
  readonly:     0
  eer_enabled:  0
  erplog:       0
  uid:          IBM.75000000010671.4d82.16
```

lsluns - Discover LUNs in Fibre Channel SANs

Use the **lsluns** command to discover and scan LUNs in Fibre Channel storage area networks (SANs) or to show LUNs actively used in Linux.

lsluns syntax



Where:

-c or --ccw *<device_bus_id>*
shows LUNs for a specific FCP device.

-p or --port *<wwpn>*
shows LUNs for the port with the specified WWPN.

-a or --active
shows the currently active LUNs. A bracketed `x` indicates that the corresponding disk is encrypted.

-v or --version
displays the version number of **lsluns** and exits.

-h or --help
displays an overview of the syntax. To view the man page, enter **man lsluns**.

Examples

- This example shows all LUNs for port 0x500507630300c562:

```
# lsluns --port 0x500507630300c562
Scanning for LUNs on adapter 0.0.5922
        at port 0x500507630300c562:
                0x4010400000000000
                0x4010400100000000
                0x4010400200000000
                0x4010400300000000
                0x4010400400000000
                0x4010400500000000
```

- This example shows all LUNs for an FCP device with bus ID 0.0.5922:

```
# lsluns -c 0.0.5922
        at port 0x500507630300c562:
                0x4010400000000000
                0x4010400100000000
                0x4010400200000000
                0x4010400300000000
                0x4010400400000000
                0x4010400500000000
        at port 0x500507630303c562:
                0x4010400000000000
                0x4010400100000000
                0x4010400200000000
                0x4010400300000000
                0x4010400400000000
                0x4010400500000000
```

- This example shows all active LUNs:

```
# lsluns -a
adapter = 0.0.5922
     port = 0x500507630300c562
          lun = 0x401040a200000000    /dev/sg0    Disk IBM:2107900
          lun = 0x401040a300000000(x) /dev/sg1    Disk IBM:2107900
          lun = 0x401040a400000000    /dev/sg2    Disk IBM:2107900
          lun = 0x401040a500000000    /dev/sg3    Disk IBM:2107900
     port = 0x500507630303c562
         lun = 0x401040a400000000     /dev/sg4    Disk IBM:2107900
          lun = 0x401040a500000000    /dev/sg5    Disk IBM:2107900
adapter = 0.0.593a
     port = 0x500507630307c562
          lun = 0x401040b000000000    /dev/sg6    Disk IBM:2107900
          lun = 0x401040b300000000    /dev/sg7    Disk IBM:2107900
          ...
```

The (x) in the output indicates that the device is encrypted.

lsmem - Show online status information about memory blocks

Use the **lsmem** command to list the ranges of available memory with their online status.

The listed memory blocks correspond to the memory block representation in sysfs. The command also shows the memory block size, the device size, and the amount of memory in online and offline state.

lsmem syntax

```
►►──lsmem──┬────┬──────────────────────────────────────────────────►◄
           └─-a─┘
```

Where:

-a or --all
lists each individual memory block, instead of combining memory blocks with similar attributes.

-v or --version
displays the version number of **lsmem**, then exits.

-h or --help
displays a short help text, then exits. To view the man page, enter **man lsmem**.

The columns in the command output have this meaning:

Address range
Start and end address of the memory range.

Size
Size of the memory range in MB (1024 x 1024 bytes).

State
Indication of the online status of the memory range. State `on->off` means that the address range is in transition from online to offline.

Removable
`yes` if the memory range can be set offline, `no` if it cannot be set offline. A dash (-) means that the range is already offline. The kernel method that identifies removable memory ranges is heuristic and not exact. Occasionally, memory ranges are falsely reported as removable or falsely reported as not removable.

Device
Device number or numbers that correspond to the memory range.

A device represents a unit of memory for the hypervisor in control of the memory. The hypervisor cannot reuse a device unless the entire corresponding memory range is offline.

The memory units that you can set online or offline from Linux are memory blocks. In most memory configurations, there is a one-to-one mapping of devices and memory blocks or a mapping of multiple devices to a single memory block. In other configurations, multiple memory blocks might map to a single device. Memory might be used inefficiently if a device includes both online and offline memory blocks.

The **chmem** command with the size parameter automatically chooses the best suited device or devices for setting memory online or offline. The device size depends on the hypervisor and on the amount of total online and offline memory.

Examples

- The output of this command, shows ranges of adjacent memory blocks with similar attributes.

```
# lsmem
Address range                          Size (MB)  State    Removable  Device
===============================================================================
0x0000000000000000-0x000000000fffffff        256  online   no         0
0x0000000010000000-0x000000002fffffff        512  online   yes        1-2
0x0000000030000000-0x000000003fffffff        256  online   no         3
0x0000000040000000-0x000000006fffffff        768  online   yes        4-6
0x0000000070000000-0x00000000ffffffff       2304  offline  -          7-15

Memory device size  : 256 MB
Memory block size   : 256 MB
Total online memory : 1792 MB
Total offline memory: 2304 MB
```

- The output of this command, shows each memory block as a separate range.

```
# lsmem -a
Address range                          Size (MB)  State    Removable  Device
===============================================================================
0x0000000000000000-0x000000000fffffff        256  online   no         0
0x0000000010000000-0x000000001fffffff        256  online   yes        1
0x0000000020000000-0x000000002fffffff        256  online   yes        2
0x0000000030000000-0x000000003fffffff        256  online   no         3
0x0000000040000000-0x000000004fffffff        256  online   yes        4
0x0000000050000000-0x000000005fffffff        256  online   yes        5
0x0000000060000000-0x000000006fffffff        256  online   yes        6
0x0000000070000000-0x000000007fffffff        256  offline  -          7
0x0000000080000000-0x000000008fffffff        256  offline  -          8
0x0000000090000000-0x000000009fffffff        256  offline  -          9
0x00000000a0000000-0x00000000afffffff        256  offline  -          10
0x00000000b0000000-0x00000000bfffffff        256  offline  -          11
0x00000000c0000000-0x00000000cfffffff        256  offline  -          12
0x00000000d0000000-0x00000000dfffffff        256  offline  -          13
0x00000000e0000000-0x00000000efffffff        256  offline  -          14
0x00000000f0000000-0x00000000ffffffff        256  offline  -          15

Memory device size  : 256 MB
Memory block size   : 256 MB
Total online memory : 1792 MB
Total offline memory: 2304 MB
```

lsqeth - List qeth-based network devices

Use the **lsqeth** command to display a summary of information about qeth-based network devices.

Before you begin: To be able to use this command, you must also install **qethconf** (see “qethconf - Configure qeth devices” on page 658). You install both **qethconf** and **lsqeth** with the s390-tools RPM.

lsqeth syntax

```
►►──lsqeth──┬────┬──┬─────────────┬──────────────────────────────►◄
            └ -p ┘  └ <interface> ┘
```

Where:

-p or --proc
: displays the interface information in the former /proc/qeth format. This option can generate input to tools that expect this particular format.

<interface>
: limits the output to information about the specified interface only.

-v or --version
: displays version information for the command.

-h or --help
: displays a short help text, then exits. To view the man page, enter **man lsqeth**.

Examples

- The following command lists information about 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
        state                   : UP (LAN ONLINE)
        priority_queueing       : always queue 2
        fake_broadcast          : 0
        buffer_count            : 64
        layer2                  : 0
        isolation               : none
        sniffer                 : 0
```

- The following command lists information about all qeth-based interfaces in the former /proc/qeth format:

```
# lsqeth -p
devices                    CHPID interface  cardtype       port chksum prio-q'ing rtr4 rtr6 lay'2 cnt
-------------------------- ----- ---------- -------------- ---- ------ ---------- ---- ---- ----- -----
0.0.833f/0.0.8340/0.0.8341 xFE   hsi0       HiperSockets   0    sw     always_q_2 no   no   0     128
0.0.f5a2/0.0.f5a3/0.0.f5a4 xB5   eth0       OSD_1000       0    sw     always_q_2 no   no   1     64
0.0.fba2/0.0.fba3/0.0.fba4 xB0   eth1       OSD_1000       0    sw     always_q_2 no   no   0     64
```

lsreipl - List IPL and re-IPL settings

Use the **lsreipl** command to find out which boot device and which options are used if you issue the reboot command.

You can also display information about the current boot device.

lsreipl syntax

```
►►──lsreipl──┬────┬──────────────────────────────────────────────────►◄
             └─-i─┘
```

where:

-i or --ipl
displays the IPL setting.

-v or --version
displays the version number of **lsreipl** and exits.

-h or --help
displays an overview of the syntax. Any other parameters are ignored. To view the man page, enter **man lsreipl**.

By default the re-IPL device is set to the current IPL device. Use the chreipl command to change the re-IPL settings.

Examples

- This example shows the current re-IPL settings:

```
# lsreipl
Re-IPL type:     fcp
WWPN:            0x500507630300c562
LUN:             0x401040b300000000
Device:          0.0.1700
bootprog:        0
br_lba:          0
Bootparms:       ""
```

lsscm - List storage-class memory increments

Use the **lsscm** command to list status and other information about available storage-class memory increments.

lsscm syntax

```
►►──lsscm──┬──────┬──────────────────────────────────────────────►◄
           ├─ -h──┤
           └─ -v──┘
```

Where:

-h or --help
displays help information for the command. To view the man page, enter **man lsscm**.

-v or --version
displays version information for the command.

In the output table, the columns have the following meaning:

SCM Increment
Starting address of the storage-class memory increment.

Size
Size of the block device that represents the storage-class memory increment.

Name
Name of the block device that represents the storage-class memory increment.

Rank
A quality ranking in the form of a number in the range 1 - 15 where a lower number means better ranking.

D_state
Data state of the storage-class memory increment. A number that indicates whether there is data on the increment. The data state can be:

1	The increment contains zeros only.
2	Data was written to the increment.
3	No data was written to the increment since the increment was attached.

O_state
Operation state of the storage-class memory increment.

Pers
Persistence attribute.

ResID
Resource identifier.

Examples

- This command lists all increments:

```
    # lsscm
 SCM Increment    Size    Name  Rank D_state O_state Pers ResID
--------------------------------------------------------------
0000000000000000 16384MB scma     1       2       1    2     1
0000000400000000 16384MB scmb     1       2       1    2     1
```

lsshut - List the current system shutdown actions

Use the **lsshut** command to see how the Linux instance is configured for the `halt`, `poff`, `reboot`, `restart`, and `panic` system shutdown triggers.

For more information about the shutdown triggers and possible shutdown actions, see Chapter 8, “Shutdown actions,” on page 117.

If the action is kdump, a second action might be listed. This second action is the backup action that is taken if kdump fails. See *Using the Dump Tools*, SC33-8412 for details about using kdump.

lsshut syntax

```
►►──lsshut──┬──────┬─────────────────────────────────────────────►◄
            ├─ -h──┤
            └─ -v──┘
```

where:

-h or --help
displays a short help text, then exits. To view the man page, enter **man lsshut**.

-v or --version
displays the version number of **lsshut** and exits.

Examples

- To query the configuration issue:

```
# lsshut
Trigger    Action
========================
Halt       stop
Power off  vmcmd (LOGOFF)
Reboot     reipl
Restart    kdump,dump_reipl
Panic      kdump,dump_reipl
```

lstape - List tape devices

Use the **lstape** command to gather information about tape devices and display it in a summary format.

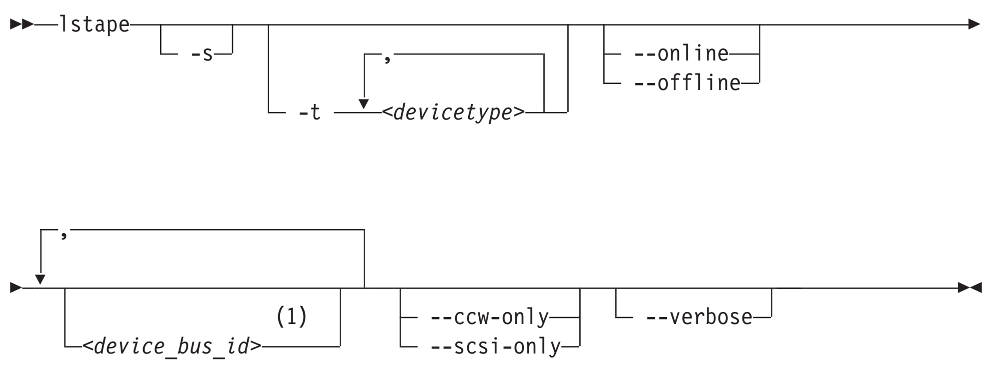
It gathers information about CCW-attached tape devices and tape devices that are attached to the SCSI bus from sysfs (see "Displaying tape information" on page 236).

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.

lstape syntax



Notes:

1 specify the first device bus-ID with a leading blank.

Where:

-s or --shortid
strips the "0.<*n*>." from the device bus-IDs in the command output. For CCW-attached devices only.

-t or --type
limits the output to information about the specified type or types of CCW-attached devices only.

--ccw-only
limits the output to information about CCW-attached devices only.

--scsi-only
limits the output to information about tape devices that are attached to the SCSI bus.

--online | --offline
limits the output to information about online or offline CCW-attached tape devices only.

<device_bus_id>
limits the output to information about the specified tape device or devices only.

-V or --verbose
For tape devices attached to the SCSI bus only. Displays the serial of the tape and information about the FCP connection as an additional text line that follows each SCSI tape in the list.

-h or --help
displays a short help text. To view the man page, enter **man lstape**.

-v or --version
displays the version of the command.

Examples

- This command displays information about all tapes found, here one CCW-attached tape and one tape and changer device that is configured for zFCP:

```
#> lstape
FICON/ESCON tapes (found 1):
TapeNo  BusID      CuType/Model  DevType/Model  BlkSize  State   Op   MedState
0       0.0.0480   3480/01       3480/04        auto     UNUSED  ---  UNLOADED

SCSI tape devices (found 2):
Generic  Device       Target        Vendor     Model         Type     State
sg4      IBMchanger0  0:0:0:0       IBM        03590H11      changer  running
sg5      IBMtape0     0:0:0:1       IBM        03590H11      tapedrv  running
```

If only the generic tape driver (st) and the generic changer driver (ch) are loaded, the output lists those names in the device section:

```
#> lstape
FICON/ESCON tapes (found 1):
TapeNo  BusID      CuType/Model  DevType/Model  BlkSize  State   Op    MedState
0       0.0.0480   3480/01       3480/04        auto     UNUSED  ---   UNLOADED

SCSI tape devices (found 2):
Generic Device       Target         Vendor     Model         Type     State
sg0     sch0         0:0:0:0        IBM        03590H11      changer  running
sg1     st0          0:0:0:1        IBM        03590H11      tapedrv  running
```

- This command displays information about all available CCW-attached tapes.

```
# lstape —-ccw-only
TapeNo  BusID      CuType/Model DevType/DevMod  BlkSize State   Op      MedState
0       0.0.0132   3590/50      3590/11         auto    IN_USE  ---     LOADED
1       0.0.0110   3490/10      3490/40         auto    UNUSED  ---     UNLOADED
2       0.0.0133   3590/50      3590/11         auto    IN_USE  ---     LOADED
3       0.0.012a   3480/01      3480/04         auto    UNUSED  ---     UNLOADED
N/A     0.0.01f8   3480/01      3480/04         N/A     OFFLINE ---     N/A
```

- This command limits the output to tapes of type 3480 and 3490.

```
# lstape -t 3480,3490
TapeNo  BusID      CuType/Model DevType/DevMod  BlkSize State   Op      MedState
1       0.0.0110   3490/10      3490/40         auto    UNUSED  ---     UNLOADED
3       0.0.012a   3480/01      3480/04         auto    UNUSED  ---     UNLOADED
N/A     0.0.01f8   3480/01      3480/04         N/A     OFFLINE ---     N/A
```

- This command limits the output to those tapes of type 3480 and 3490 that are currently online.

```
# lstape -t 3480,3490 --online
TapeNo  BusID      CuType/Model DevType/DevMod  BlkSize State   Op      MedState
1       0.0.0110   3490/10      3490/40         auto    UNUSED  ---     UNLOADED
3       0.0.012a   3480/01      3480/04         auto    UNUSED  ---     UNLOADED
```

- This command limits the output to the tape with device bus-ID 0.0.012a and strips the "0.<*n*>." from the device bus-ID in the output.

```
# lstape -s 0.0.012a
TapeNo  BusID      CuType/Model DevType/DevMod  BlkSize State   Op      MedState
3       012a       3480/01      3480/04         auto    UNUSED  ---     UNLOADED
```

- This command limits the output to SCSI devices but gives more details. The serial numbers are only displayed if the **sg_inq** command is found on the system.

```
#> lstape --scsi-only --verbose
Generic Device       Target        Vendor     Model        Type       State
        HBA          WWPN                     Serial
sg0     st0          0:0:0:1       IBM        03590H11     tapedrv    running
        0.0.1708     0x500507630040727b       NO/INQ
sg1     sch0         0:0:0:2       IBM        03590H11     changer    running
        0.0.1708     0x500507630040727b       NO/INQ
```

Data fields for SCSI tape devices

There are specific data fields for SCSI tape devices.

Table 63. lstape data fields for SCSI tape devices

Attribute	Description
Generic	SCSI generic device file for the tape drive (for example `/dev/sg0`). This attribute is empty if the **sg_inq** command is not available.
Device	Main device file for accessing the tape drive, for example: • `/dev/st0` for a tape drive that is attached through the Linux st device driver • `/dev/sch0` for a medium changer device that is attached through the Linux changer device driver • `/dev/IBMchanger0` for a medium changer that is attached through the IBMtape or lin_tape device driver • `/dev/IBMtape0` for a tape drive that is attached through the IBMtape or lin_tape device driver
Target	The ID in Linux used to identify the SCSI device.
Vendor	The vendor field from the tape drive.
Model	The model field from the tape drive.
Type	"Tapedrv" for a tape driver or "changer" for a medium changer.
State	The state of the SCSI device in Linux. This state is an internal state of the Linux kernel, any state other than "running" can indicate problems.

Table 63. lstape data fields for SCSI tape devices (continued)

Attribute	Description
HBA	The FCP device to which the tape drive is attached.
WWPN	The WWPN (worldwide port name) of the tape drive in the SAN.
Serial	The serial number field from the tape drive.

lszcrypt - Display zcrypt devices

Use the **lszcrypt** command to display information about cryptographic adapters that are managed by zcrypt and its AP bus attributes.

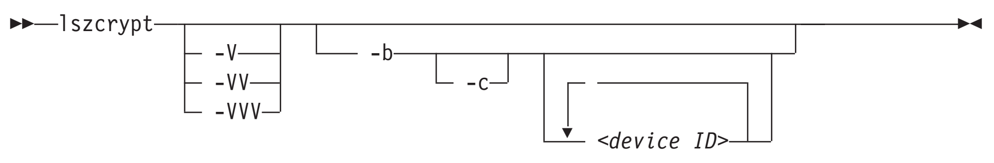
To set the attributes, use "chzcrypt - Modify the zcrypt configuration" on page 556. The following information can be displayed for each cryptographic adapter:

- The card type
- The online status
- The hardware card type
- The card capability
- 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

lszcrypt syntax



Where:

-V or --verbose, -VV, -VVV
increases the verbose level for cryptographic adapter information.

-V or --verbose
displays card type and online status.

-VV displays card type, online status, hardware card type, hardware queue depth, and request count.

-VVV displays card type, online status, hardware card type, hardware queue depth, request count, pending request queue count, outstanding request queue count, and installed function facilities.

<device ID>
specifies the cryptographic adapter that is displayed. A cryptographic adapter can be specified either in decimal notation or hexadecimal notation with a '0x' prefix. If no adapters are specified, information about all available adapters is displayed.

-b or --bus
displays the AP bus attributes.

-c or --capability
shows the capabilities of a cryptographic adapter of hardware type 6 or higher.

The capabilities of a cryptographic adapter depend on the card type and the installed function facilities. A cryptographic adapter can provide one or more of the following capabilities:

- RSA 2K Clear Key
- RSA 4K Clear Key
- CCA Secure Key
- EP11 Secure Key
- Long RNG

`-h or --help`
displays help information for the command. To view the man page, enter **`man lszcrypt`**.

`-v or --version`
displays version information.

Examples

These examples illustrate common uses for **`lszcrypt`**.

- To display information about all available cryptographic adapters:

```
# lszcrypt
```

This command displays output similar to the following example:

```
card00: CEX3A
card01: CEX3C
card02: CEX3A
card03: CEX3C
card04: CEX3C
card05: CEX3C
card06: CEX4A
card08: CEX4A
card09: CEX4P
card0a: CEX4P
card0b: CEX4C
```

- To display card type and online status of all available cryptographic adapters:

```
# lszcrypt -V
```

This command displays output similar to the following example:

```
card00: CEX3A online
card01: CEX3C online
card02: CEX3A offline
card03: CEX3C online
card04: CEX3C online
card05: CEX3C online
card06: CEX4A offline
card08: CEX4A online
card09: CEX4P online
card0a: CEX4P online
card0b: CEX4C online
```

- To display card type, online status, hardware card type, hardware queue depth, and request count for cryptographic adapters 00, 02, and 0a.:

```
# lszcrypt -VV 0x00 0x02 0x0b
```

This command displays output similar to the following example:

```
card00: CEX3A online  hwtype=8  depth=8 request_count=0
card02: CEX3A offline hwtype=8  depth=8 request_count=0
card0b: CEX4C online  hwtype=10 depth=8 request_count=292
```

Tip: In the adapter specification you can also use one-digit hexadecimal or decimal notation. The specifications `0x0 0x2 0xb`, `0x00 0x02 0x0b` and `0 2 11` are all equivalent.

- To display the device ID and the installed function facility in hexadecimal notation as well as card type, online status, hardware card type, hardware queue depth, request count, pending request queue count, outstanding request queue count, and installed function facilities:

```
# lszcrypt -VVV 0x00 0x02 0x0b
```

This command displays output similar to the following example:

```
card00: CEX3A online  hwtype=8  depth=8 request_count=0   pendingq_count=0 requestq_count=0 functions=0x60000000
card02: CEX3A offline hwtype=8  depth=8 request_count=0   pendingq_count=0 requestq_count=0 functions=0x60000000
card0b: CEX4C online  hwtype=10 depth=8 request_count=292 pendingq_count=0 requestq_count=0 functions=0x90000000
```

- To display AP bus information:

```
# lszcrypt -b
```

This command displays output similar to the following example:

```
ap_domain=8
ap_interrupts are enabled
config_time=30 (seconds)
poll_thread is disabled
poll_timeout=250000 (nanoseconds)
```

- To display the capabilities for the cryptographic adapter with device index 0x0b:

```
# lszcrypt -c 0x0b
```

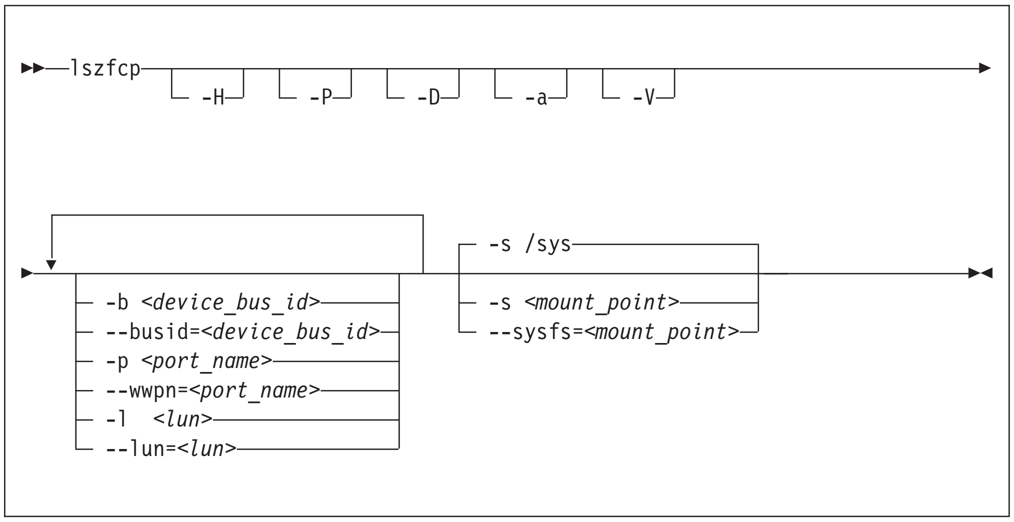
This command displays output similar to the following example:

```
Coprocessor card0b provides capability for:
CCA Secure Key
RSA 4K Clear Key
Long RNG
```

lszfcp - List zfcp devices

Use the **lszfcp** command to gather information about zfcp devices, ports, units, and their associated class devices from sysfs and to display it in a summary format.

lszfcp syntax



Where:

-H or --hosts
shows information about hosts.

-P or --ports
shows information about ports.

-D or --devices
shows information about SCSI devices.

-a or --attributes
shows all attributes (implies **-V**).

-V or --verbose
shows sysfs paths of associated class and bus devices.

-b or --busid *<device_bus_id>*
limits the output to information about the specified device.

-p or --wwpn *<port_name>*
limits the output to information about the specified port name.

-l or --lun *<lun>*
limits the output to information about the specified LUN.

-s or --sysfs *<mount_point>*
specifies the mount point for sysfs.

-v or --version
displays version information.

-h or --help
displays a short help text. To view the man page, enter **man lszfcp**.

Examples

- This command displays information about all available hosts, ports, and SCSI devices.

```
# lszfcp -H -D -P
0.0.3d0c host0
0.0.500c host1
...
0.0.3c0c host5
0.0.3d0c/0x500507630300c562 rport-0:0-0
0.0.3d0c/0x50050763030bc562 rport-0:0-1
0.0.3d0c/0x500507630303c562 rport-0:0-2
0.0.500c/0x50050763030bc562 rport-1:0-0
...
0.0.3c0c/0x500507630303c562 rport-5:0-2
0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:0
0.0.3d0c/0x500507630300c562/0x4010403300000000 0:0:0:1
0.0.3d0c/0x50050763030bc562/0x4010403200000000 0:0:1:0
0.0.3d0c/0x500507630303c562/0x4010403200000000 0:0:2:0
0.0.500c/0x50050763030bc562/0x4010403200000000 1:0:0:0
...
0.0.3c0c/0x500507630303c562/0x4010403200000000 5:0:2:0
```

- This command shows SCSI devices and limits the output to the devices that are attached through the FCP device with bus ID 0.0.3d0c:

```
# lszfcp -D -b 0.0.3d0c
0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:0
0.0.3d0c/0x500507630300c562/0x4010403300000000 0:0:0:1
0.0.3d0c/0x50050763030bc562/0x4010403200000000 0:0:1:0
0.0.3d0c/0x500507630303c562/0x4010403200000000 0:0:2:0
```

mon_fsstatd – Monitor z/VM guest file system size

The **`mon_fsstatd`** command is a user space daemon that collects physical file system size data from Linux on z/VM.

The daemon periodically writes the data as defined records to the z/VM monitor stream using the monwriter character device driver.

You can start the daemon with 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 begin:

- Install the monwriter device driver and set up z/VM to start the collection of monitor sample data. See Chapter 33, “Writing z/VM monitor records,” on page 427 for information about 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 following publications provide general information about DCSSs, DIAG x'DC', CP commands, and APPLDATA:

- See *z/VM Saved Segments Planning and Administration*, SC24-6229 for general information about DCSSs.
- See *z/VM CP Programming Services*, SC24-6179 for information about the DIAG x'DC' instruction.
- See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about the CP commands.
- See *z/VM Performance*, SC24-6208 for information about monitor APPLDATA.

You can run the **`mon_fsstatd`** command in two ways.

- Calling mon_statd with the service utility, if this utility is provided by your distribution. Use this method when the mon_statd service script is installed in `/etc/init.d`. This method reads the configuration file `/etc/sysconfig/mon_statd`. The mon_statd service script also controls other daemons, such as mon_procd.
- Calling mon_fsstatd manually from a command line.

mon_statd service utility syntax

If you run the **`mon_fsstatd`** daemon through the service utility, you configure the daemon through specifications in a configuration file.

```
►►──┬─service mon_statd──────────┬──┬─ start───┬──────────────────────────►◄
    │                     (1)    │  ├─ stop────┤
    └─/etc/init_d/mon_statd──────┘  ├─ status──┤
                                    └─ restart─┘
```

Notes:

1 If your distribution does not provide the service utility, the mon_statd init script can also be called directly by using `/etc/init.d/mon_statd`

Where:

`start`
: enables monitoring of guest file system size, by using the configuration in `/etc/sysconfig/mon_statd`.

`stop`
: disables monitoring of guest file system size.

`status`
: shows 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 wanted 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" is interpreted as "no".

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 can look different and include messages for other daemons, such as mon_procd:

- To enable guest file system size monitoring:

```
> service mon_statd start
...
Starting mon_fsstatd:[ OK ]
...
```

- To display the status:

```
> service mon_statd status
...
mon_fsstatd (pid 1075, interval: 30) is running.
...
```

- To disable guest file system size monitoring:

```
> service mon_statd stop
...
Stopping mon_fsstatd:[ OK ]
...
```

- To display the status again and check that monitoring is now disabled:

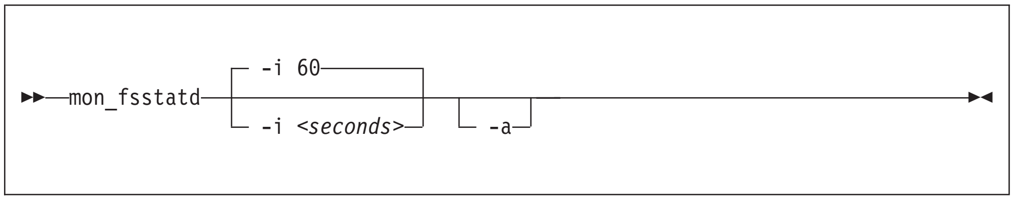
```
> service mon_statd status
...
mon_fsstatd is not running
...
```

- To restart the daemon and re-read the configuration file:

```
> service mon_statd restart
...
stopping mon_fsstatd:[ OK ]
starting mon_fsstatd:[ OK ]
...
```

mon_fsstatd command-line syntax

If you call the **mon_fsstatd** daemon from the command line, you configure the daemon through command parameters.



Where:

-i or --interval *<seconds>*
specifies the wanted sampling interval in seconds.

-a or --attach
runs the daemon in the foreground.

-h or --help
displays help information for the command. To view the man page, enter **man mon_fsstatd**.

-v or --version
displays version information for the command.

Examples of command-line use

- To start mon_fsstatd with default setting:

```
> mon_fsstatd
```

- To start mon_fsstatd with a sampling interval of 30 seconds:

```
> mon_fsstatd -i 30
```

- To start mon_fsstatd and have it run in the foreground:

```
> mon_fsstatd -a
```

- To start mon_fsstatd with a sampling interval of 45 seconds and have it run in the foreground:

```
> mon_fsstatd -a -i 45
```

Processing monitor data

The mon_fsstatd daemon writes physical file system size data for Linux on z/VM to the z/VM monitor stream.

The following is the format of the file system size data that is passed to the z/VM monitor stream. One sample monitor record is written for each physical file system that is mounted at the time of the sample interval. The monitor data in each record contains a header that consists of a time stamp, the length of the data, and an offset. The header is followed by the file system data (as obtained from statvfs). The file system data fields begin with “fs_”.

Table 64. File system size data format

Type	Name	Description
__u64	time_stamp	Time at which the file system data was sampled.
__u16	data_len	Length of data that follows the header.
__u16	data_offset	Offset from start of the header to the start of the file system data (that is, to the fields that begin with fs_).
__u16	fs_name_len	Length of the file system name. The file system name can be too long to fit in the monitor record. If so, this length is the portion of the name that is contained in the monitor record.
char [fs_name_len]	fs_name	The file system name. If the name is too long to fit in the monitor record, the name is truncated to the length in the fs_name_len field.
__u16	fs_dir_len	Length of the mount directory name. The mount directory name can be too long to fit in the monitor record. If so, this length is the portion of the name that is contained in the monitor record.
char[fs_dir_len]	fs_dir	The mount directory name. If the name is too long to fit in the monitor record, the name is truncated to the length in the fs_dir_len field.
__u16	fs_type_len	Length of the mount type. The mount type can be too long to fit in the monitor record. If so, this length is the portion that is contained in the monitor record.
char[fs_type_len]	fs_type	The mount type (as returned by getmntent). If the type is too long to fit in the monitor record, the type is truncated to the length in the fs_type_len field.

Table 64. File system size data format (continued)

Type	Name	Description
__u64	fs_bsize	File system block size.
__u64	fs_frsize	Fragment size.
__u64	fs_blocks	Total data blocks in file system.
__u64	fs_bfree	Free blocks in fs.
__u64	fs_bavail	Free blocks avail to non-superuser.
__u64	fs_files	Total file nodes in file system.
__u64	fs_ffree	Free file nodes in fs.
__u64	fs_favail	Free file nodes available to non-superuser.
__u64	fs_flag	Mount flags.

Use the time_stamp to correlate all file systems that were sampled in a given interval.

Reading the monitor data

All records that are written to the z/VM monitor stream begin with a product identifier.

The product ID is a 16-byte structure of the form pppppppffnvvrrmm, where for records that are written by mon_fsstatd, these values are:

ppppppp
is a fixed ASCII string LNXAPPL.

ff is the application number for mon_fsstatd = x'0001'.

n is the record number = x'00'.

vv is the version number = x'0000'.

rr is reserved for future use and should be ignored.

mm is reserved for mon_fsstatd and should be ignored.

Note: Though the mod_level field (mm) of the product ID varies, there is no relationship between any particular mod_level and file system. The mod_level field should be ignored by the reader of this monitor data.

There are many tools available to read z/VM monitor data. One such tool is the Linux monreader character device driver. For more information about monreader, see Chapter 34, “Reading z/VM monitor records,” on page 431.

mon_procd – Monitor Linux on z/VM

The **mon_procd** command is a user space daemon that gathers system summary information and information about up to 100 concurrent processes on Linux on z/VM.

The daemon writes this data to the z/VM monitor stream by 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 begin:

- Install the monwriter device driver and set up z/VM to start the collection of monitor sample data. See Chapter 33, “Writing z/VM monitor records,” on page 427 for information about 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 Linux instance on which the proc_mond deamon runs requires a z/VM guest virtual machine with the OPTION APPLMON statement in the CP directory entry.

The following publications provide general information about DCSSs, CP commands, and APPLDATA:

- See *z/VM Saved Segments Planning and Administration*, SC24-6229 for general information about DCSSs.
- See *z/VM CP Commands and Utilities Reference*, SC24-6175 for information about the CP commands.
- See *z/VM Performance*, SC24-6208 for information about monitor APPLDATA.

You can run the **mon_procd** command in two ways:

- Calling **mon_procd** with the service utility. Use this method when the mon_statd service script is installed in `/etc/init.d`. This method reads 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.

mon_statd service utility syntax

If you run the **mon_procd** daemon through the service utility, you configure the daemon through specifications in a configuration file.

```
►►──┬─service mon_statd────────────┬──┬─ start───┬──────────────────────────►◄
    │                      (1)     │  ├─ stop────┤
    └─/etc/init_d/mon_statd────────┘  ├─ status──┤
                                      └─ restart─┘
```

Notes:

1 If your distribution does not provide the service utility, the mon_statd init script can also be called directly by using `/etc/init.d/mon_statd`

Where:

`start`
enables monitoring of guest process data, by using the configuration in `/etc/sysconfig/mon_statd`.

`stop`
disables monitoring of guest process data.

`status`
shows current status of guest process data monitoring.

`restart`
stops and restarts guest process data monitoring. Useful to re-read the configuration file when it was changed.

Configuration file keywords

PROC_INTERVAL="<*n*>"
specifies the wanted sampling interval in seconds.

PROC="yes | no"
specifies whether to enable the mon_procd daemon. Set to "yes" to enable the daemon. Anything other than "yes" is interpreted as "no".

Examples of service utility use

- This example sets the sampling interval to 30 seconds and enables the mon_procd:

```
PROC_INTERVAL="30"
PROC="yes"
```

Example of mon_statd use (note that your output might look different and include messages for other daemons, such as mon_fsstatd):

- To enable guest process data monitoring:

```
> service mon_statd start
...
Starting mon_procd:[ OK ]
...
```

- To display the status:

```
> service mon_statd status
...
mon_procd (pid 1075, interval: 30) is running.
...
```

- To disable guest process data monitoring:

```
> service mon_statd stop
...
Stopping mon_procd:[ OK ]
...
```

- To display the status again and check that monitoring is now disabled:

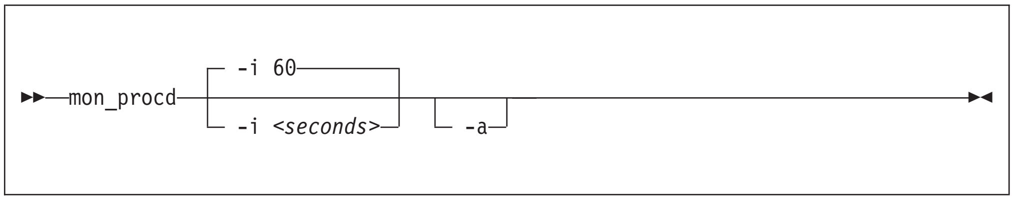
```
> service mon_statd status
...
mon_procd is not running
...
```

- To restart the daemon and re-read the configuration file:

```
> service mon_statd restart
...
stopping mon_procd:[ OK ]
starting mon_procd:[ OK ]
...
```

mon_procd command-line syntax

If you call the **mon_procd** daemon from the command line, you configure the daemon through command parameters.



Where:

-i or --interval *<seconds>*
specifies the wanted sampling interval in seconds.

-a or --attach
runs the daemon in the foreground.

-h or --help
displays help information for the command. To view the man page, enter **man mon_procd**.

-v or --version
displays version information for the command.

Examples of command-line use

- To start mon_procd with default setting:

```
> mon_procd
```

- To start mon_procd with a sampling interval of 30 seconds:

```
> mon_procd -i 30
```

- To start mon_procd and have it run in the foreground:

```
> mon_procd -a
```

- To start mon_procd with a sampling interval of 45 seconds and have it run in the foreground:

```
> mon_procd -a -i 45
```

Processing monitor data

The mon_procd daemon writes process data to the z/VM monitor stream.

The data includes system summary information and information of each process for up to 100 processes currently being managed by an instance of Linux on z/VM to the z/VM monitor stream.

At the time of the sample interval, one sample monitor record is written for system summary data. Then, one sample monitor record is written for each process for up to 100 processes currently being managed by the Linux instance. If more than 100 processes exist in a Linux instance at a given time, processes are sorted by the sum of CPU and memory usage percentage values. Only the top 100 processes' data is written to the z/VM monitor stream.

The monitor data in each record begins with a header (a time stamp, the length of the data, and the offset). The data after the header depends on the field "record number" of the 16-bit product ID and can be summary data or process data. See “Reading the monitor data” on page 652 for details.

Table 65. System summary data format

Type	Name	Description
__u64	time_stamp	Time at which the process data was sampled.
__u16	data_len	Length of data that follows the header.
__u16	data_offset	Offset from start of the header to the start of the process data.
__u64	uptime	Uptime of the Linux instance.
__u32	users	Number of users on the Linux instance.
char[6]	loadavg_1	Load average over the last 1 minute.
char[6]	loadavg_5	Load average over the last 5 minutes.
char[6]	loadavg_15	Load average over the last 15 minutes.
__u32	task_total	total number of tasks on the Linux instance.
__u32	task_running	Number of running tasks.
__u32	task_sleeping	Number of sleeping tasks.
__u32	task_stopped	Number of stopped tasks.
__u32	task_zombie	Number of zombie tasks.
__u32	num_cpus	Number of CPUs.
__u16	puser	A number that represents (100 * percentage of total CPU time used for normal processes executing in user mode).
__u16	pnice	A number that represents (100 * percentage of total CPU time used for niced processes executing in user mode).
__u16	psystem	A number that represents (100 * percentage of total CPU time used for processes executing in kernel mode).
__u16	pidle	A number that represents (100 * percentage of total CPU idle time).
__u16	piowait	A number that represents (100 * percentage of total CPU time used for I/O wait).
__u16	pirq	A number that represents (100 * percentage of total CPU time used for interrupts).
__u16	psoftirq	A number that represents (100 * percentage of total CPU time used for softirqs).

Table 65. System summary data format (continued)

Type	Name	Description
__u16	psteal	A number that represents (100 * percentage of total CPU time spent in stealing).
__u64	mem_total	Total memory in KB.
__u64	mem_used	Used memory in KB.
__u64	mem_free	Free memory in KB.
__u64	mem_buffers	Memory in buffer cache in KB.
__u64	mem_pgpgin	Data read from disk in KB.
__u64	mem_pgpgout	Data written to disk in KB
__u64	swap_total	Total swap memory in KB.
__u64	swap_used	Used swap memory in KB.
__u64	swap_free	Free swap memory in KB.
__u64	swap_cached	Cached swap memory in KB.
__u64	swap_pswpin	Pages that are swapped in.
__u64	swap_pswpout	Pages that are swapped out.

The following is the format of a process information data that is passed to the z/VM monitor stream.

Table 66. Process data format

Type	Name	Description
__u64	time_stamp	Time at which the process data was sampled.
__u16	data_len	Length of data that follows the header.
__u16	data_offset	Offset from start of the header to the start of the process data.
__u32	pid	ID of the process.
__u32	ppid	ID of the process parent.
__u32	euid	Effective user ID of the process owner.
__u16	tty	Device number of the controlling terminal or 0.
__s16	priority	Priority of the process
__s16	nice	Nice value of the process.
__u32	processor	Last used processor.
__u16	pcpu	A number that represents (100 * percentage of the elapsed cpu time that is used by the process since last sampling).
__u16	pmem	A number that represents (100 * percentage of physical memory that is used by the process).
__u64	total_time	Total cpu time the process used.
__u64	ctotal_time	Total cpu time the process and its dead child processes used.
__u64	size	Total virtual memory that is used by the task in KB.
__u64	swap	Swapped out portion of the virtual memory in KB.
__u64	resident	Non-swapped physical memory that is used by the task in KB.

Table 66. Process data format (continued)

Type	Name	Description
__u64	trs	Physical memory that is devoted to executable code in KB.
__u64	drs	Physical memory that is devoted to other than executable code in KB.
__u64	share	Shared memory that is used by the task in KB.
__u64	dt	Dirty page count.
__u64	maj_flt	Number of major page faults occurred for the process.
char	state	Status of the process.
__u32	flags	The process current scheduling flags.
__u16	ruser_len	Length of real user name of the process owner and should not be larger than 64.
char[ruser_len]	ruser	Real user name of the process owner. If the name is longer than 64, the name is truncated to the length 64.
__u16	euser_len	Length of effective user name of the process owner and should not be larger than 64.
char[euser_len]	euser	Effective user name of the process owner. If the name is longer than 64, the name is truncated to the length 64.
__u16	egroup_len	Length of effective group name of the process owner and should not be larger than 64.
char [egroup_len]	egroup	Effective group name of the process owner. If the name is longer than 64, the name is truncated to the length 64.
__u16	wchan_len	Length of sleeping in function's name and should not be larger than 64.
char[wchan_len]	wchan_name	Name of sleeping in function or '-'. If the name is longer than 64, the name is truncated to the length 64.
__u16	cmd_len	Length of command name or program name that is used to start the process and should not be larger than 64.
char[cmd_len]	cmd	Command or program name that is used to start the process. If the name is longer than 64, the name is truncated to the length 64.
__u16	cmd_line_len	Length of command line that is used to start the process and should not be larger than 1024.
char [cmd_line_len]	cmd_line	Command line that is used to start the process. If the name is longer than 1024, the name is truncated to the length 1024.

Use the time_stamp to correlate all process information that were sampled in a given interval.

Reading the monitor data

All records that are written to the z/VM monitor stream begin with a product identifier.

The product ID is a 16-byte structure of the form pppppppffnvvrrmm, where for records that are written by mon_procd, these values are:

ppppppp
is a fixed ASCII string LNXAPPL.

ff	is the application number for mon_procd = x'0002'.
n	is the record number as follows: • x'00' indicates summary data. • x'01' indicates process data.
vv	is the version number = x'0000'.
rr	is the release number, which can be used to mark different versions of process APPLDATA records.
mm	is reserved for mon_procd and should be ignored.

Note: Though the mod_level field (mm) of the product ID varies, there is no relationship between any particular mod_level and process. The mod_level field should be ignored by the reader of this monitor data.

This item uses at most 101 monitor buffer records from the monwriter device driver. A maximum number of buffers is set when a monwriter module is loaded. Because of this, the maximum number of buffers must not be less than the sum of buffer records that are used by all monwriter applications.

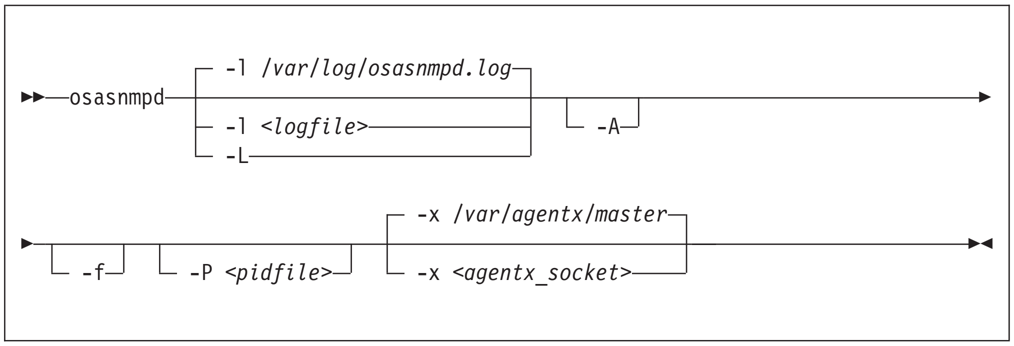
There are many tools available to read z/VM monitor data. One such tool is the Linux monreader character device driver. For more information about monreader, see Chapter 34, “Reading z/VM monitor records,” on page 431.

osasnmpd – Start OSA-Express SNMP subagent

Use the **osasnmpd** command to start the OSA-Express Simple Network Management Protocol (SNMP) subagent (osasnmpd).

See Chapter 16, "OSA-Express SNMP subagent support," on page 319 for information about SNMP agent and osasnmpd subagent setup and usage.

osasnmpd syntax



-l or --logfile *<logfile>*
: specifies a file for logging all subagent messages and warnings, including stdout and stderr. If no path is specified, the log file is created in the current directory. The default log file is `/var/log/osasnmpd.log`.

-L or --stderrlog
: prints 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.

-h or --help
: displays help information for the command.

-v or --version
: displays version information for the command.

Examples

To start the osasnmpd subagent with all default settings:

```
# osasnmpd
```

qetharp - Query and modify ARP data

Use the **qetharp** command to query and purge address data such as MAC and IP addresses from the ARP cache of the OSA and HiperSockets hardware.

Before you begin:

- The **qetharp** command applies only to devices in layer 3 mode (see "Layer 2 and layer 3" on page 258).
- The **qetharp** command supports IPv6 only for real HiperSockets and z/VM guest LAN HiperSockets.
- For HiperSockets, z/VM guest LAN and VSWITCH interfaces, the **qetharp** command supports only the **--query** option.

qetharp syntax

```
►►──qetharp──┬─┬─────┬──┬────┬─ -q <interface>──────────────────────┬──────────►◄
             │ ├─ -n─┤  └ -6─┘                                       │
             │ └─ -c─┘                                               │
             ├─ -a <interface>── -i <ip_address>── -m <mac_address>──┤
             ├─ -d <interface>── -i <ip_address>─────────────────────┤
             └─ -p <interface>───────────────────────────────────────┘
```

Where:

-q or --query
: shows the address resolution protocol (ARP) information about the specified network interface. Depending on the device that the interface was assigned to, this information is obtained from an OSA feature's ARP cache or a HiperSockets ARP cache.

  The default command output shows symbolic host names and includes only numerical addresses for host names that cannot be resolved. Use the **-n** option to show numerical addresses instead of host names.

  By default, qetharp omits IPv6 related information. Use the **-6** option to include IPv6 information for HiperSockets.

-n or --numeric
: shows numerical addresses instead of trying to resolve the addresses to the symbolic host names. This option can be used only with the **-q** option.

-c or --compact
: limits the output to numerical addresses only. This option can be used only with the **-q** option.

-6 or --ipv6
: includes IPv6 information for HiperSockets. For real HiperSockets, shows the IPv6 addresses. For guest LAN HiperSockets, shows the IPv6 to MAC address mappings. This option can be used only with the **-q** option.

<interface>
: specifies the qeth interface to which the command applies.

-a or --add
: adds a static ARP entry to the OSA adapter. Static entries can be deleted with **-d**.

-d or --delete
deletes a static ARP entry from the OSA adapter. Static entries are created with **-a**.

-p or --purge
flushes the ARP cache of the OSA. The cache contains dynamic ARP entries, which the OSA adapter creates through ARP queries. After flushing the cache, the OSA adapter creates new dynamic entries. This option works only with OSA devices. qetharp returns immediately.

-i *<ip_address>* **or --ip** *<ip_address>*
specifies the IP address to be added to or removed from the OSA adapter.

-m *<mac_address>* **or --mac** *<mac_address>*
specifies the MAC address to be added to the OSA adapter.

-v or --version
displays version information and exits.

-h or --help
displays usage information and exits. To view the man page, enter **man qetharp**.

Examples

- Show all ARP entries of the OSA defined as eth0:

```
# qetharp -q eth0
```

- Show all ARP entries of the HiperSockets interface that is defined as hsi0 including IPv6 entries:

```
qetharp -6q hsi0
```

- Show all ARP entries of the OSA defined as eth0 without resolving host names:

```
# qetharp -nq eth0
```

- Show all ARP entries, including IPv6 entries, of the HiperSockets interface that is defined as hsi0 without resolving host names:

```
qetharp -n6q hsi0
```

- Flush the OSA ARP cache for eth0:

```
# qetharp -p eth0
```

- Add a static entry for eth0 and IP address 1.2.3.4 to the OSA ARP cache, with 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 ARP cache.

```
# qetharp -d eth0 -i 1.2.3.4
```

qethconf - Configure qeth devices

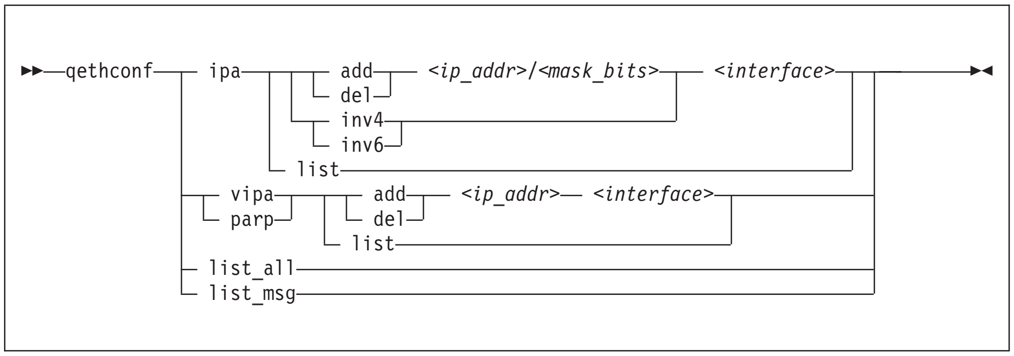
Use the **qethconf** command to configure IP address takeover, virtual IP address (VIPA), and proxy ARP for layer3 qeth devices.

See Chapter 15, “qeth device driver for OSA-Express (QDIO) and HiperSockets,” on page 251 for details about the following concepts:
- IP address takeover
- VIPA (virtual IP address)
- Proxy ARP

You cannot use this command with the layer2 option.

From the arguments that are specified, **qethconf** assembles the function command and redirects it to the corresponding sysfs attributes. You can also use **qethconf** to list the already defined entries.

qethconf syntax



The **qethconf** command has these function keywords:

ipa
: configures qeth for IP address takeover (IPA).

vipa
: configures qeth for virtual IP address (VIPA).

parp or rxip
: configures qeth for proxy ARP.

The **qethconf** command has these action keywords:

add
: adds an IP address or address range.

del
: deletes an IP address or address range.

inv4
: inverts the selection of address ranges for IPv4 address takeover. This inversion makes the list of IP addresses that was specified with `qethconf add` and `qethconf del` an exclusion list.

inv6
: inverts the selection of address ranges for IPv6 address takeover. This inversion makes the list of IP addresses that was specified with `qethconf add` and `qethconf del` an exclusion list.

list
: lists existing definitions for specified qeth function.

list_all
: lists existing definitions for IPA, VIPA, and proxy ARP.

<ip_addr>
: specifies the IP address. Can be specified in one of these formats:
  - IP version 4 format, for example, `192.168.10.38`
  - IP version 6 format, for example, `FE80::1:800:23e7:f5db`
  - 8- or 32-character hexadecimals prefixed with `-x`, for example, `-xc0a80a26`

<mask_bits>
: specifies the number of bits that are set in the network mask. Enables you to specify an address range.

  Example: A *<mask_bits>* of 24 corresponds to a network mask of `255.255.255.0`.

<interface>
: specifies the name of the interface that is associated with the specified address or address range.

list_msg
: lists qethconf messages and explanations.

-h or --help
: displays help information. To view the man page, enter **man qethconf**.

-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 that are destined for 1.2.3.5:

```
# qethconf parp add 1.2.3.5 eth0
qethconf: Added 1.2.3.5 to /sys/class/net/eth0/device/rxip/add4.
qethconf: Use "qethconf parp list" to check for the result
```

  Confirm the new proxy ARP definitions:

```
# qethconf parp list
parp add 1.2.3.4 eth0
parp add 1.2.3.5 eth0
```

- Configure eth0 for IP address takeover for all addresses that start with 192.168.10:

```
# qethconf ipa add 192.168.10.0/24 eth0
qethconf: Added 192.168.10.0/24 to /sys/class/net/eth0/device/ipa_takeover/add4.
qethconf: Use "qethconf ipa list" to check for the result
```

Display the new IP address takeover definitions:

```
# qethconf ipa list
ipa add 192.168.10.0/24 eth0
```

- Configure VIPA for eth1:

```
# qethconf vipa add 10.99.3.3 eth1
qethconf: Added 10.99.3.3 to /sys/class/net/eth1/device/vipa/add4.
qethconf: Use "qethconf vipa list" to check for the result
```

Display the new VIPA definitions:

```
# qethconf vipa list
vipa add 10.99.3.3 eth1
```

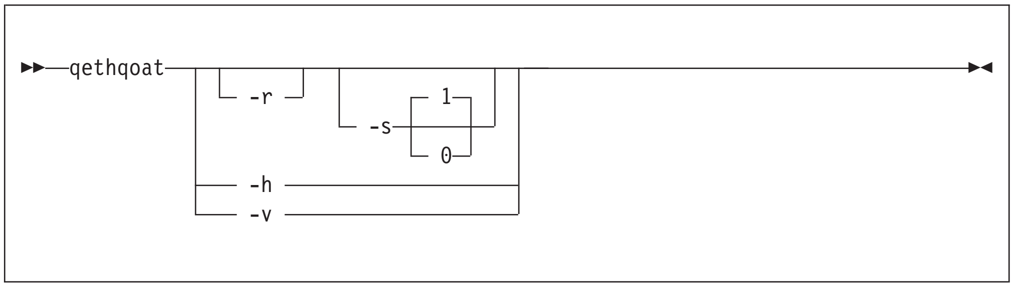
- List all existing IPA, VIPA, and proxy ARP definitions.

```
# qethconf list_all
parp add 1.2.3.4 eth0
parp add 1.2.3.5 eth0
ipa add 192.168.10.0/24 eth0
vipa add 10.99.3.3 eth1
```

qethqoat - Query OSA address table

Use the **qethqoat** command to query the OSA address table and display physical and logical device information.

qethqoat syntax



where:

`-r or --raw`
writes raw data to stdout.

`-s or --scope`
defines the scope of the query. The following values are valid:

0 queries the level of the OSA address table.

1 interface (this option is the default).

`-h or --help`
displays help information. To view the man page, enter **`man qethqoat`**.

`-v or --version`
displays version information.

Examples

To display physical and logical device information for interface eth0, issue:

```
# qethqoat eth0
PCHID: 0x05f1
CHPID: 0x76
Manufacturer MAC address: 00:14:5e:76:a2:40
Configured MAC address: 00:00:00:00:00:00
Data device sub-channel address: 0xf5f2
CULA: 0x00
Unit address: 0x53
Physical port number: 0
Number of output queues: 1
Number of input queues: 1
Number of active input queues: 0
Interface flags: 0x0e000000
OSA Generation:  OSA-Express5S
Port speed/mode: 1000 mbs / full duplex
Port media type: copper
Jumbo frames: yes
Firmware: 0x00000010

IPv4 router: no
IPv6 router: no
IPv4 vmac router: no
IPv6 vmac router: no
Connection isolation: not active
Connection isolation VEPA: no
IPv4 assists enabled: 0x00111c77
IPv6 assists enabled: 0x00215c60
IPv4 outbound checksum enabled: 0x0000001a
IPv6 outbound checksum enabled: 0x00000000
IPv4 inbound checksum enabled: 0x0000001a
IPv6 inbound checksum enabled: 0x00000000

IPv4 Address:                          IPA Flags:
-------------                          ----------
192.0.2.0                              0x00000000

IPv4 Multicast Address:                MAC Address:
-----------------------                ------------
224.0.0.1                              01:00:5e:00:00:01

IPv6 Address:                          IPA Flags:
-------------                          ----------
2001:DB8:0:0:0:0:0:0                   0x00000001

IPv6 Multicast Address:                MAC Address:
-----------------------                ------------
ff02::1                                33:33:00:00:00:01
ff02::1:ff76:a240                      33:33:ff:76:a2:40
ff02::202                              33:33:00:00:02:02
```

This example uses scope 0 to query the supported OAT level and descriptor header types.

```
# qethqoat -s 0 eth0
Supported Scope mask: 0x00000001
Supported Descriptor hdr types: 0x0001070f
```

This example shows how the binary output from **qethqoat** can be processed in another tool. Here it is displayed in a hexdump viewer:

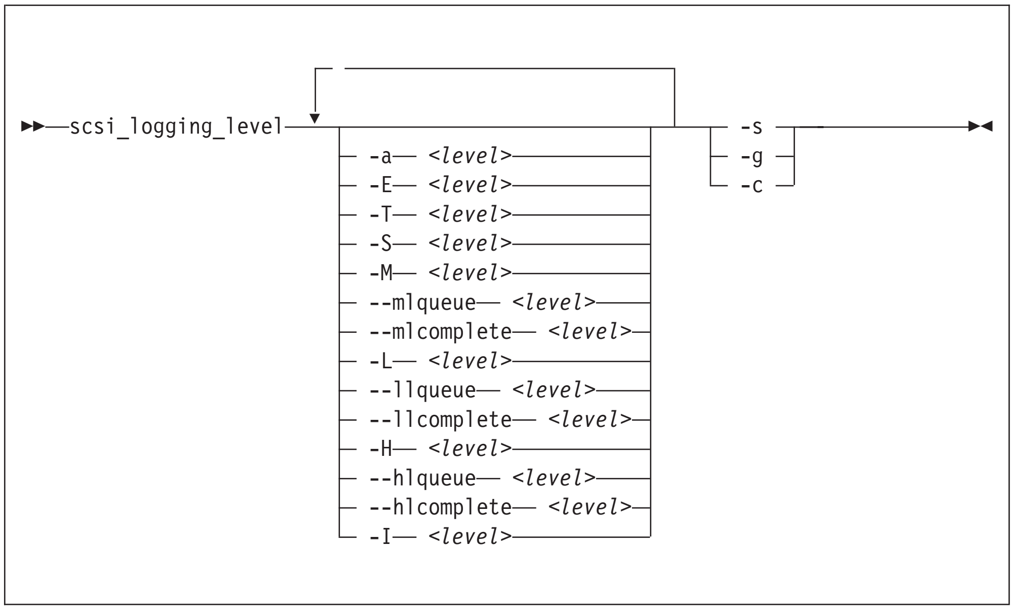
```
# qethqoat -r eth0 | hexdump
0000000 0158 0000 0008 0000 0000 0101 0000 0000
0000010 0000 0001 0000 0000 0000 0000 0000 0000
0000020 0004 0050 0001 0000 0000 0000 d7c8 4040
0000030 0120 0094 001a 643b 8a22 0000 0000 0000
0000040 e102 0002 0000 0004 0001 0000 0800 0000
0000050 0100 0480 0000 0766 0000 0000 0000 0000
0000060 0000 0000 0000 0000 0000 0000 0000 0000
0000070 0008 0060 0001 0000 0000 0000 d3c8 4040
0000080 0000 0000 0000 0000 0000 0000 0000 0000
0000090 0000 0000 0000 0000 0000 0000 0011 1c77
00000a0 0021 5c60 0000 001a 0000 0000 0000 001a
00000b0 0000 0000 0000 0000 0000 0000 0000 0000
00000c0 0002 0000 0000 0000 0000 0000 0000 0000
00000d0 0010 0030 0001 0000 0000 0000 c4c8 f4d4
00000e0 0000 0002 0000 0000 0000 0001 0000 0010
00000f0 0001 0001 0000 0000 0000 0000 0000 0000
0000100 e000 0001 0100 5e00 0001 0000 0000 0000
0000110 0010 0030 0001 0000 0000 0000 c4c8 f6d4
0000120 0000 0008 0000 0000 0000 0001 0000 0018
0000130 0001 0001 0000 0000 0000 0000 0000 0000
0000140 ff02 0000 0000 0000 0000 0000 0000 0001
0000150 3333 0000 0001 0000
0000158
```

scsi_logging_level - Set and get the SCSI logging level

Use the **`scsi_logging_level`** command to create, set, or get the SCSI logging level.

The SCSI logging feature is controlled by a 32-bit value – the SCSI logging level. This value is divided into 3-bit fields that describe the log level of a specific log area. Due to the 3-bit subdivision, setting levels or interpreting the meaning of current levels of the SCSI logging feature is not trivial. The scsi_logging_level script helps with both tasks.

scsi_logging_level syntax



Where:

`-a or --all` *`<level>`*
: specifies value for all SCSI_LOG fields.

`-E or --error` *`<level>`*
: specifies SCSI_LOG_ERROR.

`-T or --timeout` *`<level>`*
: specifies SCSI_LOG_TIMEOUT.

`-S or --scan` *`<level>`*
: specifies SCSI_LOG_SCAN.

`-M or --midlevel` *`<level>`*
: specifies SCSI_LOG_MLQUEUE and SCSI_LOG_MLCOMPLETE.

`--mlqueue` *`<level>`*
: specifies SCSI_LOG_MLQUEUE.

`--mlcomplete` *`<level>`*
: specifies SCSI_LOG_MLCOMPLETE.

`-L or --lowlevel` *`<level>`*
: specifies SCSI_LOG_LLQUEUE and SCSI_LOG_LLCOMPLETE.

`--llqueue` *`<level>`*
: specifies SCSI_LOG_LLQUEUE.

--llcomplete *<level>*
specifies SCSI_LOG_LLCOMPLETE.

-H or --highlevel *<level>*
specifies SCSI_LOG_HLQUEUE and SCSI_LOG_HLCOMPLETE.

--hlqueue *<level>*
specifies SCSI_LOG_HLQUEUE.

--hlcomplete *<level>*
specifies SCSI_LOG_HLCOMPLETE.

-I or --ioctl *<level>*
specifies SCSI_LOG_IOCTL.

-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.

-v or --version
displays version information.

-h or --help
displays help text.

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 displays the logging word of the SCSI logging feature and each logging level.

```
#> scsi_logging_level -g
Current scsi logging level:
dev.scsi.logging_level = 0
SCSI_LOG_ERROR=0
SCSI_LOG_TIMEOUT=0
SCSI_LOG_SCAN=0
SCSI_LOG_MLQUEUE=0
SCSI_LOG_MLCOMPLETE=0
SCSI_LOG_LLQUEUE=0
SCSI_LOG_LLCOMPLETE=0
SCSI_LOG_HLQUEUE=0
SCSI_LOG_HLCOMPLETE=0
SCSI_LOG_IOCTL=0
```

- This command sets all logging levels to 3:

```
#> scsi_logging_level -s -a 3
New scsi logging level:
dev.scsi.logging_level = 460175067
SCSI_LOG_ERROR=3
SCSI_LOG_TIMEOUT=3
SCSI_LOG_SCAN=3
SCSI_LOG_MLQUEUE=3
SCSI_LOG_MLCOMPLETE=3
SCSI_LOG_LLQUEUE=3
SCSI_LOG_LLCOMPLETE=3
SCSI_LOG_HLQUEUE=3
SCSI_LOG_HLCOMPLETE=3
SCSI_LOG_IOCTL=3
```

- This command sets SCSI_LOG_HLQUEUE=3, SCSI_LOG_HLCOMPLETE=2 and assigns all other SCSI_LOG fields the value 1.

```
# scsi_logging_level --hlqueue 3 --highlevel 2 --all 1 -s
New scsi logging level:
dev.scsi.logging_level = 174363209
SCSI_LOG_ERROR=1
SCSI_LOG_TIMEOUT=1
SCSI_LOG_SCAN=1
SCSI_LOG_MLQUEUE=1
SCSI_LOG_MLCOMPLETE=1
SCSI_LOG_LLQUEUE=1
SCSI_LOG_LLCOMPLETE=1
SCSI_LOG_HLQUEUE=3
SCSI_LOG_HLCOMPLETE=2
SCSI_LOG_IOCTL=1
```

sncap - Manage CPU capacity

Use the simple network System z CPU capacity management (**`sncap`**) command to specify a temporary capacity record activation or deactivation and operation parameters for a CPC. This command can control the Capacity BackUp (CBU), Capacity for Planned Events, and On/Off Capacity On-Demand (OOCOD) temporary capacity records.

For command return codes, see the man page.

The **`sncap`** command is part of the **`snipl`** package as of version 2.3.0. You can obtain the **`snipl`** package at www.ibm.com/developerworks/linux/linux390/snipl.html.

Before you begin:

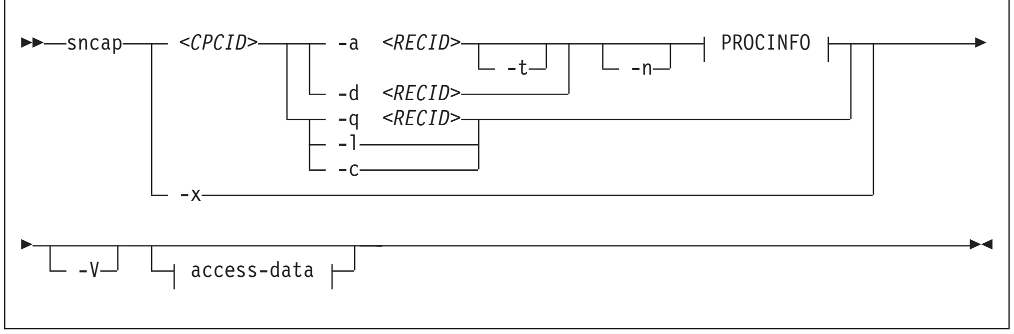
- **`sncap`** requires the Support Element (SE) and Hardware Management Console (HMC) software version 2.10.0 or later. The command can operate only with the records that are installed on the SE.
- **`sncap`** uses the management application programming interfaces (APIs) provided by the SE or HMC (HWMCAAPI API servers). For information about the management APIs of the SE and the HMC, see *System z Application Programming Interfaces*, SB10-7030, available from IBM Resource Link at `www.ibm.com/servers/resourcelink`.

  To communicate with the server, **`sncap`** establishes a network connection and uses the SNMP protocol to send and retrieve data by using HWMCAAPI API calls. The server must be configured to allow the initiating host system to access the API.

Note:

- A temporary capacity record activation or deactivation command might cancel due to a timeout. The timeout is indicated by return code 12 - "Timeout occurred, the command is canceled" or 18 - "An error was received from the HWMCAAPI API", with the short message about the timeout. If a timeout occurs, the requested operation can still continue to run on the CPC support element and potentially complete successfully. Use **`-q`** to investigate the state of the record before you issue the next command for that CPC.
- The **`sncap`** command processes cannot be run in parallel for the same CPC for temporary capacity record activation or deactivation. Also, a **`sncap`** process that is started for a temporary capacity record activation or deactivation cannot run in parallel with a **`snipl`** process for the same CPC.

sncap syntax



where:

<CPCID>
: identifies the Central Processing Complex that is specified in the SE network settings configuration. This parameter also identifies the configuration file section where the server connection parameters can be specified, see `access-data`. Find the *<CPCID>* value either in the SE user interface in the Customize Network Settings window under the Identification tab, or by using **sncap** with the **-x** option. This parameter is mandatory for the activation, deactivation, and query operations. Specify it as a command-line argument.

<RECID>
: identifies a temporary capacity record that is installed on the SE that you want to work with.

-a or --activate
: activates the temporary capacity record with the record identifier *<RECID>* and processor parameters PROCINFO. The PROCINFO parameters must be specified for the record activation.

-t or --test
: specifies the temporary capacity record activation in the test mode for up to 10 days. The test mode allows temporary record activation the number of times that are specified in the record definition. Real activation is possible only while no test is active. This option can be used only with the **-a** option.

-d or --deactivate
: deactivates the temporary capacity record with the record identifier *<RECID>* and processor parameters PROCINFO on the CPC *<CPCID>*. If the PROCINFO data is specified, **-d** deactivates only the specified processors in the CPC configuration. If the PROCINFO parameters are not specified, **-d** deactivates the entire record.

-n or --no_record_changes
: skips any actions that would change the records. This mode can be used for debugging purposes. When specified, **sncap** does not change the temporary capacity record state during the record activation or deactivation. It assumes that the activation or deactivation request is always successful. The querying functions run as in the regular mode.

-x or --list_cpcs
: sends the output of the list of CPCs that are defined on an SE or HMC to standard out. The list contains the CPC identifiers and its support element version numbers. When the **-x** option is specified, the **-S** specification of the server IP address or DNS name is required as part of the access data.

-q or --query
: displays detailed information about the temporary capacity record *<RECID>*, installed on the specified CPC. The information includes the data for the available CPU capacity models that are defined in the record and the current CPC processor capacity parameters.

  - If the temporary capacity record activation parameters have a value of `-1`, the parameter value is unlimited.
  - The negative value of PU or CLI in the Available Model Capacity Identifiers table designates the number of PU or CLI to be deactivated to achieve the listed model capacity.
  - If the maximum quantity of CP type PUs shows an asterisk (*), all the PUs defined in the temporary capacity record can be activated as the CP type PUs.

-l or --list_records
displays the list of temporary capacity records that are installed on the specified CPC. A value of -1 in the Real Act. and Test Act. report fields means that there is an unlimited number of available record activation attempts.

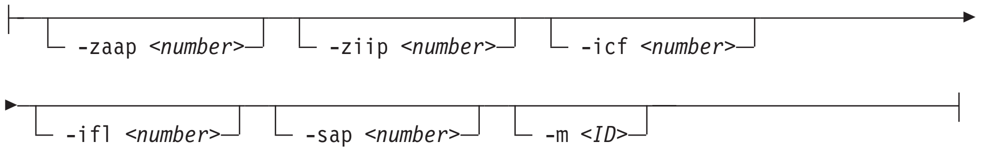
-c or --pu_configuration
displays the information about the current CPC processing unit configuration, including the number of active processors, the processors available for temporary activation, model capacity identifier, and current MSUs available on the CPC. A minus sign (-) in the report fields means that the value is not applicable for temporary or permanent configuration.

-V or --verbose
displays information useful for debugging.

-v or --version
displays the version number of **sncap**, then exits.

-h or --help
displays a short usage description and exits. To view the man page, issue **man sncap**.

PROCINFO:



Specifies the processor types and quantities to be activated or deactivated on the CPC <*CPCID*> to change the record activation level. The temporary capacity record activation operation requires the PROCINFO parameters. The PROCINFO parameters can be omitted for the record deactivation. If no specific processor type is specified, all the processors from the temporary capacity record are deactivated in the CPC. The model capacity identifier is set to the minimal available model capacity value. Each processor type can be specified only once. If more processors are specified for activation or deactivation than are defined in the record, the command returns with return code 17. No processors are activated or deactivated.

--zaap <number>
specifies the number of zAAP processors to be activated or deactivated.

--ziip <number>
specifies the number of zIIP processors to be activated or deactivated.

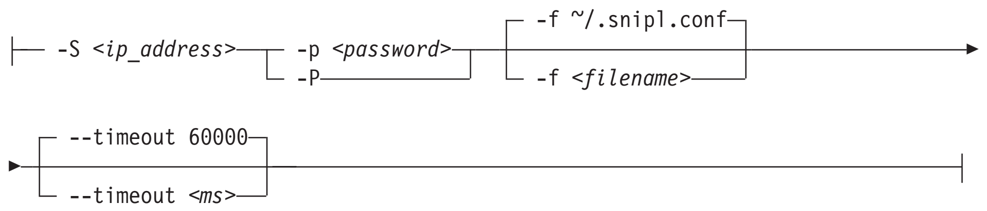
--icf <number>
specifies the number of ICF processors to be activated or deactivated.

--ifl <number>
specifies the number of IFL processors to be activated or deactivated.

--sap <number>
specifies the number of SAP processors in the PROCINFO parameters to be activated or deactivated.

-m or --model-capacity *<ID>*
specifies the model capacity identifier <*ID*> to be activated by the command. The model capacity identifiers are supplied in the temporary capacity record. They can be found either by using the support element user interface, or the `--query` <*RECID*> option of the **sncap** application. Use the model capacity identifier to control the number of CP processors and the Capacity Level Indicator value to be activated or deactivated to achieve the target CPU capacity model. Also, the model capacity identifier influenced the Target MSU Value and MSU Cost parameters. If the `-m` option is specified without the processor types and quantities, it activates or deactivates only the specified capacity model. It then leaves the active auxiliary processor quantities unchanged.

access-data:



-S or --se *<ip_address>*
specifies the IP address or DNS name for the SE or HMC that controls the CPC you want to work with. You can omit this parameter if the SE or HMC IP address or DNS name and community are specified in the sncap configuration file. The IP address of SE or HMC is identified in the configuration file with the cpcid attribute.

-p or --password *<password>*
specifies the password (community) from the SNMP configuration settings on the SE or HMC that controls the CPC you want to work with. This parameter is required. It must be specified either in the command line or in the configuration file. Alternatively, use the `-P` option to prompt the user for the password.

-P or --promptpassword
prompts for a password (community) in protected entry mode.

-f or --configfilename *<filename>*
specifies the name of the sncap configuration file that maps CPC identifiers to the corresponding specifications for the SE or HMC addresses and passwords. If no configuration file is specified, the user-specific default file `~/.snipl.conf` is used. If this file does not exist, the system default file `/etc/snipl.conf` is used. A connection to server requires specification of the CPC ID, the SE or HMC IP address or DNS name, and the password (community). If only the <*CPCID*> parameter is specified on the command line, it identifies the section of the configuration file that contains the credentials values. If the CPC ID and the server IP address are specified, **sncap** looks for the password in the configuration file using the server IP address for the configuration file section identification. If your specification maps to multiple sections, the first match is processed. If conflicting specifications of credentials are provided through the command line and the configuration file, the command-line specification is

used. If no configuration file is specified or available at the default locations, all required parameters must be specified on the command line.

--timeout *<ms>*
specifies the timeout in milliseconds for general management API calls. The default is 60000 ms.

Configuration file structure

Any required connection parameters that are not provided on the command line must be specified through the configuration file. The command-line specifications override specifications in the configuration file. The **sncap** command uses the CPC identifier to select the configuration file sections to retrieve the relevant connection parameters. You must specify the CPC identifier on the command line for all sncap operations except the `-x` option. The `-x` option is used to retrieve the CPC identifier list that is defined on a server.

The structure of the sncap configuration file is similar to the snipl configuration file structure. You can use the snipl configuration file with sncap if you add the CPC identifiers to the snipl server definition sections by using the cpcid keyword. The cpcid keywords can be added only to the support element HWMCAAPI API server definitions (LPAR type sections). They cannot be added to the configuration file sections of the VM type. VM type sections define connections to z/VM systems in the snipl configuration file and sncap can connect only to SEs or HMCs.

An sncap configuration file contains one or more sections. Each section consists of multiple lines with specifications of the form `<keyword>=<value>` for an SE or HMC. The **sncap** command identifies the sections by using the CPC identifier. To retrieve the connection parameters from the configuration file, at least the CPC identifier must be specified on the command line. If both the server IP address (or DNS name) and the CPC identifier are specified on the command line, the password is selected in the configuration file by using the server IP address (or DNS name). When you use the `-x` command-line option to get the list of defined CPCs on a server, specify only the server IP address (or DNS name) on the command line.

The following rules apply to the configuration file:
- Lines that begin with a number sign (#) are comment lines.
- A number sign in the middle of a line makes the remaining line a comment.
- Empty lines are allowed.
- The specifications are not case-sensitive.
- In a *<keyword>*=*<value>* pair, one or more blanks are allowed before or after the equal sign (=).

The following list maps the configuration file keywords to command line equivalents:

server (required, once per section) starts a configuration file section by specifying the IP address or DNS name of an SE or HMC. This attribute is equivalent to the `--se` command-line argument.

password
(optional, at most once per section) specifies the password (community) from the SNMP settings of the SE or HMC. If omitted, you must specify

the password in the **sncap** command-line arguments. Alternatively, use -P option to prompt the user for the password. This attribute specifies the --password command-line argument.

cpcid (required, at least once per section) specifies the Central Processing Complex name that is defined in the hardware. This server attribute is used to map the CPC identifier to the server IP address (DNS name) and password. There can be more than one cpcid entry in a section if the server is an HMC.

type (optional, at most once per section) specifies the server type. This parameter is used to provide compatibility with the snipl configuration file. If it is specified, it must have the value "LPAR".

Sample configuration file

```
-----------------------------------------------------
# Comment line (ignored).
#
# A section that defines a support element connection.
#
Server = 192.0.2.4
type = LPAR
cpcid = SZ01CP00
password = pw42play
#
# A section that defines a hardware management console
# connection.
#
Server = 192.0.2.2
type = LPAR
cpcid = SZ02CP00
cpcid = SZ02CP01
cpcid = SZ02CP03
cpcid = SZ02CP04
password= pw42play
<EOF>
----------------------------------------------------------
```

Examples

- To activate a CBU temporary capacity record CB7KHB38 on CPC SCZP201 to temporarily upgrade it to model capacity identifier 741:

  ```
  sncap SCZP201 -S 192.0.2.4 -P -a CB7KHB38 -m 741
  ```
- To activate only a subset of processors defined in temporary capacity record CB7KHB38 on the CPC SCZP201:

  ```
  sncap SCZP201 -S 192.0.2.4 -P -a CB7KHB38 --zaap 2 --ziip 2
  ```
- To deactivate a CBU temporary capacity record CB7KH38 on the CPC SCZP201:

  ```
  sncap SCZP201 -S 192.0.2.4 -P -d CB7KHB38
  ```
- To deactivate only a subset of processors defined in temporary capacity record CB7KHB38 on the CPC SCZP201:

  ```
  sncap SCZP201 -S 192.0.2.4 -P -d CB7KHB38 --zaap 2 --ziip 2
  ```

  With a suitable configuration file at /etc/xcfg the previous command can be shortened to:

  ```
  sncap SCZP201 -f /etc/xcfg -d CB7KHB38 --zaap 2 --ziip 2
  ```

  With a suitable default configuration file the command can be further shortened to:

  ```
  sncap SCZP201 -d CB7KHB38 --zaap 2 --ziip 2
  ```

For information about the **sncap** report fields and sample workflows for the temporary capacity record installation, activation and deactivation, see the Redbooks publication *IBM System z10 Capacity on Demand*, SG24-7504 or any updates of this publication that applies to your mainframe system.

tape390_crypt - Manage tape encryption

Use the **tape390_crypt** command to enable and disable tape encryption for a channel attached tape device. You can also specify key encrypting keys (KEK) by using labels or hashes.

For 3592 tape devices, it is possible to write data in an encrypted format. The encryption keys are stored on an encryption key manager (EKM) server, which can run on any machine with TCP/IP and Java support. The EKM communicates with the tape drive over the tape control unit by using TCP/IP. The control unit acts as a proxy and forwards the traffic between the tape drive and the EKM. This type of setup is called out-of-band control-unit based encryption.

The EKM creates a data key that encrypts data. The data key itself is encrypted with KEKs and is stored in so called external encrypted data keys (EEDKs) on the tape medium.

You can store up to two EEDKs on the tape medium. With two EEDKs, one can contain a locally available KEK and the other can contain the public KEK of the location or company to where the tape is to be transferred. Then, the tape medium can be read in both locations.

When the tape device is mounted, the tape drive sends the EEDKs to the EKM. The EKM tries to unwrap one of the two EEDKs and sends back the extracted data key to the tape drive.

Linux can address KEKs by specifying either hashes or labels. Hashes and labels are stored in the EEDKs.

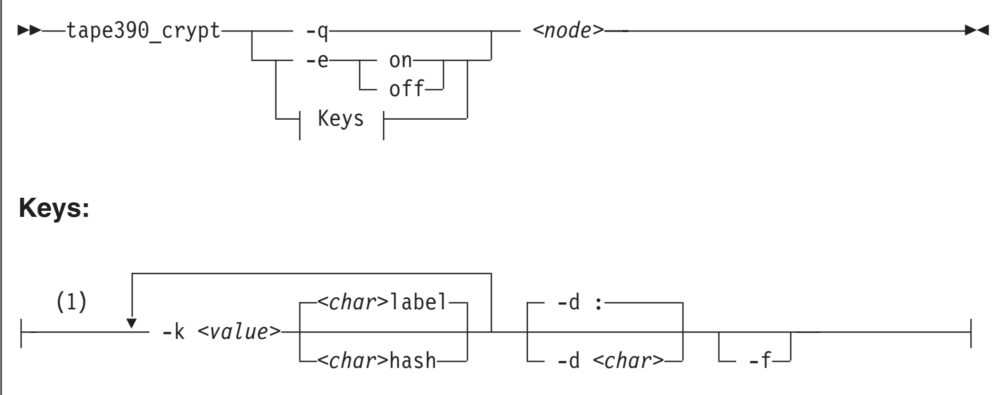
Note: If a tape is encrypted, it cannot be used for IPL.

Before you begin:

To use tape encryption, you need:

- A 3592 crypto-enabled tape device and control unit that is configured as system-managed encryption.
- A crypto-enabled 3590 channel-attached tape device driver. See Chapter 13, "Channel-attached tape device driver," on page 229.
- A key manager. See *Encryption Key Manager Component for the Java(TM) Platform Introduction, Planning, and User's Guide*, GA76-0418 for more information.

tape390_crypt syntax



Notes:

1 The `-k` or `--key` operand can be specified maximally twice.

where:

`-q or --query`
displays information about the tape's encryption status. If encryption is active and the medium is encrypted, additional information about the encryption keys is displayed.

`-e or --encryption`
sets tape encryption on or off.

`-k or --key`
sets tape encryption keys. You can specify the `-k` option only if the tape medium is loaded and rewound. While processing the `-k` option, the tape medium is initialized and all previous data contained on the tape medium is lost.

You can specify the `-k` option twice because the tape medium can store two EEDKs. If you specify the `-k` option once, two identical EEDKs are stored.

<value>
specifies the key encrypting key (KEK), which can be up to 64 characters long. The keywords **label** or **hash** specify how the KEK in *<value>* is to be stored on the tape medium. The default store type is **label**.

`-d or --delimiter`
specifies the character that separates the KEK in *<value>* from the store type (**label** or **hash**). The default delimiter is ":" (colon).

<char>
is a character that separates the KEK in *<value>* from the store type (**label** or **hash**).

`-f or --force`
specifies that no prompt message is to be issued before writing the KEK information and initializing the tape medium.

<node>
specifies the device node of the tape device.

-h or --help
: displays help text. To view the man page, enter **man tape390_crypt**.

-v or --version
: displays information about the version.

Examples

The following scenarios illustrate the most common use of tape encryption. In all examples `/dev/ntibm0` is used as the tape device.

Querying a tape device before and after encryption is turned on

This example shows a query of tape device `/dev/ntibm0`. Initially, encryption for this device is off. Encryption is then turned on, and the status is queried again.

```
tape390_crypt -q /dev/ntibm0
ENCRYPTION: OFF
MEDIUM: NOT ENCRYPTED

tape390_crypt -e on /dev/ntibm0

tape390_crypt -q /dev/ntibm0
ENCRYPTION: ON
MEDIUM: NOT ENCRYPTED
```

Then, two keys are set, one in label format and one in hash format. The status is queried and there is now additional output for the keys.

```
tape390_crypt -k my_first_key:label -k my_second_key:hash /dev/ntibm0
--->> ATTENTION! <<---
All data on tape /dev/ntibm0 will be lost.
Type "yes" to continue: yes
SUCCESS: key information set.

tape390_crypt -q /dev/ntibm0
ENCRYPTION: ON
MEDIUM: ENCRYPTED
KEY1:
 value:  my_first_key
 type:  label
 ontape:  label
KEY2:
 value:  my_second_key
 type:  label
 ontape:  hash
```

Using default keys for encryption

1. Load the cartridge. If the cartridge is already loaded:
   - Switch off encryption:
     `tape390_crypt -e off /dev/ntibm0`
   - Rewind:
     `mt -f /dev/ntibm0 rewind`
2. Switch encryption on:
   `tape390_crypt -e on /dev/ntibm0`
3. Write data.

Using specific keys for encryption

1. Load the cartridge. If the cartridge is already loaded, rewind:
   ```
   mt -f /dev/ntibm0 rewind
   ```
2. Switch encryption on:
   ```
   tape390_crypt -e on /dev/ntibm0
   ```
3. Set new keys:
   ```
   tape390_crpyt -k key1 -k key2 /dev/ntibm0
   ```
4. Write data.

Writing unencrypted data

1. Load the cartridge. If the cartridge is already loaded, rewind:
   ```
   mt -f /dev/ntibm0 rewind
   ```
2. If encryption is on, switch off encryption:
   ```
   tape390_crypt -e off /dev/ntibm0
   ```
3. Write data.

Appending new files to an encrypted cartridge

1. Load the cartridge
2. Switch encryption on:
   ```
   tape390_crypt -e on /dev/ntibm0
   ```
3. Position the tape.
4. Write data.

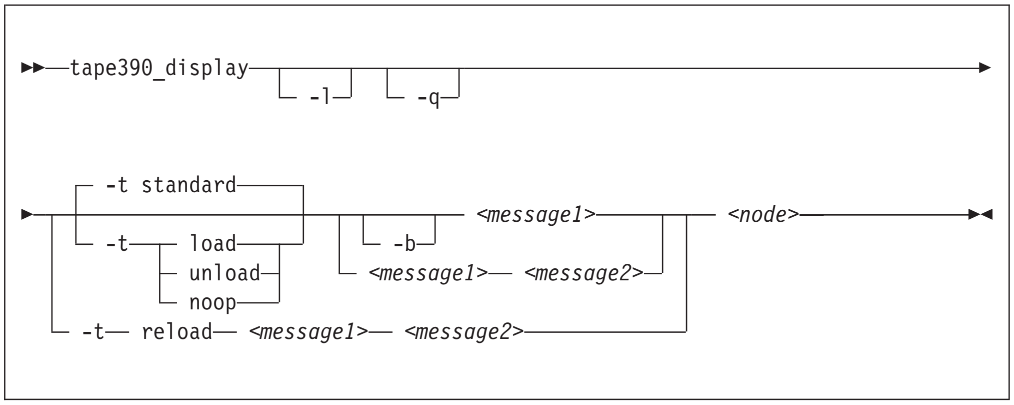
Reading an encrypted tape

1. Load the cartridge
2. Switch encryption on:
   ```
   tape390_crypt -e on /dev/ntibm0
   ```
3. Read data.

tape390_display - Display messages on tape devices and load tapes

Use the **`tape390_display`** command to show messages on the display unit of a physical tape device, optionally in conjunction with loading a tape.

tape390_display syntax



where:

`-l or --load`
instructs the tape unit to load the next indexed tape from the automatic tape loader (if installed). Ignored if no loader is installed or if the loader is not in “system” mode. The loader “system” mode allows the operating system to handle tape loads.

`-t or --type`
The possible values have the following meanings:

standard
displays the message or messages until the physical tape device processes the next tape movement command.

load displays the message or messages until a tape is loaded; if a tape is already loaded, the message is ignored.

unload
displays the message or messages while a tape is loaded; if no tape is loaded, the message is ignored.

reload displays the first message while a tape is loaded and the second message when the tape is removed. If no tape is loaded, the first message is ignored and the second message is displayed immediately. The second message is displayed until the next tape is loaded.

noop is intended for test purposes only. It accesses the tape device but does not display the message or messages.

`-b or --blink`
causes *<message1>* to be displayed repeatedly for 2 seconds with a half-second pause in between.

`<message1>`
is the first or only message to be displayed. The message can be up to 8 byte.

`<message2>`
is a second message to be displayed alternately with the first, at 2-second intervals. The message can be up to 8 byte.

<node>
: is a device node of the target tape device

`-q or --quiet`
: suppresses all error messages.

`-h or --help`
: displays help text. To view the man page, enter **`man tape390_display`**.

`-v or --version`
: displays information about the version.

Note:

1. Symbols that can be displayed include:

   Alphabetic characters:
   A through Z (uppercase only) and spaces. Lowercase letters are converted to uppercase.

   Numeric characters:
   0 1 2 3 4 5 6 7 8 9

   Special characters:
   @ $ # , . / ' ( ) * & + - = % : _ < > ? ;

   The following are included in the 3490 hardware reference but might not display on all devices: | ¢
2. If only one message is defined, it remains displayed until the tape device driver next starts to move or the message is updated.
3. If the messages contain spaces or shell-sensitive characters, they must be enclosed in quotation marks.

Examples

The following examples assume that you are using standard devices nodes and not device nodes that are created by udev:

- Alternately display “BACKUP” and “COMPLETE” at 2-second intervals until device `/dev/ntibm0` processes the next tape movement command:

  ```
  tape390_display BACKUP COMPLETE /dev/ntibm0
  ```
- Display the message “REM TAPE” while a tape is in the physical tape device followed by the message“NEW TAPE” until a new tape is loaded:

  ```
  tape390_display --type reload "REM TAPE" "NEW TAPE" /dev/ntibm0
  ```
- Attempts to unload the tape and load a new tape automatically, the messages are the same as in the previous example:

  ```
  tape390_display -l -t reload "REM TAPE" "NEW TAPE" /dev/ntibm0
  ```

tunedasd - Adjust low-level DASD settings

Use the **tunedasd** command to adjust performance relevant settings and other low-level DASD device settings.

In particular, you can perform these tasks:

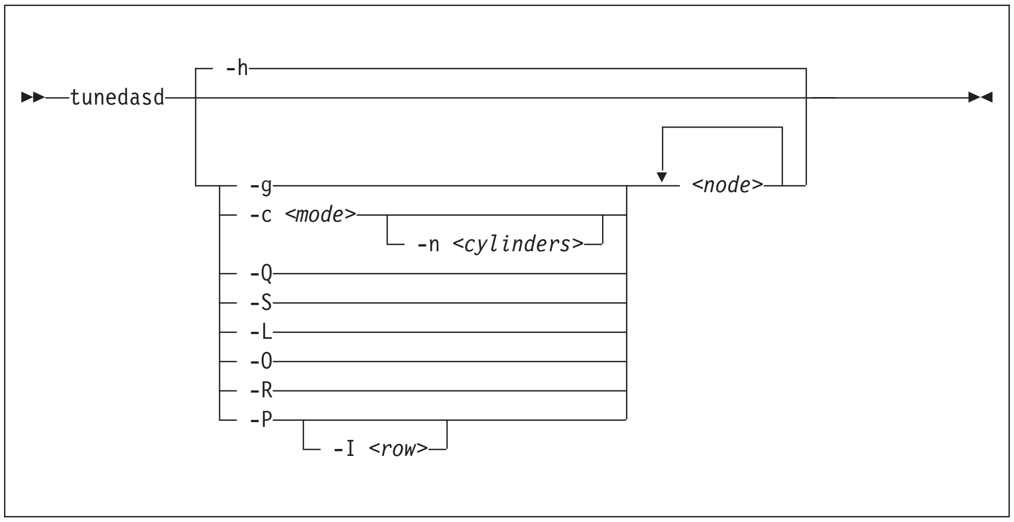
- Query and set a DASD's cache mode
- Display and reset DASD performance statistics

  Tip: Use the **dasdstat** command to display performance statistics. This command includes and extends the statistics that are available through the **tunedasd** command.
- Reserve and release DASD
- Break the lock of an online DASD (to learn how to access a boxed DASD that is not yet online, see "Accessing DASD by force" on page 164)

Before you begin: For the performance statistics:

- Your kernel needs to be compiled with the kernel configuration option CONFIG_DASD_PROFILE (see "Building a kernel with the DASD device driver" on page 155).
- Data gathering must be turned on by writing "on" to `/proc/dasd/statistics`.

tunedasd syntax



Where:

<node>
: specifies a device node for the DASD to which the command is to be applied.

-g or --get_cache
: gets the current caching mode of the storage controller. This option applies to ECKD only.

-c *<mode>* **or --cache** *<mode>*
: Sets the caching mode on the storage controller to *<mode>*. This option applies to ECKD only.

  Today's ECKD devices support the following behaviors:

normal
: for normal cache replacement.

bypass
: to bypass cache.

inhibit
: to inhibit cache.

sequential
: for sequential access.

prestage
: for sequential prestage.

record
: for record access.

For details, see *IBM TotalStorage Enterprise Storage Server® System/390 Command Reference 2105 Models E10, E20, F10, and F20*, SC26-7295.

`-n` *<cylinders>* **or `--no_cyl`** *<cylinders>*
specifies the number of cylinders to be cached. This option applies to ECKD only.

`-Q or --query_reserve`
queries the reserve status of the device. The status can be:

none
: the device is not reserved.

implicit
: the device is not reserved, but there is a contingent or implicit allegiance to this Linux instance.

other
: the device is reserved to another operating system instance.

reserved
: the device is reserved to this Linux instance.

For details, see the "Storage Control Reference" of the attached storage server.

This option applies to ECKD only.

`-S or --reserve`
reserves the device. This option applies to ECKD only.

`-L or --release`
releases the device. This option applies to ECKD only.

`-O or --slock`
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`
resets the profile information of the device.

`-P or --profile`
displays a usage profile of the device.

`-I` *<row>* **or `--prof_item`** *<row>*
displays the usage profile item that is specified by <*row*>. <*row*> can be one of:

reqs	number of DASD I/O requests.
sects	number of 512-byte sectors.
sizes	histogram of sizes.
total	histogram of I/O times.
totsect	histogram of I/O times per sector.
start	histogram of I/O time until ssch.
irq	histogram of I/O time between ssch and irq.

irqsect
histogram of I/O time between ssch and irq per sector.

end histogram of I/O time between irq and end.

queue number of requests in the DASD internal request queue at enqueueing.

-v or --version
displays version information.

-h or --help
displays help information. To view the man page, enter **man tunedasd**.

Examples

- The following sequence of commands first checks the reservation status of a DASD and then reserves it:

```
# tunedasd -Q /dev/dasdzzz
none
# tunedasd -S /dev/dasdzzz
Reserving device </dev/dasdzzz>...
Done.
# tunedasd -Q /dev/dasdzzz
reserved
```

- This example first queries the current setting for the cache mode of a DASD with device node /dev/dasdzzz and then sets it to one cylinder "prestage".

```
# tunedasd -g /dev/dasdzzz
normal (0 cyl)
# tunedasd -c prestage -n 2 /dev/dasdzzz
Setting cache mode for device </devdasdzzz>...
Done.
# tunedasd -g /dev/dasdzzz
prestage (2 cyl)
```

- In this example two device nodes are specified. The output is printed for each node in the order in which the nodes where specified.

```
# tunedasd -g /dev/dasdzzz /dev/dasdzzy
prestage (2 cyl)
normal (0 cyl)
```

- The following command displays the usage profile of a DASD.

```
# tunedasd -P /dev/dasdzzz

19617 dasd I/O requests
with 4841336 sectors(512B each)

   __<4    ___8    __16    __32    __64    _128    _256    _512    __1k    __2k    __4k    __8k    _16k    _32k    _64k    128k
   _256    _512    __1M    __2M    __4M    __8M    _16M    _32M    _64M    128M    256M    512M    __1G    __2G    __4G    _>4G
Histogram of sizes (512B secs)
      0       0     441      77      78      87     188   18746       0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O times (microseconds)
      0       0       0       0       0       0       0       0     235     150     297   18683     241       3       4       4
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O times per sector
      0       0       0   18736     333     278      94      78      97       1       0       0       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O time till ssch
  19234      40      32       0       2       0       0       3      40      53     128      85       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O time between ssch and irq
      0       0       0       0       0       0       0       0     387     208     250   18538     223       3       4       4
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O time between ssch and irq per sector
      0       0       0   18803     326     398      70      19       1       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
Histogram of I/O time between irq and end
  18520     735     246      68      43       4       1       0       0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
# of req in chanq at enqueuing (1..32)
      0   19308     123      30      25     130       0       0       0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
```

- The following command displays a row of the usage profile of a DASD. The output is on a single line as indicated by the (cont...) (... cont) in the illustration:

```
# tunedasd -P -I irq /dev/dasdzzz
         0|         0|         0|         0|         0|         0|         0|         0|       503|       271|(cont...)
 (... cont)       267|     18544|       224|         3|         4|         4|         0|         0|         0|(cont...)
 (... cont)         0|         0|         0|         0|         0|         0|         0|         0|         0|(cont...)
 (... cont)         0|         0|         0|         0|
```

vmcp - Send CP commands to the z/VM hypervisor

Use the **vmcp** command to send control program (CP) commands to the z/VM hypervisor and display the response from z/VM.

The **vmcp** command expects the command line as a parameter and returns the response to stdout. Error messages are written to stderr.

You can issue CP commands through the /dev/vmcp device node (see Chapter 39, “z/VM CP interface device driver,” on page 467) or with the **vmcp** command. In both cases, the vmcp module must be compiled into the kernel.

vmcp syntax

```
                         ┌─ 8 KB─┐
►►──vmcp──┬──────┬───────┼───────┼── <command>──────────────────────────────►◄
          └─ -k──┘       └─ -b───┘
```

Where:

-k or --keepcase
: preserves the case of the characters in the specified command string. By default, the command string is converted to uppercase characters.

-b *<size>* **or --buffer** *<size>*
: specifies the buffer size in bytes for the response from z/VM CP. Valid values are from 4096 (or 4k) up to 1048756 (or 1M). By default, **vmcp** allocates an 8192 byte (8k) buffer. You can use k and M to specify kilo- and megabytes.

<command>
: specifies the command that you want to send to CP.

-h or --help
: displays help information. To view the man page, enter **man vmcp**.

-v or --version
: displays version information.

If the command completes successfully, **vmcp** returns 0. Otherwise, **vmcp** returns one of the following values:

1. CP returned a non-zero response code.
2. The specified buffer was not large enough to hold CP's response. The command was run, but the response was truncated. You can use the **--buffer** option to increase the response buffer.
3. Linux reported an error to **vmcp**. See the error message for details.
4. The options that are passed to **vmcp** were erroneous. See the error messages for details.

Examples

- To get your user ID issue:

```
# vmcp query userid
```

- To attach the device 1234 to your guest, issue:

```
# vmcp attach 1234 \*
```

- If you add the following line to `/etc/sudoers`:

  ```
  ALL ALL=NOPASSWD:/sbin/vmcp indicate
  ```

  every user on the system can run the indicate command by using:

  ```
  # sudo vmcp indicate
  ```

- If you need a larger response buffer, use the **`--buffer`** option:

  ```
  # vmcp --buffer=128k q 1-ffff
  ```

vmur - Work with z/VM spool file queues

Use the **vmur** command to work with z/VM spool file queues.

In particular, the **vmur** command provides these functions:

Receive
Read data from the z/VM reader file queue. The command performs the following steps:

- Places the reader queue file to be received at the top of the queue.
- Changes the reader queue file attribute to NOHOLD.
- Closes the z/VM reader after 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

displays detailed information about one or all files on the specified spool file queue.

Purge removes one or all files on the specified spool file queue.

Order positions 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 to prevent unpredictable results or destructive conflicts.

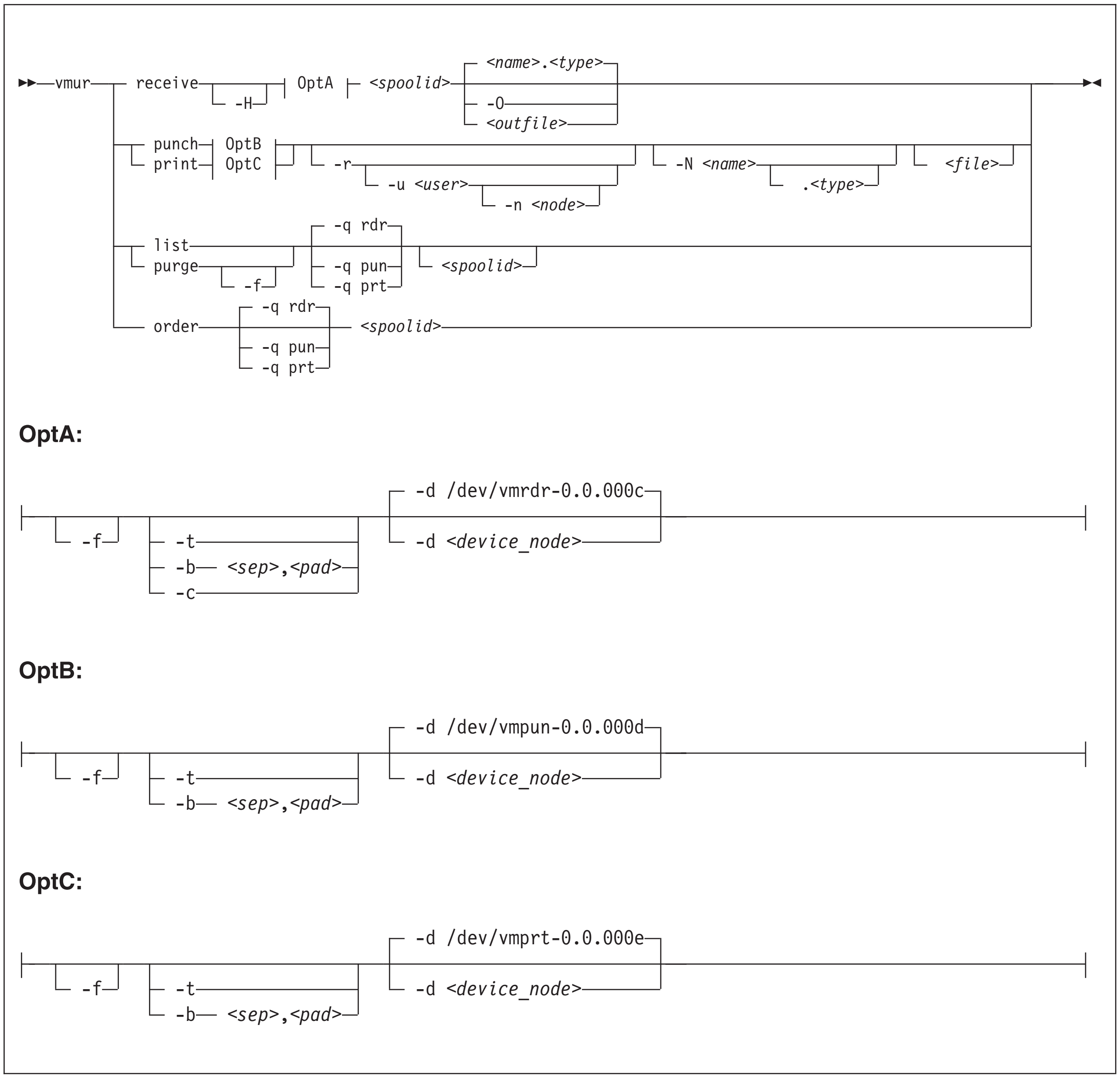
For example, this serialization prevents a process from issuing **vmur purge -f** while another process is executing **vmur receive 1234**. However, **vmur** is not serialized against concurrent CP commands that are issued through **vmcp**: if one process is executing **vmur receive 1234** and another process issues **vmcp purge rdr 1234**, then the received file might be incomplete. To avoid such unwanted effects, use **vmur** exclusively when you work 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 begin: To use the receive, punch, and print functions, the vmur device driver must be loaded and the corresponding unit record devices must be set online.

vmur syntax



Where:

re or receive
specifies that a file on the z/VM reader queue is to be received.

pun or punch
specifies that a file is to be written to the z/VM punch queue.

li or list
specifies that information about 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 that are 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 that requires 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 that is 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 for punch or 132 for print, 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 must be received or written by using the blocked mode. As parameter for the **-b** option, specify the hex codes of the separator and the padding character. Example:

```
--blocked 0xSS,0xPP
```

Use this option if you need to use character sets other than IBM037 and ISO-8859-1 for conversion.
- For the receive function: All trailing padding characters are removed from the end of each record that is read from the virtual reader and the separator character is inserted afterward. The receive function's output can be piped to **iconv** by using the appropriate character sets. Example:

```
# vmur rec 7 -b 0x25,0x40 -O | iconv -f EBCDIC-US -t ISO-8859-1 > myfile
```

- For the punch or print function: The separator is used to identify the line end character of the file to punch or print. If a line has fewer characters than the record length of the used unit record device, the residual of the record is filled up with the specified padding byte. If a line exceeds the record size, an error is printed. Example:

```
# iconv test.txt -f ISO-8859-1 -t EBCDIC-US | vmur pun -b 0x25,0x40 -N test
```

-c or --convert
converts the VMDUMP spool file into a format appropriate for further analysis with crash.

-r or --rdr
specifies that the punch or print file is to be transferred to a reader.

-u *<user>* **or --user** *<user>*
specifies the z/VM user ID to whose reader the data is to be transferred. If user is omitted, the data is transferred to your own machine's reader. The user option is only valid if the **-r** option was specified.

-n *<node>* **or --node** *<node>*
specifies the z/VM node of the z/VM system to which the data is to be transferred. Remote Spooling Communications Subsystem (RSCS) must be installed on the z/VM systems and the specified node must be defined in the RSCS machine's configuration file. 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 was specified.

-f or --force
suppresses confirmation messages.
- For the receive function: specifies that *<outfile>* is to be overwritten without displaying any confirmation message.
- For the purge function: specifies that the spool files specified are to be purged without displaying any confirmation message.
- For the punch or print option: convert Linux input file name to valid spool file name automatically without any error message.

-O or --stdout
specifies that the reader file's contents are written to standard output.

-N or --name
specifies a name and, optionally, a type for the z/VM spool file to be created by the punch or print option. To specify a type after the file name, enter a period followed by the type. For example:

```
# vmur pun -r /boot/parmfile -N myname.mytype
```

Both the name and the type must comply to z/VM file name rules (that is, must be one to eight characters long).

If omitted, the Linux input file name (if any) is used instead. Use the **--force** option to enforce valid spool file names and types.

-H or --hold
specifies that the spool file to be received remains in the reader queue. If omitted, the spool file is purged.

<spoolid>
denotes the spool ID that identifies a file that belongs 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 in the 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, the name and type of the spool file to be received (see the NAME and TYPE columns in **vmur list** output) are used

to build the output file <*name*>.<*type*>. If the spool file to be received is an unnamed file, an error message is issued.

<*file*>
specifies the file data to be punched or printed. If file is omitted, the data is read from standard input.

-h or --help
displays help information for the command. To view the man page, enter **man vmur**.

-v or --version
displays version information.

Examples

These examples illustrate common scenarios for unit record devices.

In all examples the following device nodes are used:
- /dev/vmrdr-0.0.000c as virtual reader.
- /dev/vmpun-0.0.000d as virtual punch.

Besides the vmur device driver and the **vmur** command, these scenarios require that:
- The vmcp module is loaded.
- The **vmcp** and **vmconvert** commands from the s390-tools package are available.

Creating and reading a guest memory dump

You can use the **vmur** command to read a guest memory dump that was created, for example, with the **vmcp** command.

Procedure

1. Produce a memory dump of the z/VM guest virtual machine memory:
```
# vmcp vmdump
```
Depending on the memory size this command might take some time to complete.
2. List the spool files for the reader to find the spool ID of the dump file, VMDUMP. In the example, the spool ID of VMDUMP is 463.
```
# vmur li

ORIGINID FILE CLASS RECORDS  CPY HOLD DATE  TIME     NAME   TYPE DIST
T6360025 0463 V DMP 00020222 001 NONE 06/11 15:07:42 VMDUMP FILE T6360025
```
3. Read and convert the VMDUMP spool file to a file in the current working directory of the Linux file system:
```
# vmur rec 463 -c linux_dump
```

Using FTP to receive and convert a dump file:

Use the --convert option with the --stdout option to receive a VMDUMP spool file straight from the z/VM reader queue, convert it, and send it to another host with FTP.

Procedure

1. Establish an FTP session with the target host and log in.
2. Enter the FTP command **binary**.
3. Enter the FTP command:

```
put |"vmur re <spoolid> -c -O" <filename_on_target_host>
```

Logging and reading the z/VM guest virtual machine console

You can use the **vmur** command to read a console transcript that was spooled, for example, with the **vmcp** command.

Procedure

1. Begin console spooling:

```
# vmcp sp cons start
```

2. Produce output to the z/VM console. Use, for example, CP TRACE.
3. Stop console spooling, close the file with the console output, and transfer the file to the reader queue. In the resulting CP message, the spool ID follows the FILE keyword. In the example, the spool ID is 398:

```
# vmcp sp cons stop close \* rdr

RDR FILE 0398 SENT FROM T6360025 CON WAS 0398 RECS 1872 CPY 001 T NOHOLD NOKEEP
```

4. Read the file with the console output into a file in the current working directory on the Linux file system:

```
# vmur re -t 398 linux_cons
```

Preparing the z/VM reader as an IPL device for Linux

You can use the **vmur** command to transfer all files for booting Linux to the z/VM reader. You can also arrange the files such that the reader can be used as an IPL device.

Procedure

1. Send the kernel parameter file, `parmfile`, to the z/VM punch device and transfer the file to the reader queue. The resulting message shows the spool ID of the parameter file.

```
# vmur pun -r /boot/parmfile

Reader file with spoolid 0465 created.
```

2. Send the kernel image file to the z/VM punch device and transfer the file to the reader queue. The resulting message shows the spool ID of the kernel image file.

```
# vmur pun -r /boot/vmlinuz -N image

Reader file with spoolid 0466 created.
```

3. Optional: Check the spool IDs of `image` and `parmfile` in the reader queue. In this example, the spool ID of `parmfile` is 465 and the spool ID of `image` is 466.

```
# vmur li

ORIGINID FILE CLASS RECORDS  CPY HOLD DATE  TIME     NAME      TYPE     DIST
T6360025 0463 V DMP 00020222 001 NONE 06/11 15:07:42 VMDUMP    FILE     T6360025
T6360025 0465 A PUN 00000002 001 NONE 06/11 15:30:31 parmfile           T6360025
T6360025 0466 A PUN 00065200 001 NONE 06/11 15:30:52 image              T6360025
```

4. Move `image` to the first and `parmfile` to the second position in the reader queue:

```
# vmur or 465
# vmur or 466
```

5. Configure the z/VM reader as the re-IPL device:

```
# echo 0.0.000c > /sys/firmware/reipl/ccw/device
```

6. Boot Linux from the z/VM reader:

```
# reboot
```

Sending a file to different z/VM guest virtual machines

You can use the **vmur** command to send files to other z/VM guest virtual machines.

About this task

This scenario describes how to send a file called `lnxprofile.exec` from the file system of an instance of Linux on z/VM to other z/VM guest virtual machines.

For example, `lnxprofile.exec` could contain the content of a PROFILE EXEC file with CP and CMS commands to customize z/VM guest virtual machines for running Linux.

Procedure

1. Send `lnxprofile.exec` to two z/VM guest virtual machines: z/VM user ID t2930020 at node boet2930 and z/VM user ID t6360025 at node boet6360.

```
vmur pun lnxprofile.exec -t -r -u t2930020 -n boet2930 -N PROFILE
vmur pun lnxprofile.exec -t -r -u t6360025 -n boet6360 -N PROFILE
```

2. Log on to t2930020 at boet2930, IPL CMS, and issue the CP command:

```
QUERY RDR ALL
```

   The command output shows the spool ID of PROFILE in the FILE column.

3. Issue the CMS command:

```
RECEIVE <spoolid> PROFILE EXEC A (REPL
```

   In the command, *<spoolid>* is the spool ID of PROFILE found in step 2.

4. Repeat steps 2 and 3 for t6360025 at boet6360.

Sending a file to a z/VSE instance

You can use the **vmur** command to send files to a z/VSE instance.

Procedure

To send `lserv.job` to user ID vseuser at node vse01sys, issue:

```
vmur pun lserv.job -t -r -u vseuser -n vse01sys -N LSERV
```

zdsfs - Mount a z/OS DASD

Use the **zdsfs** command to mount z/OS DASDs as a Linux file system.

The zdsfs file system translates the z/OS data sets, which are stored on the DASDs in records of arbitrary or even variable size, into Linux semantics.

Through the zdsfs file system, applications on Linux can read z/OS physical sequential data sets (PS) and partitioned data sets (PDS) on the DASD. In the Linux file system, physical sequential data sets are represented as files. Partitioned data sets are represented as directories that contain the PDS members as files. Other z/OS data set formats, such as extended format data sets or VSAM data sets, are not supported. zdsfs is optimized for sequential read access.

Attention:

- To avoid data inconsistencies, set the DASDs offline in z/OS before you mount them in Linux.
- Through the zdsfs file system, the whole DASDs are accessible to Linux, but the access is not controlled by z/OS auditing mechanisms.

  To avoid security problems, you might want to dedicate the z/OS DASDs only for providing data for Linux.

Per default, only the Linux user who mounts the zdsfs file system has access to it.

Tip: If you want to grant a user group access to the zdsfs file system, mount it with the fuse options `default_permissions`, `allow_other`, and `gid`.

To unmount file systems that you mounted with **zdsfs**, you can use **fusermount**, whether root or non-root user. See the **fusermount** man page for details.

See *z/OS DFSMS Using Data Sets*, SC26-7410 for more information about z/OS data sets.

Before you begin:

- You need a kernel that is built with the common code option CONFIG_FUSE_FS to include FUSE support.
- FUSE support must be compiled into the kernel or the fuse module must be loaded, for example, with **modprobe fuse**.
- The FUSE library must be installed on your system. If the library is not included in your distribution, you can obtain it from sourceforge at sourceforge.net/projects/fuse.
- The raw-track access mode of the DASD must be enabled.

  Make sure that the DASD is set offline when you enable the raw-track access mode.

  See “Accessing full ECKD tracks” on page 176 for details.
- The DASD must be online.

  Tip: You can use the **chccwdev** command to enable the raw-track access mode and set the device online afterward in one step.

  Set the DASD offline in z/OS before you set it online in Linux.
- You must have the appropriate read permissions for the device node.

zdsfs syntax

```
►►──zdsfs──┬──────────────────┬─┬─────────────────┬─┬─ -l── <file-name>─┬──────►
           └─ <zdsfs-options>─┘ └─ <fuse-options>─┘ └─ <node-list>──────┘

►─ <mount-point>──────────────────────────────────────────────────────────────►◄
```

where:

<zdsfs-options>
: zdsfs-specific options.

-o ignore_incomplete
: represents all complete data sets in the file system, even if there are incomplete data sets. Incomplete data sets are not represented.

  In z/OS, data sets might be distributed over different DASDs. For each incomplete data set, a warning message is issued to the standard error stream. If there are incomplete data sets and this option is not specified, the **zdsfs** command returns with an error.

-o rdw
: keeps record descriptor words (RDWs) of data sets that are stored by using the z/OS concept of variable record lengths.

-o tracks=*<n>*
: specifies the track buffer size in tracks. The default is 128 tracks.

  zdsfs allocates a track buffer of <*n*>*120 KB for each open file to store and extract the user data. Increasing the track buffer size might improve your system performance.

-o seekbuffer=*<s>*
: sets the maximum seek history buffer size in bytes. The default is 1,048,576 B.

  zdsfs saves offset information about a data set in the seek history buffer to speed up the performance of a seek operation.

<fuse-options>
: options for FUSE. The following options are supported by the **zdsfs** command. To use an option, it must also be supported by the version of FUSE that is installed.

-d or -o debug
: enables debug output (implies **-f**).

-f runs the command as a foreground operation.

-o allow_other
: allows access to other users.

-o allow_root
: allows access to root.

-o nonempty
: allows mounts over files and non-empty directories.

-o default_permissions
: enables permission checking by the kernel.

-o max_read=<*n*>
sets maximum size of read requests.

-o kernel_cache
caches files in the kernel.

-o [no]auto_cache
enables or disables caching based on modification times.

-o umask=<*mask*>
sets file permissions (octal).

-o uid=<*n*>
sets the file owner.

-o gid=<*n*>
sets the file group.

-o max_write=<*n*>
sets the maximum size of write requests.

-o max_readahead=<*n*>
sets the maximum readahead value.

-o async_read
performs reads asynchronously (default).

-o sync_read
performs reads synchronously.

<*node-list*>
one or more device nodes for the DASDs, separated by blanks.

<*file-name*>
a file that contains a node list.

<*mount-point*>
the mount point in the Linux file system where you want to mount the z/OS data sets.

-h or --help
displays help information for the command. To view the man page, enter **man zdsfs**.

-v or --version
displays version information for the command.

File characteristics

There are two ways to handle the z/OS characteristics of a file:

- The file metadata.txt:

  The metadata.txt file is in the root directory of the mount point. It contains one row for each file or directory, where:

  dsn
  specifies
  - the name of the file in the form <*file-name*> for z/OS physical sequential data sets.
  - the name of the directory in the form <*directory-name*>, and the name of a file in that directory in the form <*directory-name*>*(*<*file-name*>*)* for z/OS partitioned data sets.

dsorg
: specifies the organization of the file. The organization is PO for a directory, and PS for a file.

lrecl
: specifies the record length of the file.

recfm
: specifies the z/OS record format of the file. Supported record formats are: V, F, U, B, S, A, and M.

Example:

```
dsn=FOOBAR.TESTF.TXT,recfm=FB,lrecl=80,dsorg=PS
dsn=FOOBAR.TESTVB.TXT,recfm=VB,lrecl=100,dsorg=PS
dsn=FOOBAR.PDSF.DAT,recfm=F,lrecl=80,dsorg=PO
dsn=FOOBAR.PDSF.DAT(TEST1),recfm=F,lrecl=80,dsorg=PS
dsn=FOOBAR.PDSF.DAT(TEST2),recfm=F,lrecl=80,dsorg=PS
dsn=FOOBAR.PDSF.DAT(TEXT3),recfm=F,lrecl=80,dsorg=PS
```

- Extended attributes:

  user.dsorg
  : specifies the organization of the file.

  user.lrecl
  : specifies the record length of the file.

  user.recfm
  : specifies the z/OS record format of the file.

  You can use the following system calls to work with extended attributes:

  listxattr
  : to list the current values of all extended attributes.

  getxattr
  : to read the current value of a particular extended attribute.

  You can use these system calls through the **getfattr** command. For more information, see the man pages of these commands and of the listxattr and getxattr system calls.

Examples

- Enable the raw-track access mode of DASD device 0.0.7000 and set the device online afterward:

```
# chccwdev -a raw_track_access=1 -e 0.0.7000
```

- Mount the partitioned data set on the DASDs represented by the file nodes `/dev/dasde` and `/dev/dasdf` at `/mnt`:

```
# zdsfs /dev/dasde /dev/dasdf /mnt
```

- As user “myuser”, mount the partitioned data set on the DASD represented by the file node `/dev/dasde` at `/home/myuser/mntzos`:
  - Access the mounted file system exclusively:

```
# zdsfs /dev/dasde /home/myuser/mntzos
```

  - Allow the root user to access the mounted file system:

```
# zdsfs -o allow_root /dev/dasde /home/myuser/mntzos
```

The **ls** command does not reflect these permissions. In both cases, it shows:

```
# ls -al /home/myuser/mntzos
total 121284
dr-xr-x--- 2 root   root          0 Dec  3 15:54 .
drwx------ 3 myuser myuser     4096 Dec  3 15:51 ..
-r--r----- 1 root   root    2833200 Jun 27  2012 EXPORT.BIN1.DAT
-r--r----- 1 root   root    2833200 Jun 27  2012 EXPORT.BIN2.DAT
-r--r----- 1 root   root    2833200 Jun 27  2012 EXPORT.BIN3.DAT
-r--r----- 1 root   root    2833200 Feb 14  2013 EXPORT.BIN4.DAT
dr-xr-x--- 2 root   root   13599360 Aug  9  2012 EXPORT.PDS1.DAT
dr-xr-x--- 2 root   root   13599360 Aug  9  2012 EXPORT.PDS2.DAT
dr-xr-x--- 2 root   root   13599360 Aug  9  2012 EXPORT.PDS3.DAT
dr-xr-x--- 2 root   root   55247400 Aug  9  2012 EXPORT.PDS4.DAT
-r--r----- 1 root   root        981 Dec  3 15:54 metadata.txt

$ ls -al /dev/dasde
brw-rw---- 1 root disk 94, 16 Dec  3 13:58 /dev/dasde
```

- As root user, mount the partitioned data set on the DASD represented by the file node `/dev/dasde` at `/mnt` on behalf of the user ID “myuser” (UID=1002), and permit the members of the group ID “zosimport” (GID=1002) file access:

```
# zdsfs /dev/dasde /mnt -o uid=1002,gid=1002,allow_other,default_permissions
```

The **ls** command indicates the owner “myuser” and the access right for group “zosimport”:

```
$ ls -al /mnt
total 121284
dr-xr-x---  2 myuser zosimport        0 Dec  3 14:22 .
drwxr-xr-x 23 root   root          4096 Dec  3 13:59 ..
-r--r-----  1 myuser zosimport      981 Dec  3 14:22 metadata.txt
-r--r-----  1 myuser zosimport  2833200 Jun 27  2012 EXPORT.BIN1.DAT
-r--r-----  1 myuser zosimport  2833200 Jun 27  2012 EXPORT.BIN2.DAT
-r--r-----  1 myuser zosimport  2833200 Feb 14  2013 EXPORT.BIN3.DAT
-r--r-----  1 myuser zosimport  2833200 Jun 27  2012 EXPORT.BIN4.DAT
dr-xr-x---  2 myuser zosimport 13599360 Aug  9  2012 EXPORT.PDS1.DAT
dr-xr-x---  2 myuser zosimport 13599360 Aug  9  2012 EXPORT.PDS2.DAT
dr-xr-x---  2 myuser zosimport 55247400 Aug  9  2012 EXPORT.PDS3.DAT
dr-xr-x---  2 myuser zosimport 13599360 Aug  9  2012 EXPORT.PDS4.DAT
```

- Unmount the partitioned data set that is mounted at `/mnt`:

```
# fusermount -u /mnt
```

- Show the extended attributes of a file, `FB.XMP.TXT`, on a z/OS DASD that is mounted on `/mnt`:

```
# getfattr -d /mnt/FB.XMP.TXT
```

- Show the extended attributes of all files on a z/OS DASD that is mounted on `/mnt`:

```
# cat /mnt/metadata.txt
```

znetconf - List and configure network devices

Use the **znetconf** command to list, configure, add, and remove network devices.

The **znetconf** command:
- Lists potential network devices.
- Lists configured network devices.
- Automatically configures and adds network devices.
- Removes network devices.

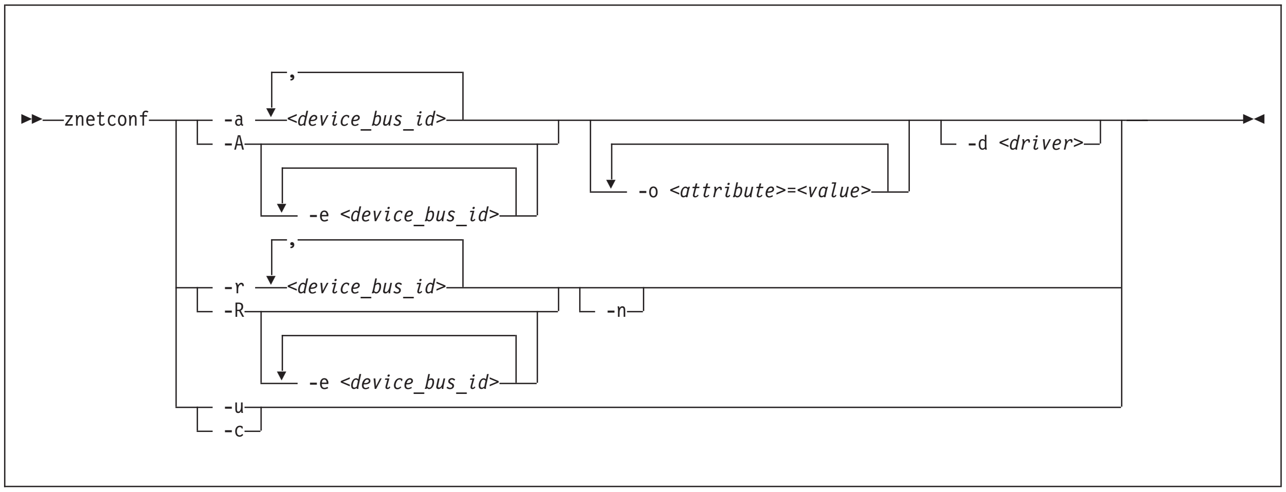
For automatic configuration, **znetconf** first builds a channel command word (CCW) group device from sensed CCW devices. It then configures any specified option through the sensed network device driver and sets the new network device online.

During automatic removal, **znetconf** sets the device offline and removes it.

Attention: Removing all network devices might lead to complete loss of network connectivity. Unless you can access your Linux instance from a terminal server on z/VM (see *How to Set up a Terminal Server Environment on z/VM*, SC34-2596), you might require the HMC or a 3270 terminal session to restore the connectivity.

Before you begin: The qeth, ctcm, or lcs device drivers must be part of the compiled kernel or loaded as modules. If needed, the **znetconf** command attempts to load the particular device driver.

znetconf syntax



Where:

-a or --add
: configures the network device with the specified device bus-ID. If you specify only one bus ID, the command automatically identifies the remaining bus IDs of the group device. You can enter a list of device bus-IDs that are separated by commas. The **znetconf** command does not check the validity of the combination of device bus-IDs.

<device_bus_id>
: specifies the device bus-ID of the CCW devices that constitute the network

device. If a device bus-ID begins with "0.0.", you can abbreviate it to the final hexadecimal digits. For example, you can abbreviate 0.0.f503 to f503.

-A or --add-all
configures all potential network devices. After you run **znetconf -A**, enter **znetconf -c** to see which devices were configured. You can also enter **znetconf -u** to display devices that were not configured.

-e or --except
omits the specified devices when configuring all potential network devices or removing all configured network devices.

-o or --option *<attribute>=<value>*
configures devices with the specified sysfs option.

-d or --driver *<driver name>*
configures devices with the specified device driver. Valid values are qeth, lcs, ctc, or ctcm.

-n or --non-interactive
answers all confirmation questions with "Yes".

-r or --remove
removes the network device with the specified device bus-ID. You can enter a list of device bus-IDs that are separated by a comma. You can remove only configured devices as listed by **znetconf -c**.

-R or --remove-all
removes all configured network devices. After successfully running this command, all devices that are listed by **znetconf -c** become potential devices that are listed by **znetconf -u**.

-u or --unconfigured
lists all network devices that are not yet configured.

-c or --configured
lists all configured network devices.

-h or --help
displays help information for the command. To view the man page, enter **man znetconf**.

-v or --version
displays version information.

If the command completes successfully, **znetconf** returns 0. Otherwise, 1 is returned.

Examples

- To list all potential network devices:

```
# znetconf -u
Device IDs                 Type    Card Type  CHPID Drv.
--------------------------------------------------------
0.0.f500,0.0.f501,0.0.f502 1731/01 OSA (QDIO) 00    qeth
0.0.f503,0.0.f504,0.0.f505 1731/01 OSA (QDIO) 01    qeth
```

- To configure device 0.0.f503:

```
znetconf -a 0.0.f503
```

or

```
znetconf -a f503
```

- To configure the potential network device 0.0.f500 with the layer2 option with the value 0 and the `portname` option with the value `myname`:

```
znetconf -a f500 -o layer2=0 -o portname=myname
```

- To list configured network devices:

```
znetconf -c
Device IDs                 Type    Card Type     CHPID Drv. Name State
-----------------------------------------------------------------------
0.0.f500,0.0.f501,0.0.f502 1731/01 Virt.NIC QDIO 00    qeth eth2 online
0.0.f503,0.0.f504,0.0.f505 1731/01 Virt.NIC QDIO 01    qeth eth1 online
0.0.f5f0,0.0.f5f1,0.0.f5f2 1731/01 OSD_1000      76    qeth eth0 online
```

- To remove network device 0.0.f503:

```
znetconf -r 0.0.f503
```

or

```
znetconf -r f503
```

- To remove all configured network devices except the devices with bus IDs 0.0.f500 and 0.0.f5f0:

```
znetconf -R -e 0.0.f500 -e 0.0.f5f0
```

- To configure all potential network devices except the device with bus ID 0.0.f503:

```
znetconf -A -e 0.0.f503
```

Chapter 53. Selected kernel parameters

You can use kernel parameters that are beyond the scope of an individual device driver or feature to configure Linux in general.

Kernel parameters that are specific to a particular device driver or feature are described in the setup section of the respective device driver or feature.

See Chapter 3, “Kernel and module parameters,” on page 25 for information about specifying kernel parameters.

cio_ignore - List devices to be ignored

Use the `cio_ignore=` kernel parameter to list specifications for I/O devices that are to be ignored.

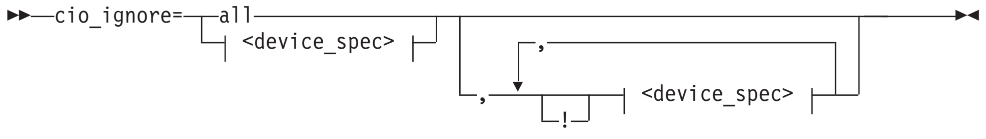
When a Linux on System z instance boots, it senses and analyzes all available I/O devices. You can use the `cio_ignore=` kernel parameter to list specifications for devices that are to be ignored. This exclusion list can cover all possible devices, even devices that do not actually exist. The following applies to ignored devices:

- Ignored devices are not sensed and analyzed. The device cannot be used until it is analyzed.
- Ignored devices are not represented in sysfs.
- Ignored devices do not occupy storage in the kernel.
- The subchannel to which an ignored device is attached is treated as if no device were attached.
- For Linux on z/VM, cio_ignore might hide essential devices such as the console. The console is typically device number 0.0.0009.

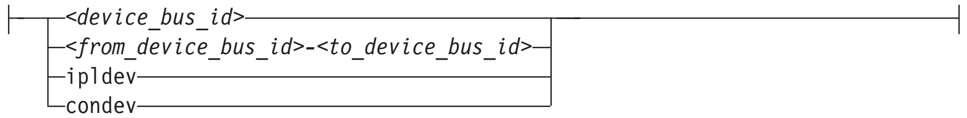
See also "Changing the exclusion list" on page 705.

Format

cio_ignore syntax



<device_spec>:



Where:

`all`
: states that all devices are to be ignored.

`<device_bus_id>`
: specifies a device. Device bus-IDs are of the form 0.<*n*>.<*devno*>, where <*n*> is a subchannel set ID and <*devno*> is a device number.

`<from_device_bus_id>-<to_device_bus_id>`
: are two device bus-IDs that specify the first and the last device in a range of devices.

`ipldev`
: specifies the IPL device. Use this keyword with the ! operator to avoid ignoring the IPL device.

condev
specifies the CCW console. Use this keyword with the ! operator to avoid ignoring the console device.

! makes the following term an exclusion statement. This operator is used to exclude individual devices or ranges of devices from a preceding more general specification of devices.

Examples

- This example specifies that all devices in the range 0.0.b100 through 0.0.b1ff, and the device 0.0.a100 are to be ignored.
```
cio_ignore=0.0.b100-0.0.b1ff,0.0.a100
```
- This example specifies that all devices except the console are to be ignored.
```
cio_ignore=all,!condev
```
- This example specifies that all devices but the range 0.0.b100 through 0.0.b1ff, and the device 0.0.a100 are to be ignored.
```
cio_ignore=all,!0.0.b100-0.0.b1ff,!0.0.a100
```
- This example specifies that all devices in the range 0.0.1000 through 0.0.1500 are to be ignored, except for devices in the range 0.0.1100 through 0.0.1120.
```
cio_ignore=0.0.1000-0.0.1500,!0.0.1100-0.0.1120
```
This is equivalent to the following specification:
```
cio_ignore=0.0.1000-0.0.10ff,0.0.1121-0.0.1500
```
- This example specifies that all devices in range 0.0.1000 through 0.0.1100 and all devices in range 0.1.7000 through 0.1.7010, plus device 0.0.1234 and device 0.1.4321 are to be ignored.
```
cio_ignore=0.0.1000-0.0.1100, 0.1.7000-0.1.7010, 0.0.1234, 0.1.4321
```

Changing the exclusion list

Use the **cio_ignore** command or the procfs interface to view or change the list of I/O device specifications that are ignored.

When a Linux on System z instance boots, it senses and analyzes all available I/O devices. You can use the cio_ignore kernel parameter to list specifications for devices that are to be ignored.

On a running Linux instance, you can view and change the exclusion list through a procfs interface or with the **cio_ignore** command (see “cio_ignore - Manage the I/O exclusion list” on page 558). This information describes the procfs interface.

After booting Linux you can display the exclusion list by issuing:
```
# cat /proc/cio_ignore
```

To add device specifications to the exclusion list issue a command of this form:
```
# echo add <device_list> > /proc/cio_ignore
```

When you add specifications for a device that is already sensed and analyzed, there is no immediate effect of adding it to the exclusion list. For example, the device still appears in the output of the **lscss** command and can be set online.

However, if the device later becomes unavailable, it is ignored when it reappears. For example, if the device is detached in z/VM it is ignored when it is attached again.

To make all devices that are in the exclusion list and that are currently offline unavailable to Linux issue a command of this form:

```
# echo purge > /proc/cio_ignore
```

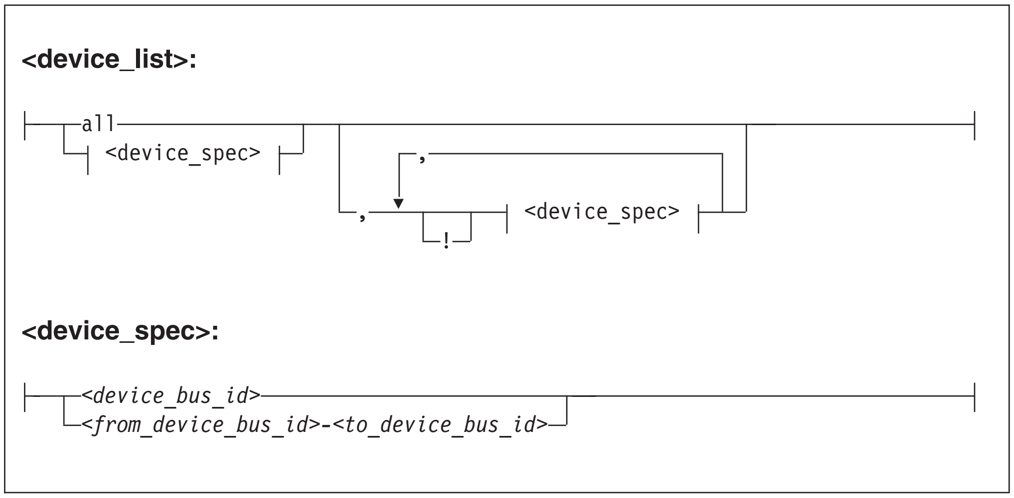
This command does not make devices unavailable if they are online.

To remove device specifications from the exclusion list issue a command of this form:

```
# echo free <device_list> > /proc/cio_ignore
```

When you remove device specifications from the exclusion list, the corresponding devices are sensed and analyzed if they exist. Where possible, the respective device driver is informed, and the devices become available to Linux.

In these commands, *<device_list>* follows this syntax:



Where the keywords and variables have the same meaning as in “Format” on page 704.

Ensure device availability

After the echo command completes successfully, some time might elapse until the freed device becomes available to Linux. Issue the following command to ensure that the device is ready to be used:

```
# echo 1 > /proc/cio_settle
```

This command returns after all required sysfs structures for the newly available device are completed. The **cio_ignore** command (see “cio_ignore - Manage the I/O exclusion list” on page 558) also returns after any new sysfs structures are completed. You do not need a separate **echo** command when using **cio_ignore** to remove devices from the exclusion list.

Results

The dynamically changed exclusion list is taken into account only when a device in this list is newly made available to the system, for example after it is defined to the system. It does not have any effect on setting devices online or offline within Linux.

Examples

- This command removes all devices from the exclusion list.

```
# echo free all > /proc/cio_ignore
```

- This command adds all devices in the range 0.0.b100 through 0.0.b1ff and device 0.0.a100 to the exclusion list.

```
# echo add 0.0.b100-0.0.b1ff,0.0.a100 > /proc/cio_ignore
```

- This command lists the ranges of devices that are ignored by common I/O.

```
# cat /proc/cio_ignore
0.0.0000-0.0.a0ff
0.0.a101-0.0.b0ff
0.0.b200-0.0.ffff
```

- This command removes all devices in the range 0.0.b100 through 0.0.b1ff and device 0.0.a100 from the exclusion list.

```
# echo free 0.0.b100-0.0.b1ff,0.0.a100 > /proc/cio_ignore
```

- This command removes the device with bus ID 0.0.c104 from the exclusion list.

```
# echo free 0.0.c104 > /proc/cio_ignore
```

- This command adds the device with bus ID 0.0.c104 to the exclusion list.

```
# echo add 0.0.c104 > /proc/cio_ignore
```

- This command makes all devices that are in the exclusion list and that are currently offline unavailable to Linux.

```
# echo purge > /proc/cio_ignore
```

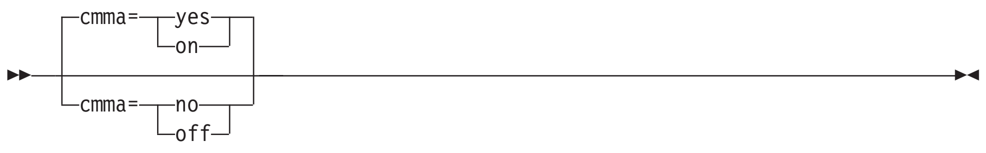
cmma - Reduce hypervisor paging I/O overhead

Use the `cmma=` kernel parameter to reduce hypervisor paging I/O overhead.

You can use Collaborative Memory Management Assist (CMMA, or "cmm2") on System z9 and later IBM mainframe systems. With this support, the z/VM control program and guest virtual machines can communicate attributes for specific 4K-byte blocks of guest memory. This exchange of information helps both the z/VM host and the guest virtual machines to optimize their use and management of memory.

Format

cmma syntax



Examples

This specification disables the CMMA support:

```
cmma=off
```

Alternatively, you can use the following specification to disable the CMMA support:

```
cmma=no
```

maxcpus - Restrict the number of CPUs Linux can use at IPL

Use the `maxcpus=` kernel parameter to restrict the number of CPUs that Linux can use at IPL.

For example, if there are four CPUs, specifying `maxcpus=2` causes the kernel to use only two CPUs. See also “possible_cpus - Limit the number of CPUs Linux can use” on page 712.

Format

maxcpus syntax

```
►►──maxcpus=<number>──────────────────────────────────────────────►◄
```

Examples

```
maxcpus=2
```

mem - Restricts memory usage

Use the mem= kernel parameter to restrict memory usage to the size specified.

You can use the K, M, or G suffix to specify the value in kilobyte, megabyte, or gigabyte.

To dump only the restricted memory, specify the size when using the **zipl** command (see Chapter 5, “Initial program loader for System z - zipl,” on page 65) to prepare the dump device.

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 - Bypass the initial ramdisk

The `noinitrd` statement is applicable to kernels that are compiled with initial RAM disk support. Use this kernel parameter to bypass 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 - Limit the number of CPUs Linux can use

Use the `possible_cpus=` kernel parameter to specify the maximum number of usable CPUs that Linux can add to the system. Alternatively, you can use the common code kernel parameter `nr_cpus`.

See also "maxcpus - Restrict the number of CPUs Linux can use at IPL" on page 709.

Format

possible_cpus syntax

```
►►──possible_cpus=<number>──────────────────────────────────────────────►◄
```

Examples

```
possible_cpus=8
```

ramdisk_size - Specify the ramdisk size

Use the `ramdisk_size=` kernel parameter to specify the size of the ramdisk in kilobytes.

Format

ramdisk_size syntax

```
►►──ramdisk_size=<size>──────────────────────────────────────────────►◄
```

Examples

```
ramdisk_size=32000
```

ro - Mount the root file system read-only

Use the `ro` kernel parameter to mount the root file system read-only.

Format

ro syntax

```
►►──ro────────────────────────────────────────────────────────────────────►◄
```

root - Specify the root device

Use the `root=` kernel parameter to tell Linux what to use as the root when mounting the root file system.

Format

root syntax

```
►►──root=<rootdevice>──────────────────────────────────────────────►◄
```

Examples

This example makes Linux use `/dev/dasda1` when mounting the root file system:

```
root=/dev/dasda1
```

vdso - Optimize system call performance

Use the `vdso=` kernel parameter to control the vdso support for the `gettimeofday`, `clock_getres`, and `clock_gettime` system calls.

The virtual dynamic shared object (vdso) support is a shared library that the kernel maps to all dynamically linked programs. The glibc detects the presence of the vdso and uses the functions that are provided in the library.

The vdso support is included in the Linux on System z kernel.

Format

vdso syntax

```
   ┌─vdso=──┬─1──┬─┐
   │        └─on─┘ │
►►─┼───────────────┼──────────────────────────────────────────────►◄
   └─vdso=──┬─0───┬┘
            └─off─┘
```

As the vdso library is mapped to all user-space processes, this change is visible in user space. In the unlikely event that a user-space program does not work with the vdso support, you can disable the support.

Examples

This example disables the vdso support:

```
vdso=0
```

vmhalt - Specify CP command to run after a system halt

Use the `vmhalt=` kernel parameter to specify a command to be issued to CP after a system halt.

This command applies only to Linux on z/VM.

Format

vmhalt syntax

```
►►──vmhalt=<COMMAND>──────────────────────────────────────────────────────►◄
```

Examples

This example specifies that an initial program load of CMS is to follow the Linux **`halt`** command:

```
vmhalt="CPU 00 CMD I CMS"
```

Note: The command must be entered in uppercase.

vmpanic - Specify CP command to run after a kernel panic

Use the `vmpanic=` kernel parameter to specify a command to be issued to CP after a kernel panic.

This command applies only to Linux on z/VM.

Note: Ensure that the **`dumpconf`** service is disabled when you use this kernel parameter. Otherwise, **`dumpconf`** will override the setting.

Format

vmpanic syntax

```
►►──vmpanic=<COMMAND>──────────────────────────────────────────────►◄
```

Examples

This example specifies that a VMDUMP is to follow a kernel panic:

```
vmpanic="VMDUMP"
```

Note: The command must be entered in uppercase.

vmpoff - Specify CP command to run after a power off

Use the `vmpoff=` kernel parameter to specify a command to be issued to CP after a system power off.

This command applies only to Linux on z/VM.

Format

vmpoff syntax

```
►►──vmpoff=<COMMAND>───────────────────────────────────────────────────►◄
```

Examples

This example specifies that CP is to clear the guest virtual machine after the Linux **`power off`** or **`halt -p`** command:

```
vmpoff="SYSTEM CLEAR"
```

Note: The command must be entered in uppercase.

vmreboot - Specify CP command to run on reboot

Use the vmreboot= kernel parameter to specify a command to be issued to CP on reboot.

This command applies only to Linux on z/VM.

Format

vmreboot syntax

```
►►──vmreboot=<COMMAND>─────────────────────────────────────────────►◄
```

Examples

This example specifies a message to be sent to the z/VM guest virtual machine OPERATOR if a reboot occurs:

```
vmreboot="MSG OPERATOR Reboot system"
```

Note: The command must be entered in uppercase.

Chapter 54. Linux diagnose code use

Linux on System z issues several diagnose instructions to the hypervisor (LPAR or z/VM).

Table 67 lists all diagnoses that 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** by using the `MODIFY diag` command in z/VM.

Table 67. Linux diagnoses

Number	Description	Linux use	Required/ Optional
0x008	z/VM CP command console interface	• The **vmcp** command • The 3215 and 3270 console drivers • The z/VM recording device driver (vmlogrdr) • smsgiucv	Required
0x010	Release pages	CMM	Required
0x014	Input spool file manipulation	The vmur device driver	Required
0x044	Voluntary time-slice end	In the kernel for spinlock and udelay	Required
0x064	Allows Linux to attach a DCSS	The DCSS block device driver (dcssblk), xip, and the MONITOR record device driver (monreader).	Required
0x09c	Voluntary time slice yield	Spinlock.	Optional
0x0dc	Monitor stream	The APPLDATA monitor record and the MONITOR stream application support (monwriter).	Required
0x204	LPAR Hypervisor data	The hypervisor file system (hypfs).	Required
0x210	Retrieve device information	• The common I/O layer • The DASD driver DIAG access method • DASD read-only query • The vmur device driver	Required
0x224	CPU type name table	The hypervisor file system (hypfs).	Required
0x250	Block I/O	The DASD driver DIAG access method.	Required
0x258	Page-reference services	In the kernel, for pfault.	Optional
0x288	Virtual machine time bomb	The watchdog device driver.	Required
0x2fc	Hypervisor cpu and memory accounting data	The hypervisor file system (hypfs).	Required
0x308	Re-ipl	Re-ipl and dump code.	Required

Required means that a function is not available without the diagnose; optional means that the function is available but there might be a performance impact.

Chapter 55. Kernel configuration menu options

There are numerous kernel configuration options that are specific to System z. Many of these options do not correspond to a particular device driver or feature.

Kernel builders: This information is intended for those who want to build their own kernel. Be aware that both compiling your own kernel or recompiling an existing distribution usually means that you have to maintain your kernel yourself.

The options that are described in this section are sorted into two groups:
- "General architecture-specific options" on page 724
- "Device driver-related options" on page 734

For each group, there is an overview of the options in the order in which you find them in the kernel configuration menu (see Figure 116 on page 724 and Figure 118 on page 734). Each overview is followed by an alphabetically sorted list of the options with a description.

Dependencies between options

Some kernel configuration options depend on one or more other options. Such options are hidden in the configuration menu unless the dependencies on other options are met.

Simple dependencies, where an option depends on another option that directly precedes it in the configuration menu, are shown in the overviews (Figure 116 on page 724 and Figure 118 on page 734). The dependent option is shown indented and graphically joined (└─) to the option it depends on. Options that have more complex dependencies are marked with an asterisk (*).

The option descriptions that follow the overviews include more detailed information about the dependencies. This information is provided in boolean format as it appears in the Kconfig files in the Linux source tree, with the CONFIG_ prefix omitted.

Common code options are not included in this summary. See the Linux source tree for descriptions of common code options. To locate the description of an option in the Linux source tree, open a command prompt and change the working directory to the root of the Linux source tree. Issue a command of this form:

```
# grep -rl --include='Kconfig' '^config <OPTION>' *
```

where <*option*> is the option of interest.

Note: In the Kconfig files, the options do not have the CONFIG_ prefix. Be sure to omit the prefix when searching for the option.

Example: To locate the Kconfig file with the description of the CONFIG_KMSG_IDS kernel configuration option, issue:

```
# grep -rl --include='Kconfig' '^config KMSG_IDS' *
arch/s390/Kconfig
```

General architecture-specific options

The general architecture-specific options comprise all System z specific options that are not in the Device Drivers main menu.

Figure 116 summarizes the general architecture-specific options in the order in which you find them in the kernel configuration menu. The pages that follow provide explanations for each option in alphabetical order. For device driver-specific options, see Figure 118 on page 734.

```
General setup --->
   ...
   Profiling support                                                      (CONFIG_PROFILING)
   OProfile system profiling                                              (CONFIG_OPROFILE)*
   ...
Processor type and features --->
   Processor type (choice)
     • System/390 model G5 and G6                                         (CONFIG_MARCH_G5)*
     • IBM zSeries model z800 and z900                                    (CONFIG_MARCH_Z900)
     • IBM zSeries model z890 and z990                                    (CONFIG_MARCH_Z990)
     • IBM System z9                                                      (CONFIG_MARCH_Z9_109)
     • IBM System z10                                                     (CONFIG_MARCH_Z10)
     • IBM zEnterprise 114 and 196                                        (CONFIG_MARCH_Z196)
     • IBM zBC12 and zEC12                                                (CONFIG_MARCH_ZEC12)
   Tune code generation (choice)
     • Default                                                            (CONFIG_TUNE_DEFAULT)
     • System/390 model G5 and G6                                         (CONFIG_TUNE_G5)
     • IBM zSeries model z800 and z900                                    (CONFIG_TUNE_Z900)
     • IBM zSeries model z890 and z990                                    (CONFIG_TUNE_Z990)
     • IBM System z9                                                      (CONFIG_TUNE_Z9_109)
     • IBM System z10                                                     (CONFIG_TUNE_Z10)
     • IBM zEnterprise 114 and 196                                        (CONFIG_TUNE_Z196)
     • IBM zBC12 and zEC12                                                (CONFIG_TUNE_ZEC12)
   64 bit kernel                                                          (CONFIG_64BIT)
   └─ Kernel support for 31 bit emulation                                 (CONFIG_COMPAT)
   Symmetric multi-processing support                                     (CONFIG_SMP)
   ├─ Maximum number of CPUs (2-256)                                      (CONFIG_NR_CPUS)
   ├─ Support for hot-pluggable CPUs                                      (CONFIG_HOTPLUG_CPU)
   └─ Book scheduler support                                              (CONFIG_SCHED_BOOK)
   ...
 Memory setup --->
   ...
   Pack kernel stack                                                      (CONFIG_PACK_STACK)
   Detect kernel stack overflow                                           (CONFIG_CHECK_STACK)
   └─ Size of the guard area (128-1024)                                   (CONFIG_STACK_GUARD)
   ...
I/O subsystem --->
   QDIO support                                                           (CONFIG_QDIO)
   PCI support --->                                                       (CONFIG_PCI)*
     Maximum number of PCI functions (1-4096)                            (CONFIG_PCI_NR_FUNCTIONS)
     Maximum number of MSI interrupts (64-32768)                         (CONFIG_PCI_NR_MSI)
       ...
   Support for CHSC subchannels                                           (CONFIG_CHSC_SCH)
   SCM bus driver                                                         (CONFIG_SCM_BUS)
   └─ Support for EADM subchannels                                        (CONFIG_EADM_SCH)
```

Figure 116. General architecture-specific kernel configuration menu options. The └ symbols indicate dependencies on preceding options. Options with more complex dependencies are marked with an asterisk (*).

```
Executable file formats / Emulations --->
   ...
   Enable seccomp to safely compute untrusted bytecode                  (CONFIG_SECCOMP)*
   ...
Networking support --->                             (common code option CONFIG_NET)
   ...
   Networking options --->
      ...
      IUCV support (S390 - z/VM only)                                   (CONFIG_IUCV)
      AF_IUCV Socket support (S390 - z/VM and HiperSockets transport)   (CONFIG_AFIUCV)*
   ...
Cryptographic API --->                              (common code option CONFIG_CRYPTO)
   ...
   Hardware crypto devices --->                     (common code option CONFIG_CRYPTO_HW)
      Support for PCI-attached cryptographic adapters                   (CONFIG_ZCRYPT)
      SHA1 digest algorithm                                             (CONFIG_CRYPTO_SHA1_S390)
      SHA256 digest algorithm                                           (CONFIG_CRYPTO_SHA256_S390)
      SHA384 and SHA512 digest algorithm                                (CONFIG_CRYPTO_SHA512_S390)
      DES and Triple DES cipher algorithms                              (CONFIG_CRYPTO_DES_S390)
      AES cipher algorithms                                             (CONFIG_CRYPTO_AES_S390)
      Pseudo random number generator device driver                      (CONFIG_S390_PRNG)
      GHASH digest algorithm                                            (CONFIG_CRYPTO_GHASH_S390)
      ...
Virtualization --->
   Pseudo page fault support                                            (CONFIG_PFAULT)
   VM shared kernel support                                             (CONFIG_SHARED_KERNEL)*
   Cooperative memory management                                        (CONFIG_CMM)
   └─ IUCV special message interface to cooperative memory management   (CONFIG_CMM_IUCV)*
   Linux - VM Monitor Stream, base infrastructure                       (CONFIG_APPLDATA_BASE)*
   ├─ Monitor memory management statistics                              (CONFIG_APPLDATA_MEM)*
   ├─ Monitor OS statistics                                             (CONFIG_APPLDATA_OS)
   └─ Monitor overall network statistics                                (CONFIG_APPLDATA_NET_SUM)
   s390 hypervisor file system support                                  (CONFIG_S390_HYPFS_FS)
   ...
Kernel message numbers                                                  (CONFIG_KMSG_IDS)
```

Figure 117. General architecture-specific kernel configuration menu options 2 of 2. The └─ symbols indicate dependencies on preceding options. Options with more complex dependencies are marked with an asterisk (*).

The following alphabetically sorted list has details about the general architecture-specific options in Figure 116 on page 724. For device driver-specific options, see "Device driver-related options" on page 734.

CONFIG_64BIT

Select this option if you have an IBM z/Architecture machine and want to use the 64 bit addressing mode.

CONFIG_AFIUCV

Select this option if you want to use AF_IUCV socket applications based on z/VM inter-user communication vehicle or based on HiperSockets.

Depends on QETH_L3 || IUCV.

CONFIG_APPLDATA_BASE

This provides a kernel interface for creating and updating z/VM APPLDATA monitor records. The monitor records are updated at certain time intervals, once the timer is started. Writing 1 or 0 to /proc/appldata/timer starts(1) or stops(0) the timer, i.e. enables or disables monitoring on the Linux side. A custom interval value (in seconds) can be written to /proc/appldata/interval.

Defaults are 60 seconds interval and timer off. The /proc entries can also be read from, showing the current settings.

Depends on the common code option PROC_FS.

CONFIG_APPLDATA_MEM

This provides memory management related data to the Linux - VM Monitor Stream, like paging/swapping rate, memory utilisation, etc. Writing 1 or 0 to /proc/appldata/memory creates(1) or removes(0) a z/VM APPLDATA monitor record, i.e. enables or disables monitoring this record on the z/VM side.

Default is disabled. The /proc entry can also be read from, showing the current settings.

This can also be compiled as a module, which will be called appldata_mem.o.

Depends on APPLDATA_BASE && VM_EVENT_COUNTERS.

VM_EVENT_COUNTERS is a common code option.

CONFIG_APPLDATA_NET_SUM

This provides network related data to the Linux - VM Monitor Stream, currently there is only a total sum of network I/O statistics, no per-interface data. Writing 1 or 0 to /proc/appldata/net_sum creates(1) or removes(0) a z/VM APPLDATA monitor record, i.e. enables or disables monitoring this record on the z/VM side.

Default is disabled. This can also be compiled as a module, which will be called appldata_net_sum.o.

Depends on APPLDATA_BASE && NET.

CONFIG_APPLDATA_OS

This provides OS related data to the Linux - VM Monitor Stream, like CPU utilisation, etc. Writing 1 or 0 to /proc/appldata/os creates(1) or removes(0) a z/VM APPLDATA monitor record, i.e. enables or disables monitoring this record on the z/VM side.

Default is disabled. This can also be compiled as a module, which will be called appldata_os.o.

Depends on APPLDATA_BASE.

CONFIG_CHECK_STACK

This option enables the compiler option -mstack-guard and -mstack-size if they are available. If the compiler supports them it will emit additional code to each function prolog to trigger an illegal operation if the kernel stack is about to overflow.

Say N if you are unsure.

CONFIG_CHSC_SCH

This driver allows usage of CHSC subchannels. A CHSC subchannel is usually present on LPAR only. The driver creates a device /dev/chsc, which may be used to obtain I/O configuration information about the machine and to issue asynchronous chsc commands (DANGEROUS). You will usually only want to use this interface on a special LPAR designated for system management.

To compile this driver as a module, choose M here: the module will be called chsc_sch.

If unsure, say N.

CONFIG_CMM

Select this option, if you want to enable the kernel interface to reduce the memory size of the system. This is accomplished by allocating pages of memory and put them "on hold". This only makes sense for a system running under VM where the unused pages will be reused by VM for other guest systems. The interface allows an external monitor to balance memory of many systems. Everybody who wants to run Linux under VM should select this option.

CONFIG_CMM_IUCV

Select this option to enable the special message interface to the cooperative memory management.

Depends on CMM && (SMSGIUCV=y || CMM=SMSGIUCV).

CONFIG_COMPAT

Select this option if you want to enable your system kernel to handle system-calls from ELF binaries for 31 bit ESA. This option (and some other stuff like libraries and such) is needed for executing 31 bit applications. It is safe to say "Y".

Depends on 64BIT.

CONFIG_CRYPTO_AES_S390

This is the s390 hardware accelerated implementation of the AES cipher algorithms (FIPS-197).

As of z9 the ECB and CBC modes are hardware accelerated for 128 bit keys. As of z10 the ECB and CBC modes are hardware accelerated for all AES key sizes. As of z196 the CTR mode is hardware accelerated for all AES key sizes and XTS mode is hardware accelerated for 256 and 512 bit keys.

CONFIG_CRYPTO_DES_S390

This is the s390 hardware accelerated implementation of the DES cipher algorithm (FIPS 46-2), and Triple DES EDE (FIPS 46-3).

As of z990 the ECB and CBC mode are hardware accelerated. As of z196 the CTR mode is hardware accelerated.

CONFIG_CRYPTO_GHASH_S390

This is the s390 hardware accelerated implementation of the GHASH message digest algorithm for GCM (Galois/Counter Mode).

It is available as of z196.

CONFIG_CRYPTO_SHA1_S390

This is the s390 hardware accelerated implementation of the SHA-1 secure hash standard (FIPS 180-1/DFIPS 180-2).

It is available as of z990.

CONFIG_CRYPTO_SHA256_S390

This is the s390 hardware accelerated implementation of the SHA256 secure hash standard (DFIPS 180-2).

It is available as of z9.

CONFIG_CRYPTO_SHA512_S390

This is the s390 hardware accelerated implementation of the SHA512 secure hash standard.

It is available as of z10.

CONFIG_EADM_SCH

This driver allows usage of EADM subchannels. EADM subchannels act as a communication vehicle for SCM increments.

To compile this driver as a module, choose M here: the module will be called eadm_sch.

Depends on SCM_BUS.

CONFIG_HOTPLUG_CPU

Say Y here to be able to turn CPUs off and on. CPUs can be controlled through /sys/devices/system/cpu/cpu#. Say N if you want to disable CPU hotplug.

Depends on SMP.

CONFIG_IUCV

Select this option if you want to use inter-user communication under VM or VIF. If you run on z/VM, say "Y" to enable a fast communication link between VM guests.

CONFIG_KMSG_IDS

Select this option if you want to include a message number to the prefix for kernel messages issued by the s390 architecture and driver code. See "Documentation/s390/kmsg.txt" for more details.

CONFIG_MARCH_G5

This option is part of a choice section (MARCH_G5 | MARCH_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12).

Select this to build a 31 bit kernel that works on all ESA/390 and z/Architecture machines.

Depends on !64BIT.

CONFIG_MARCH_Z10

This option is part of a choice section (MARCH_G5 | MARCH_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12).

Select this to enable optimizations for IBM System z10 (2097 and 2098 series). The kernel will be slightly faster but will not work on older machines.

CONFIG_MARCH_Z196

This option is part of a choice section (MARCH_G5 | MARCH_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12).

Select this to enable optimizations for IBM zEnterprise 114 and 196 (2818 and 2817 series). The kernel will be slightly faster but will not work on older machines.

CONFIG_MARCH_Z900

This option is part of a choice section (MARCH_G5 | MARCH_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12).

Select this to enable optimizations for model z800/z900 (2064 and 2066 series). This will enable some optimizations that are not available on older ESA/390 (31 Bit) only CPUs.

CONFIG_MARCH_Z990

This option is part of a choice section (MARCH_G5 | MARCH_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12).

Select this to enable optimizations for model z890/z990 (2084 and 2086 series). The kernel will be slightly faster but will not work on older machines.

CONFIG_MARCH_Z9_109

This option is part of a choice section (MARCH_G5 | MARCH_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12).

Select this to enable optimizations for IBM System z9 (2094 and 2096 series). The kernel will be slightly faster but will not work on older machines.

CONFIG_MARCH_ZEC12

This option is part of a choice section (MARCH_G5 | MARCH_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12).

Select this to enable optimizations for IBM zBC12 and zEC12 (2828 and 2827 series). The kernel will be slightly faster but will not work on older machines.

CONFIG_NR_CPUS

This allows you to specify the maximum number of CPUs which this kernel will support. The maximum supported value is 256 and the minimum value which makes sense is 2.

This is purely to save memory - each supported CPU adds approximately sixteen kilobytes to the kernel image.

Depends on SMP.

CONFIG_OPROFILE

OProfile is a profiling system capable of profiling the whole system, include the kernel, kernel modules, libraries, and applications.

If unsure, say N.

Depends on PROFILING and the common code option HAVE_OPROFILE.

CONFIG_PACK_STACK

This option enables the compiler option -mkernel-backchain if it is available. If the option is available the compiler supports the new stack layout which dramatically reduces the minimum stack frame size. With an old compiler a non-leaf function needs a minimum of 96 bytes on 31 bit and 160 bytes on 64 bit. With -mkernel-backchain the minimum size drops to 16 byte on 31 bit and 24 byte on 64 bit.

Say Y if you are unsure.

CONFIG_PCI

Enable PCI support.

Depends on 64BIT.

CONFIG_PCI_NR_FUNCTIONS

This allows you to specify the maximum number of PCI functions which this kernel will support.

CONFIG_PCI_NR_MSI

This defines the number of virtual interrupts the kernel will provide for MSI interrupts. If you configure your system to have too few drivers will fail to allocate MSI interrupts for all PCI devices.

CONFIG_PFAULT

Select this option, if you want to use PFAULT pseudo page fault handling under VM. If running native or in LPAR, this option has no effect. If your VM does not support PFAULT, PAGEEX pseudo page fault handling will be used. Note that VM 4.2 supports PFAULT but has a bug in its implementation that causes some problems. Everybody who wants to run Linux under VM != VM4.2 should select this option.

CONFIG_PROFILING

Say Y here to enable the extended profiling support mechanisms used by profilers such as OProfile.

CONFIG_QDIO

This driver provides the Queued Direct I/O base support for IBM System z.

To compile this driver as a module, choose M here: the module will be called qdio.

If unsure, say Y.

CONFIG_S390_HYPFS_FS

This is a virtual file system intended to provide accounting information in an s390 hypervisor environment.

CONFIG_S390_PRNG

Select this option if you want to use the s390 pseudo random number generator. The PRNG is part of the cryptographic processor functions and uses triple-DES to generate secure random numbers like the ANSI X9.17 standard. User-space programs access the pseudo-random-number device through the char device /dev/prandom.

It is available as of z9.

CONFIG_SCHED_BOOK

Book scheduler support improves the CPU scheduler's decision making when dealing with machines that have several books.

Depends on SMP.

CONFIG_SCM_BUS

Bus driver for Storage Class Memory.

CONFIG_SECCOMP

This kernel feature is useful for number crunching applications that may need to compute untrusted bytecode during their execution. By using pipes or other transports made available to the process as file descriptors supporting the read/write syscalls, it's possible to isolate those applications in their own address space using seccomp. Once seccomp is enabled via /proc/<pid>/seccomp, it cannot be disabled and the task is only allowed to execute a few safe syscalls defined by each seccomp mode.

If unsure, say Y.

Depends on the common code option PROC_FS.

CONFIG_SHARED_KERNEL

Select this option, if you want to share the text segment of the Linux kernel between different VM guests. This reduces memory usage with lots of guests but greatly increases kernel size. Also if a kernel was IPL'ed from a shared segment the kexec system call will not work. You should only select this option if you know what you are doing and want to exploit this feature.

Depends on !JUMP_LABEL.

JUMP_LABEL is a common code option.

CONFIG_SMALL_STACK

If you say Y here and the compiler supports the -mkernel-backchain option the kernel will use a smaller kernel stack size. The reduced size is 8kb instead of 16kb. This allows to run more threads on a system and reduces the pressure on the memory management for higher order page allocations.

Say N if you are unsure.

Depends on PACK_STACK && 64BIT && !LOCKDEP.

LOCKDEP is a common code option.

CONFIG_SMP

This enables support for systems with more than one CPU. If you have a system with only one CPU, like most personal computers, say N. If you have a system with more than one CPU, say Y.

If you say N here, the kernel will run on uni- and multiprocessor machines, but will use only one CPU of a multiprocessor machine. If you say Y here, the kernel will run on many, but not all, uniprocessor machines. On a uniprocessor machine, the kernel will run faster if you say N here.

See also the SMP-HOWTO available at <http://www.tldp.org/docs.html#howto>.

Even if you don't know what to do here, say Y.

CONFIG_STACK_GUARD

This allows you to specify the size of the guard area at the lower end of the kernel stack. If the kernel stack points into the guard area on function entry an illegal operation is triggered. The size needs to be a power of 2. Please keep in mind that the size of an interrupt frame is 184 bytes for 31 bit and 328 bytes on 64 bit. The minimum size for the stack guard should be 256 for 31 bit and 512 for 64 bit.

Depends on CHECK_STACK.

CONFIG_TUNE_DEFAULT

This option is part of a choice section (TUNE_DEFAULT | TUNE_G5 | TUNE_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12).

Cause the compiler to tune (-mtune) the generated code for a machine. This will make the code run faster on the selected machine but somewhat slower on other machines. This option only changes how the compiler emits instructions, not the selection of instructions itself, so the resulting kernel will run on all other machines.

CONFIG_TUNE_G5

This option is part of a choice section (TUNE_DEFAULT | TUNE_G5 | TUNE_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12).

Cause the compiler to tune (-mtune) the generated code for a machine. This will make the code run faster on the selected machine but somewhat slower on other machines. This option only changes how the compiler emits instructions, not the selection of instructions itself, so the resulting kernel will run on all other machines.

CONFIG_TUNE_Z10

This option is part of a choice section (TUNE_DEFAULT | TUNE_G5 | TUNE_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12).

Cause the compiler to tune (-mtune) the generated code for a machine. This will make the code run faster on the selected machine but somewhat slower on other machines. This option only changes how the compiler emits instructions, not the selection of instructions itself, so the resulting kernel will run on all other machines.

CONFIG_TUNE_Z196

This option is part of a choice section (TUNE_DEFAULT | TUNE_G5 | TUNE_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12).

Cause the compiler to tune (-mtune) the generated code for a machine. This will make the code run faster on the selected machine but somewhat slower on other machines. This option only changes how the compiler emits instructions, not the selection of instructions itself, so the resulting kernel will run on all other machines.

CONFIG_TUNE_Z900

This option is part of a choice section (TUNE_DEFAULT | TUNE_G5 | TUNE_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12).

Cause the compiler to tune (-mtune) the generated code for a machine. This will make the code run faster on the selected machine but somewhat slower on other machines. This option only changes how the compiler emits instructions, not the selection of instructions itself, so the resulting kernel will run on all other machines.

CONFIG_TUNE_Z990

This option is part of a choice section (TUNE_DEFAULT | TUNE_G5 | TUNE_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12).

Cause the compiler to tune (-mtune) the generated code for a machine. This will make the code run faster on the selected machine but somewhat slower on other machines. This option only changes how the compiler emits instructions, not the selection of instructions itself, so the resulting kernel will run on all other machines.

CONFIG_TUNE_Z9_109

This option is part of a choice section (TUNE_DEFAULT | TUNE_G5 | TUNE_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12).

Cause the compiler to tune (-mtune) the generated code for a machine. This will make the code run faster on the selected machine but somewhat slower on other machines. This option only changes how the compiler emits instructions, not the selection of instructions itself, so the resulting kernel will run on all other machines.

CONFIG_TUNE_ZEC12

This option is part of a choice section (TUNE_DEFAULT | TUNE_G5 | TUNE_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12).

Cause the compiler to tune (-mtune) the generated code for a machine. This will make the code run faster on the selected machine but somewhat slower on other machines. This option only changes how the compiler emits instructions, not the selection of instructions itself, so the resulting kernel will run on all other machines.

CONFIG_ZCRYPT

Select this option if you want to use a PCI-attached cryptographic adapter like: + PCI Cryptographic Accelerator (PCICA) + PCI Cryptographic Coprocessor (PCICC) + PCI-X Cryptographic Coprocessor (PCIXCC) + Crypto Express2 Coprocessor (CEX2C) + Crypto Express2 Accelerator (CEX2A) + Crypto Express3 Coprocessor (CEX3C) + Crypto Express3 Accelerator (CEX3A)

CONFIG_ZFCPDUMP

Select this option if you want to build an zfcpdump enabled kernel. Refer to <file:Documentation/s390/zfcpdump.txt> for more details on this.

Depends on 64BIT && SMP.

Device driver-related options

The device driver-related options comprise all System z specific options in the Device Drivers main menu.

Figure 118 summarizes the device driver-related options in the order in which you find them in the kernel configuration menu. The pages that follow provide explanations for each option in alphabetical order. For architecture-specific options, see "General architecture-specific options" on page 724.

```
 Device Drivers --->
    ...
    Block devices --->                                (common code option CONFIG_BLK_DEV)
       ...
       --- S/390 block device drivers (depends on S390 && BLOCK) ---
       XPRAM disk support                                                 (CONFIG_BLK_DEV_XPRAM)
       DCSSBLK support                                                    (CONFIG_DCSSBLK)
       Support for DASD devices                                           (CONFIG_DASD)
       ├─ Profiling support for dasd devices                              (CONFIG_DASD_PROFILE)
       ├─ Support for ECKD Disks                                          (CONFIG_DASD_ECKD)
       ├─ Support for FBA  Disks                                          (CONFIG_DASD_FBA)
       ├─ Support for DIAG access to Disks                                (CONFIG_DASD_DIAG)
       └─ Extended error reporting (EER)                                  (CONFIG_DASD_EER)
       Support for Storage Class Memory                                   (CONFIG_SCM_BLOCK)*
       └─ SCM force cluster writes                                        (CONFIG_SCM_BLOCK_CLUSTER_WRITE)
       ...
    SCSI device support --->
       ...
       --- SCSI support type (disk, tape, CD-ROM) (depends on SCSI) ---
       ...
       SCSI low-level drivers --->                    (common code option CONFIG_SCSI_LOWLEVEL)
          ...
          FCP host bus adapter driver for IBM eServer zSeries             (CONFIG_ZFCP)*
          ...
     Network device support --->                      (common code option CONFIG_NETDEVICES)
       ...
       S/390 network device drivers (depends on NETDEVICES && S390) --->
          Lan Channel Station Interface                                   (CONFIG_LCS)*
          CTC and MPC SNA device support                                  (CONFIG_CTCM)
          IUCV network device support (VM only)                           (CONFIG_NETIUCV)*
          IUCV special message support (VM only)                          (CONFIG_SMSGIUCV)*
          └─ Deliver IUCV special messages as uevents (VM only)           (CONFIG_SMSGIUCV_EVENT)
          CLAW device support                                             (CONFIG_CLAW)
          Gigabit Ethernet device support                                 (CONFIG_QETH)*
          ├─ qeth layer 2 device support                                  (CONFIG_QETH_L2)
          └─ qeth layer 3 device support                                  (CONFIG_QETH_L3)
```

Figure 118. Device driver-specific kernel configuration menu options 1 of 2. The └─ symbols indicate dependencies on preceding options. Options with more complex dependencies are marked with an asterisk (*).

```
   ...
Character devices --->
   ...
   z/VM IUCV Hypervisor console support (VM only)                 (CONFIG_HVC_IUCV)
   ...
   --- S/390 character device drivers (depends on S390) ---
   Support for locally attached 3270 terminals                    (CONFIG_TN3270)
   Support for tty input/output on 3270 terminals                 (CONFIG_TN3270_TTY)*
   ├─ Support for fullscreen applications on 3270 terminals       (CONFIG_TN3270_FS)*
   └─ Support for console on 3270 terminal                        (CONFIG_TN3270_CONSOLE)*
   Support for 3215 line mode terminal                            (CONFIG_TN3215)*
   └─ Support for console on 3215 line mode terminal              (CONFIG_TN3215_CONSOLE)
   Support for SCLP line mode terminal                            (CONFIG_SCLP_TTY)*
   └─ Support for console on SCLP line mode terminal              (CONFIG_SCLP_CONSOLE)
   Support for SCLP VT220-compatible terminal                     (CONFIG_SCLP_VT220_TTY)
   └─ Support for console on SCLP VT220-compatible terminal       (CONFIG_SCLP_VT220_CONSOLE)
   Control-Program Identification                                 (CONFIG_SCLP_CPI)
   Support for Call Home via Asynchronous SCLP Records            (CONFIG_SCLP_ASYNC)
   S/390 tape device support                                      (CONFIG_S390_TAPE)
   --- S/390 tape hardware support (depends on S390_TAPE) ---
   ├─ Support for 3480/3490 tape hardware                         (CONFIG_S390_TAPE_34XX)
   └─ Support for 3590 tape hardware                              (CONFIG_S390_TAPE_3590)
   Support for the z/VM recording system services (VM only)       (CONFIG_VMLOGRDR)*
   Support for the z/VM CP interface                              (CONFIG_VMCP)
   API for reading z/VM monitor service records                   (CONFIG_MONREADER)*
   API for writing z/VM monitor service records                   (CONFIG_MONWRITER)
   z/VM unit record device driver                                 (CONFIG_S390_VMUR)
   ...
Watchdog Timer Support --->                                       (CONFIG_WATCHDOG)
...
   --- Watchdog Device Drivers" #paragraph# <--remove-->#  Architecture Independent ---
   ...
|  System z diag288 Watchdog                                      (CONFIG_DIAG288_WATCHDOG)
   ...
```

Figure 119. Device driver-specific kernel configuration menu options 2 of 2. The └ symbols indicate dependencies on preceding options. Options with more complex dependencies are marked with an asterisk (*).

The following alphabetically sorted list has details about the device driver-related options summarized in Figure 118 on page 734. For architecture-specific options, see “General architecture-specific options” on page 724.

CONFIG_BLK_DEV_XPRAM

Select this option if you want to use your expanded storage on S/390 or zSeries as a disk. This is useful as a _fast_ swap device if you want to access more than 2G of memory when running in 31 bit mode. This option is also available as a module which will be called xpram. If unsure, say "N".

Depends on S390 && BLOCK.

S390 is an implicitly selected option.

CONFIG_CLAW

This driver supports channel attached CLAW devices. CLAW is Common Link Access for Workstation. Common devices that use CLAW are RS/6000s, Cisco Routers (CIP) and 3172 devices. To compile as a module, choose M. The module name is claw. To compile into the kernel, choose Y.

Depends on CCW && NETDEVICES.

CCW is an implicitly selected option.

CONFIG_CTCM

Select this option if you want to use channel-to-channel point-to-point networking on IBM System z. This device driver supports real CTC coupling using ESCON. It also supports virtual CTCs when running under VM. This driver also supports channel-to-channel MPC SNA devices. MPC is an SNA protocol device used by Communication Server for Linux. To compile as a module, choose M. The module name is ctcm. To compile into the kernel, choose Y. If you do not need any channel-to-channel connection, choose N.

Depends on CCW && NETDEVICES.

CCW is an implicitly selected option.

CONFIG_DASD

Enable this option if you want to access DASDs directly utilizing S/390s channel subsystem commands. This is necessary for running natively on a single image or an LPAR.

Depends on CCW && BLOCK.

CCW is an implicitly selected option.

CONFIG_DASD_DIAG

Select this option if you want to use Diagnose250 command to access Disks under VM. If you are not running under VM or unsure what it is, say "N".

Depends on DASD.

CONFIG_DASD_ECKD

ECKD devices are the most commonly used devices. You should enable this option unless you are very sure to have no ECKD device.

Depends on DASD.

CONFIG_DASD_EER

This driver provides a character device interface to the DASD extended error reporting. This is only needed if you want to use applications written for the EER facility.

Depends on DASD.

CONFIG_DASD_FBA

Select this option to be able to access FBA devices. It is safe to say "Y".

Depends on DASD.

CONFIG_DASD_PROFILE

Enable this option if you want to see profiling information in /proc/dasd/statistics.

Depends on DASD.

CONFIG_DCSSBLK

Support for dcss block device

Depends on S390 && BLOCK.

S390 is an implicitly selected option.

| **CONFIG_DIAG288_WATCHDOG**

|

IBM s/390 and zSeries machines running under z/VM 5.1 or later provide a virtual watchdog timer to their guest that cause a user define Control Program command to be executed after a timeout. LPAR provides a very similar interface. This driver handles both.

To compile this driver as a module, choose M here. The module will be called vmwatchdog.

CONFIG_HVC_IUCV

This driver provides a Hypervisor console (HVC) back-end to access a Linux (console) terminal via a z/VM IUCV communication path.

CONFIG_LCS

Select this option if you want to use LCS networking on IBM System z. This device driver supports FDDI (IEEE 802.7) and Ethernet. To compile as a module, choose M. The module name is lcs. If you do not know what it is, it's safe to choose Y.

Depends on CCW && NETDEVICES && (ETHERNET || FDDI).

ETHERNET and FDDI are common code options. CCW is an implicitly selected option.

CONFIG_MONREADER

Character device driver for reading z/VM monitor service records

Depends on IUCV.

CONFIG_MONWRITER

Character device driver for writing z/VM monitor service records

CONFIG_NETIUCV

Select this option if you want to use inter-user communication vehicle networking under VM or VIF. It enables a fast communication link between VM guests. Using ifconfig a point-to-point connection can be established to the Linux on IBM System z running on the other VM guest. To compile as a module, choose M. The module name is netiucv. If unsure, choose Y.

Depends on IUCV && NETDEVICES.

CONFIG_QETH

This driver supports the IBM System z OSA-Express adapters in QDIO mode (all media types), HiperSockets interfaces and z/VM virtual NICs for Guest LAN and VSWITCH.

For details please refer to the documentation provided by IBM at <http://www.ibm.com/developerworks/linux/linux390>

To compile this driver as a module, choose M. The module name is qeth.

Depends on CCW && NETDEVICES && IP_MULTICAST && QDIO.

IP_MULTICAST is a common code option. CCW is an implicitly selected option.

CONFIG_QETH_L2

Select this option to be able to run qeth devices in layer 2 mode. To compile as a module, choose M. The module name is qeth_l2. If unsure, choose y.

Depends on QETH.

CONFIG_QETH_L3

Select this option to be able to run qeth devices in layer 3 mode. To compile as a module choose M. The module name is qeth_l3. If unsure, choose Y.

Depends on QETH.

CONFIG_S390_TAPE

Select this option if you want to access channel-attached tape devices on IBM S/390 or zSeries. If you select this option you will also want to select at least one of the tape interface options and one of the tape hardware options in order to access a tape device. This option is also available as a module. The module will be called tape390 and include all selected interfaces and hardware drivers.

CONFIG_S390_TAPE_34XX

Select this option if you want to access IBM 3480/3490 magnetic tape subsystems and 100% compatibles. It is safe to say "Y" here.

Depends on S390_TAPE.

CONFIG_S390_TAPE_3590

Select this option if you want to access IBM 3590 magnetic tape subsystems and 100% compatibles. It is safe to say "Y" here.

Depends on S390_TAPE.

CONFIG_S390_VMUR

Character device driver for z/VM reader, puncher and printer.

CONFIG_SCLP_ASYNC

This option enables the call home function, which is able to inform the service element and connected organisations about a kernel panic. You should only select this option if you know what you are doing, want for inform other people about your kernel panics, need this feature and intend to run your kernel in LPAR.

CONFIG_SCLP_CONSOLE

Include support for using an IBM HWC line-mode terminal as the Linux system console.

Depends on SCLP_TTY.

CONFIG_SCLP_CPI

This option enables the hardware console interface for system identification. This is commonly used for workload management and gives you a nice name for the system on the service element. Please select this option as a module since built-in operation is completely untested. You should only select this option if you know what you are doing, need this feature and intend to run your kernel in LPAR.

CONFIG_SCLP_TTY

Include support for IBM SCLP line-mode terminals.

Depends on S390 && TTY.

TTY is a common code option. S390 is an implicitly selected option.

CONFIG_SCLP_VT220_CONSOLE

Include support for using an IBM SCLP VT220-compatible terminal as a Linux system console.

Depends on SCLP_VT220_TTY.

CONFIG_SCLP_VT220_TTY

Include support for an IBM SCLP VT220-compatible terminal.

Depends on S390 && TTY.

TTY is a common code option. S390 is an implicitly selected option.

CONFIG_SCM_BLOCK

Block device driver for Storage Class Memory (SCM). This driver provides a block device interface for each available SCM increment.

To compile this driver as a module, choose M here: the module will be called scm_block.

Depends on S390 && BLOCK && EADM_SCH && SCM_BUS.

S390 is an implicitly selected option.

CONFIG_SCM_BLOCK_CLUSTER_WRITE

Force writes to Storage Class Memory (SCM) to be in done in clusters.

Depends on SCM_BLOCK.

CONFIG_SMSGIUCV

Select this option if you want to be able to receive SMSG messages from other VM guest systems.

Depends on IUCV.

CONFIG_SMSGIUCV_EVENT

Select this option to deliver CP special messages (SMSGs) as uevents. The driver handles only those special messages that start with "APP".

To compile as a module, choose M. The module name is "smsgiucv_app".

Depends on SMSGIUCV.

CONFIG_TN3215

Include support for IBM 3215 line-mode terminals.

Depends on S390 && TTY.

TTY is a common code option. S390 is an implicitly selected option.

CONFIG_TN3215_CONSOLE

Include support for using an IBM 3215 line-mode terminal as a Linux system console.

Depends on TN3215.

CONFIG_TN3270

Include support for IBM 3270 terminals.

CONFIG_TN3270_CONSOLE

Include support for using an IBM 3270 terminal as a Linux system console. Available only if 3270 support is compiled in statically.

Depends on TN3270=y && TN3270_TTY=y.

CONFIG_TN3270_FS

Include support for fullscreen applications on an IBM 3270 terminal.

Depends on TN3270.

CONFIG_TN3270_TTY

Include support for using an IBM 3270 terminal as a Linux tty.

Depends on S390 && TTY.

TTY is a common code option. S390 is an implicitly selected option.

CONFIG_VMCP

Select this option if you want to be able to interact with the control program on z/VM

CONFIG_VMLOGRDR

Select this option if you want to be able to receive records collected by the z/VM recording system services, eg. from *LOGREC, *ACCOUNT or *SYMPTOM. This driver depends on the IUCV support driver.

Depends on IUCV.

CONFIG_WATCHDOG

If you say Y here (and to one of the following options) and create a character special file /dev/watchdog with major number 10 and minor number 130 using mknod ("man mknod"), you will get a watchdog, i.e.: subsequently opening the file and then failing to write to it for longer than 1 minute will result in rebooting the machine. This could be useful for a networked machine that needs to come back on-line as fast as possible after a lock-up. There's both a watchdog implementation entirely in software (which can sometimes fail to reboot the machine) and a driver for hardware watchdog boards, which are more robust and can also keep track of the temperature inside your computer. For details, read <file:Documentation/watchdog/watchdog-api.txt> in the kernel source.

The watchdog is usually used together with the watchdog daemon which is available from <ftp://ibiblio.org/pub/Linux/system/daemons/watchdog/>. This daemon can also monitor NFS connections and can reboot the machine when the process table is full.

If unsure, say N.

CONFIG_ZFCP

If you want to access SCSI devices attached to your IBM eServer zSeries by means of Fibre Channel interfaces say Y. For details please refer to the documentation provided by IBM at <http://oss.software.ibm.com/developerworks/opensource/linux390>

This driver is also available as a module. This module will be called zfcp. If you want to compile it as a module, say M here and read <file:Documentation/kbuild/modules.txt>.

Depends on S390 && QDIO && SCSI.

SCSI is a common code option. S390 is an implicitly selected option.

Part 11. Appendixes

Appendix A. Accessibility

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

Documentation accessibility

The Linux on System z publications are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF file and want to request a Web-based format for this publication, use the Readers' Comments form in the back of this publication, send an email to eservdoc@de.ibm.com, or write to:

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany

In the request, be sure to include the publication number and title.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility at

`www.ibm.com/able`

Appendix B. Understanding syntax diagrams

This section describes how to read the syntax diagrams in this manual.

To read a syntax diagram follow the path of the line. Read from left to right and top to bottom.

- The ▶▶—— symbol indicates the beginning of a syntax diagram.
- The ——▶ symbol, at the end of a line, indicates that the syntax diagram continues on the next line.
- The ▶—— symbol, at the beginning of a line, indicates that a syntax diagram continues from the previous line.
- The ——▶◀ symbol indicates the end of a syntax diagram.

Syntax items (for example, a keyword or variable) may be:

- Directly on the line (required)
- Above the line (default)
- Below the line (optional)

If defaults are determined by your system status or settings, they are not shown in the diagram. Instead the rule is described together with the option, keyword, or variable in the list following the diagram.

Case sensitivity
Unless otherwise noted, entries are case sensitive.

Symbols
You **must** code these symbols exactly as they appear in the syntax diagram

*****	Asterisk
:	Colon
,	Comma
=	Equals sign
-	Hyphen
//	Double slash
( )	Parentheses
.	Period
+	Add
$	Dollar sign

For example:

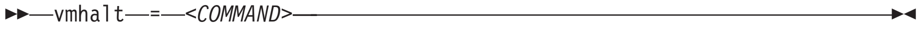
```
dasd=0.0.7000-0.0.7fff
```

Variables
An <*italicized*> lowercase word enclosed in angled brackets indicates a variable that you must substitute with specific information. For example:

```
▶▶—— -p ——<interface>——————————————————————————————————————————▶◀
```

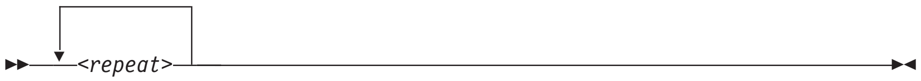
Here you must code **-p** as shown and supply a value for <*interface*>.

An italicized uppercase word in angled brackets indicates a variable that must appear in uppercase:

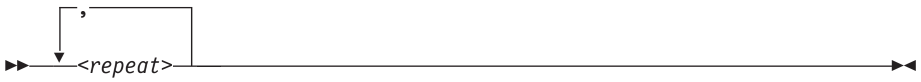


Repetition

An arrow returning to the left means that the item can be repeated.

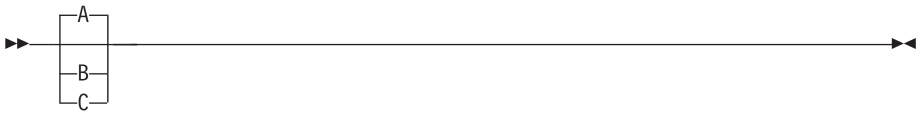


A character within the arrow means you must separate repeated items with that character.



Defaults

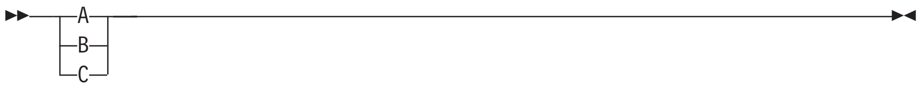
Defaults are above the line. The system uses the default unless you override it. You can override the default by coding an option from the stack below the line. For example:



In this example, A is the default. You can override `A` by choosing `B` or `C`.

Required Choices

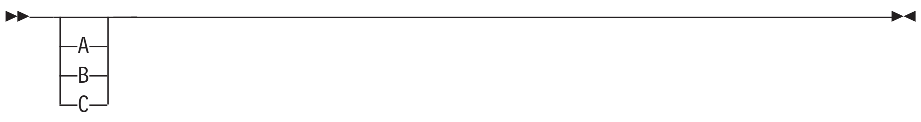
When two or more items are in a stack and one of them is on the line, you **must** specify one item. For example:



Here you must enter either `A` or `B` or `C`.

Optional Choice

When an item is below the line, the item is optional. Only one item **may** be chosen. For example:



Here you may enter either `A` or `B` or `C`, or you may omit the field.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Trademarks

Glossary

This glossary includes IBM product terminology as well as selected other terms and definitions.

Additional information can be obtained in:

- The American National Standard Dictionary for Information Systems, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036.
- The ANSI/EIA Standard–440-A, Fiber Optic Terminology. Copies may be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue, N.W., Washington, DC 20006.
- The Information Technology Vocabulary developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1).
- The IBM Dictionary of Computing, New York: McGraw-Hill, 1994.
- Internet Request for Comments: 1208, Glossary of Networking Terms
- Internet Request for Comments: 1392, Internet Users' Glossary
- The Object-Oriented Interface Design: IBM Common User Access Guidelines , Carmel, Indiana: Que, 1992.

Numerics

10 Gigabit Ethernet. An Ethernet network with a bandwidth of 10000-Mbps.

3215. IBM console printer-keyboard.

3270. IBM information display system.

3370, 3380 or 3390. IBM direct access storage device (disk).

3480, 3490, 3590. IBM magnetic tape subsystem.

9336 or 9345. IBM direct access storage device (disk).

A

address space. The range of addresses available to a computer program or process. Address space can refer to physical storage, virtual storage, or both.

asynchronous transfer mode (ATM). A transfer mode in which the information is organized into cells; it is asynchronous in the sense that the recurrence of cells containing information from an individual user is not necessarily periodic. ATM is specified in international standards such as ATM Forum UNI 3.1.

auto-detection. Listing the addresses of devices attached to a card by issuing a query command to the card.

C

CCL.

The Communication Controller for Linux on zSeries (CCL) replaces the 3745/6 Communication Controller so that the Network Control Program (NCP) software can continue to provide business critical functions like SNI, XRF, BNN, INN, and SSCP takeover. This allows you to leverage your existing NCP functions on a "virtualized" communication controller within the Linux zSeries environment.

cdl. compatible disk layout. A disk structure for Linux on System z which allows access from other System z operating systems. This replaces the older **ldl**.

CEC. (Central Electronics Complex). A synonym for *CPC*.

channel subsystem. The programmable input/output processors of the System z, which operate in parallel with the CPU.

checksum. An error detection method using a check byte appended to message data

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. In Linux, an output device for kernel messages.

CPC. (Central Processor Complex). A physical collection of hardware that includes main storage, one or more central processors, timers, and channels. Also referred to as a *CEC*.

CRC. cyclic redundancy check. A system of error checking performed at both the sending and receiving station after a block-check character has been accumulated.

CSMA/CD. carrier sense multiple access with collision detection

CTC. channel to channel. A method of connecting two computing devices.

CUU. control unit and unit address. A form of addressing for System z devices using device numbers.

D

DASD. direct access storage device. A mass storage medium on which a computer stores data.

device driver.
- A file that contains the code needed to use an attached device.
- A program that enables a computer to communicate with a specific peripheral device; for example, a printer, a videodisc player, or a CD-ROM drive.
- A collection of subroutines that control the interface between I/O device adapters and the processor.

DIAGNOSE. In z/VM, a set of instructions that programs running on z/VM guest virtual machines can call to request CP services.

disconnected device. In Linux on System z, a device that is online, but to which Linux can no longer find a connection. Reasons include:
- The device was physically removed
- The device was logically removed, for example, with a CP DETACH command in z/VM
- The device was varied offline

E

ECKD. extended count-key-data device. A disk storage device that has a data transfer rate faster than some processors can utilize and that is connected to the processor through use of a speed matching buffer. A specialized channel program is needed to communicate with such a device.

ESCON. enterprise systems connection. A set of IBM products and services that provide a dynamically connected environment within an enterprise.

Ethernet. A 10-Mbps baseband local area network that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and delayed retransmission. Ethernet uses CSMA/CD.

F

Fast Ethernet (FENET). Ethernet network with a bandwidth of 100 Mbps

FBA. fixed block architecture. A type of DASD on Multiprise 3000 or P/390 or emulated by z/VM.

FDDI. fiber distributed data interface. An American National Standards Institute (ANSI) standard for a 100-Mbps LAN using optical fiber cables.

fibre channel. A technology for transmitting data between computer devices. It is especially suited for attaching computer servers to shared storage devices and for interconnecting storage controllers and drives.

FTP. file transfer protocol. In the Internet suite of protocols, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

G

Gigabit Ethernet (GbE). An Ethernet network with a bandwidth of 1000-Mbps

G3, G4, G5 and G6. The generation names of the S/390 CMOS based product family.

H

hardware console. A service-call logical processor that is the communication feature between the main processor and the service processor.

Host Bus Adapter (HBA). An I/O controller that connects an external bus, such as a Fibre Channel, to the internal bus (channel subsystem).

In a Linux environment HBAs are normally virtual and are shown as an FCP device.

HMC. hardware management console. A console used to monitor and control hardware such as the System z microprocessors.

HFS. hierarchical file system. A system of arranging files into a tree structure of directories.

I

intraensemble data network (IEDN). A private 10 Gigabit Ethernet network for application data communications within an ensemble. Data communications for workloads can flow over the IEDN within and between nodes of an ensemble. All of the physical and logical resources of the IEDN are configured, provisioned, and managed by the Unified Resource Manager.

intranode management network (INMN). A private 1000BASE-T Ethernet network operating at 1 Gbps that is required for the Unified Resource Manager to manage the resources within a single zEnterprise node. The INMN connects the Support Element (SE) to the zEnterprise CPC and to any attached zEnterprise BladeCenter Extension (zBX).

ioctl system call. Performs low-level input- and output-control operations and retrieves device status information. Typical operations include buffer manipulation and query of device mode or status.

IOCS. input / output channel subsystem. See channel subsystem.

IP. internet protocol. In the Internet suite of protocols, a connectionless protocol that routes data through a network or interconnected networks and acts as an intermediary between the higher protocol layers and the physical network.

IP address. The unique 32-bit address that specifies the location of each device or workstation on the Internet. For example, 9.67.97.103 is an IP address.

IPIP. IPv4 in IPv4 tunnel, used to transport IPv4 packets in other IPv4 packets.

IPL. initial program load (or boot).

- The initialization procedure that causes an operating system to commence operation.
- The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction.
- The process of loading system programs and preparing a system to run jobs.

IPv6. IP version 6. The next generation of the Internet Protocol.

IUCV. inter-user communication vehicle. A z/VM facility for passing data between virtual machines and z/VM components.

K

kernel. The part of an operating system that performs basic functions such as allocating hardware resources.

kernel module. A dynamically loadable part of the kernel, such as a device driver or a file system.

kernel image. The kernel when loaded into memory.

L

LCS. LAN channel station. A protocol used by OSA.

ldl. Linux disk layout. A basic disk structure for Linux on System z. Now replaced by `cdl`.

LDP. Linux Documentation Project. An attempt to provide a centralized location containing the source material for all open source Linux documentation. Includes user and reference guides, HOW TOs, and FAQs. The homepage of the Linux Documentation Project is

`www.linuxdoc.org`

Linux. a variant of UNIX which runs on a wide range of machines from wristwatches through personal and small business machines to enterprise systems.

Linux on System z. the port of Linux to the IBM System z architecture.

LPAR. logical partition of a 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).

- A collection of objects that can be accessed by means of a network management protocol.
- A definition for management information that specifies the information available from a host or gateway and the operations allowed.

MTU. maximum transmission unit. The largest block which may be transmitted as a single unit.

Multicast. A protocol for the simultaneous distribution of data to a number of recipients, for example live video transmissions.

Multiprise. An enterprise server of the S/390 family.

N

NIC. network interface card. The physical interface between the IBM mainframe and the network.

O

OSA-2. Open Systems Adapter-2. A common System z network interface feature

OSA-Express. Abbreviation for Open Systems Adapter-Express networking features. These include 10 Gigabit Ethernet, Gigabit Ethernet, Fast Ethernet, and ATM.

OSM. OSA-Express for Unified Resource Manager. An OSA-Express channel path identifier (CHPID) type that provides connectivity to the intranode management network (INMN).

OSPF. open shortest path first. A function used in route optimization in networks.

OSX. OSA-Express for zBX. A CHPID type that provides connectivity and access control to the intraensemble data network (IEDN) from a zEnterprise CPC to zBX.

P

POR. power-on reset

POSIX. Portable Operating System Interface for Computer Environments. An IEEE operating system standard closely related to the UNIX system.

R

router. A device or process which allows messages to pass between different networks.

S

S/390. The predecessor of System z.

SA/SE. stand alone support element. See SE.

SE. support element.

- An internal control element of a processor that assists in many of the processor operational functions.
- A hardware unit that provides communications, monitoring, and diagnostic functions to a central processor complex.

SNA. systems network architecture. The IBM architecture that defines the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. The layered structure of SNA allows the ultimate origins and destinations of information (the users) to be independent of and unaffected by the specific SNA network services and facilities that are used for information exchange.

SNMP (Simple Network Management Protocol). In the Internet suite of protocols, a network management protocol that is used to monitor routers and attached networks. SNMP is an application layer protocol. Information about devices managed is defined and stored in the application's Management Information Base (MIB).

Sysctl. system control programming manual control (frame). A means of dynamically changing certain Linux kernel parameters during operation.

T

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

Unified Resource Manager. IBM zEnterprise Unified Resource Manager. Licensed internal code (LIC), also known as firmware, that is part of the Hardware Management Console. The Unified Resource Manager provides energy monitoring and management, goal-oriented policy management, increased security, virtual networking, and data management for the physical and logical resources of a given ensemble.

UNIX. An operating system developed by Bell Laboratories that features multiprogramming in a multiuser environment. The UNIX operating system was originally developed for use on minicomputers but has been adapted for mainframes and microcomputers.

V

VEPA. Virtual Ethernet Port Aggregator

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 Ethernet Port Aggregator. The capability of a physical server to collaborate with an adjacent bridge to provide frame relay services between multiple virtual machines, which are located on a server and also on the external network.

Virtual LAN (VLAN). A group of devices on one ore more LANs that are configured (using management software) so that they can communicate as if they were attached to the same wire, when in fact they are located on a number of different LAN segments. Because VLANs are based on logical rather than physical connections, they are extremely flexible.

volume. A data carrier that is usually mounted and demounted as a unit, for example a tape cartridge or a disk pack. If a storage unit has no demountable packs the volume is the portion available to a single read/write mechanism.

Z

z114. IBM zEnterprise 114.

z196. IBM zEnterprise 196.

zBC12. IBM zEnterprise BC12.

zBX. IBM zEnterprise BladeCenter Extension.

zEC12. IBM zEnterprise EC12.

zEnterprise. IBM zEnterprise System. A heterogeneous hardware infrastructure that can consist of an IBM zEnterprise BC12, a zEnterprise EC12 (zEC12), a zEnterprise 114 (z114) or a zEnterprise 196 (z196) and an attached IBM zEnterprise BladeCenter Extension (zBX), managed as a single logical virtualized system by the Unified Resource Manager.

zSeries. The family of IBM enterprise servers that demonstrate outstanding reliability, availability, scalability, security, and capacity in today's network computing environments.

Bibliography

The publications listed in this chapter are considered useful for a more detailed study of the topics contained in this publication.

Linux on System z publications

Linux on System z publications can be found on the developerWorks website.

See
www.ibm.com/developerworks/linux/linux390/documentation_dev.html

- *Device Drivers, Features, and Commands*, SC33-8411
- *Using the Dump Tools*, SC33-8412
- *How to use FC-attached SCSI devices with Linux on System z*, SC33-8413
- *How to Improve Performance with PAV*, SC33-8414
- *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596
- *Kernel Messages*, SC34-2599
- *libica Programmer's Reference*, SC34-2602

For versions of this and other documents that have been adapted to a particular distribution, see one of the following web pages:
www.ibm.com/developerworks/linux/linux390/documentation_suse.html
www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html

z/VM publications

The publication numbers listed are for z/VM version 6.

For the complete library including other versions, see
`www.ibm.com/vm/library`

- *z/VM Connectivity*, SC24-6174
- *z/VM CP Commands and Utilities Reference*, SC24-6175
- *z/VM CP Planning and Administration*, SC24-6178
- *z/VM CP Programming Services*, SC24-6179
- *z/VM Getting Started with Linux on System z*, SC24-6194
- *z/VM Performance*, SC24-6208
- *z/VM Saved Segments Planning and Administration*, SC24-6229
- *z/VM Systems Management Application Programming*, SC24-6234
- *z/VM TCP/IP Planning and Customization*, SC24-6238
- *z/VM Virtual Machine Operation*, SC24-6241
- *REXX/VM Reference*, SC24-6221
- *REXX/VM User's Guide*, SC24-6222

IBM Redbooks publications

You can search for, view, or download Redbooks publications, Redpapers™, Hints and Tips, draft publications and additional materials on the Redbooks website.

You can also order hardcopy Redbooks or CD-ROMs. See `www.ibm.com/redbooks`

- *IBM zEnterprise Unified Resource Manager*, SG24-7921
- *Building Linux Systems under IBM VM*, REDP-0120
- *FICON CTC Implementation*, REDP-0158
- *Networking Overview for Linux on zSeries*, REDP-3901
- *IBM Communication Controller Migration Guide*, SG24-6298
- *Linux on IBM eServer zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP-3596
- *Security on z/VM*, SG24-7471
- *Linux on IBM eServer zSeries and S/390: VSWITCH and VLAN Features of z/VM 4.4*, REDP-3719
- *Fibre Channel Protocol for Linux and z/VM on IBM System z*, SG24-7266

Other System z publications

General System z publications that might be of interest in the context of Linux on System z.

- *zEnterprise System Introduction to Ensembles*, GC27-2609
- *zEnterprise System Ensemble Planning and Configuring Guide*, GC27-2608
- *System z Application Programming Interfaces*, SB10-7030
- *IBM TotalStorage Enterprise Storage Server System/390 Command Reference 2105 Models E10, E20, F10, and F20*, SC26-7295
- *Processor Resource/Systems Manager Planning Guide*, SB10-7041
- *z/Architecture Principles of Operation*, SA22-7832
- *z/Architecture The Load-Program-Parameter and the CPU-Measurement Facilities*, SA23-2260
- *IBM The CPU-Measurement Facility Extended Counters Definition for z10, z196, z114 and zEC12*, SA23-2261

Networking publications

- *HiperSockets Implementation Guide*, SG24-6816
- *OSA-Express Customer's Guide and Reference*, SA22-7935
- *OSA-Express Implementation Guide*, SG25-5848

Security related publications

- *zSeries Crypto Guide Update*, SG24-6870
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294

ibm.com resources

On the ibm.com® website you can find information about many aspects of Linux on System z including z/VM, I/O connectivity, and cryptography.

- For CMS and CP Data Areas, Control Block information, and the layout of the z/VM monitor records see

  `www.ibm.com/vm/pubs/ctlblk.html`
- For I/O connectivity on System z information, see

  `www.ibm.com/systems/z/connectivity`
- For Communications server for Linux information, see

  `www.ibm.com/software/network/commserver/linux`
- For information about performance monitoring on z/VM, see

  `www.ibm.com/vm/perf`
- For cryptographic coprocessor information, see

  `www.ibm.com/security/cryptocards`
- (Requires registration.) For information for planning, installing, and maintaining IBM Systems, see

  `www.ibm.com/servers/resourcelink`
- For information about STP, see

  `www.ibm.com/systems/z/advantages/pso/stp.html`

Finding IBM publications

For the referenced IBM publications, links have been omitted to avoid pointing to a particular edition of a publication.

You can locate the latest versions of the referenced IBM publications through the IBM Publications Center at

`www.ibm.com/shop/publications/order`

Index

Readers’ Comments — We'd Like to Hear from You

Linux on System z
Device Drivers, Features, and Commands
Development stream (Kernel 3.16)

Publication No. SC33-8411-25

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:
- Send your comments to the address on the reverse side of this form.
- Send your comments via email to: eservdoc@de.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	Email address

Readers' Comments — We'd Like to Hear from You
SC33-8411-25

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany

Fold and Tape | **Please do not staple** | Fold and Tape

SC33-8411-25

Cut or Fold
Along Line

SC33-8411-25