Linux on Z and LinuxONE

*Device Drivers, Features, and Commands Development stream (Kernel 4.19)*

IBM
Note
Before using this document, be sure to read the information in “Notices” on page 803.

This edition applies to the Linux on Z Development stream for kernel 4.19, s390-tools version 2.7.0, and to all subsequent releases and modifications until otherwise indicated in new editions.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

Summary of changes........................................................................................................... vii
About this document............................................................................................................ ix
Part 1. General concepts...................................................................................................... 1
  Chapter 1. How devices are accessed by Linux............................................................... 3
  Chapter 2. Devices in sysfs............................................................................................. 7
  Chapter 3. Device auto-configuration for Linux in LPAR mode...................................... 19
  Chapter 4. Kernel and module parameters.................................................................... 23
Part 2. Booting and shutdown.......................................................................................... 31
  Chapter 5. Console device drivers................................................................................ 33
  Chapter 6. Initial program loader for IBM Z - zipl......................................................... 59
  Chapter 7. Booting Linux............................................................................................. 79
  Chapter 8. Suspending and resuming Linux................................................................. 99
  Chapter 9. Shutdown actions....................................................................................... 105
  Chapter 10. The diag288 watchdog device driver...................................................... 109
  Chapter 11. Remotely controlling virtual hardware - snipl........................................... 115
Part 3. Storage.................................................................................................................. 135
  Chapter 12. DASD device driver.................................................................................. 137
  Chapter 13. SCSI-over-Fibre Channel device driver................................................... 177
  Chapter 14. Storage-class memory device driver....................................................... 221
  Chapter 15. Channel-attached tape device driver....................................................... 225
  Chapter 16. XPRAM device driver.............................................................................. 235
Part 4. Networking............................................................................................................ 241
  Chapter 17. qeth device driver for OSA-Express (QDIO) and HiperSockets................. 243
  Chapter 18. OSA-Express SNMP subagent support..................................................... 311
  Chapter 19. LAN channel station device driver......................................................... 319
  Chapter 20. CTCM device driver............................................................................... 325
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.</td>
<td>NETIUCV device driver</td>
</tr>
<tr>
<td>22.</td>
<td>AF_IUCV address family support</td>
</tr>
<tr>
<td>23.</td>
<td>SMC protocol support</td>
</tr>
<tr>
<td>24.</td>
<td>RDMA over Converged Ethernet</td>
</tr>
<tr>
<td>25.</td>
<td>Internal shared memory device driver</td>
</tr>
<tr>
<td>26.</td>
<td>Managing CPUs</td>
</tr>
<tr>
<td>27.</td>
<td>NUMA emulation</td>
</tr>
<tr>
<td>28.</td>
<td>Managing hotplug memory</td>
</tr>
<tr>
<td>29.</td>
<td>Persistent device configuration</td>
</tr>
<tr>
<td>30.</td>
<td>Large page support</td>
</tr>
<tr>
<td>31.</td>
<td>S/390 hypervisor file system</td>
</tr>
<tr>
<td>32.</td>
<td>CHSC subchannel device driver</td>
</tr>
<tr>
<td>33.</td>
<td>Call logical processor device driver</td>
</tr>
<tr>
<td>34.</td>
<td>ETR- and STP-based clock synchronization</td>
</tr>
<tr>
<td>35.</td>
<td>Identifying the IBM Z hardware</td>
</tr>
<tr>
<td>36.</td>
<td>HMC media device driver</td>
</tr>
<tr>
<td>37.</td>
<td>Data compression with GenWQE and zEDC Express</td>
</tr>
<tr>
<td>38.</td>
<td>PCI Express support</td>
</tr>
<tr>
<td>39.</td>
<td>z/VM concepts</td>
</tr>
<tr>
<td>40.</td>
<td>Writing kernel APPLDATA records</td>
</tr>
<tr>
<td>41.</td>
<td>Writing z/VM monitor records</td>
</tr>
<tr>
<td>42.</td>
<td>Reading z/VM monitor records</td>
</tr>
<tr>
<td>43.</td>
<td>z/VM recording device driver</td>
</tr>
<tr>
<td>44.</td>
<td>z/VM unit record device driver</td>
</tr>
<tr>
<td>45.</td>
<td>z/VM DCSS device driver</td>
</tr>
<tr>
<td>46.</td>
<td>z/VM CP interface device driver</td>
</tr>
</tbody>
</table>
Summary of changes

This revision reflects changes to the Development stream for kernel 4.19.

Updates for kernel 4.19

This edition (SC33-8411-36) contains changes related to the 4.19 kernel release.

New information

- You can now use internal shared memory devices for fast communication between LPARs within a mainframe system, see Chapter 23, “SMC protocol support,” on page 349 and Chapter 25, “Internal shared memory device driver,” on page 357.
- You can now mark sets of AP queues for handling by device drivers other than zcrypt, for example in KVM guests, see “Marking AP queues for alternative device drivers” on page 508.
- The new zpcic1 command helps you to handle defective PCIE devices, see “zpcic1 - Manage defective PCIE devices” on page 755.

Changed Information

- The PCI Express information has been moved to a separate chapter, see Chapter 38, “PCI Express support,” on page 427.

This revision also includes maintenance and editorial changes.

Deleted Information

- The command references for zkey and zkey-cryptsetup have been moved to Pervasive Encryption for Data Volumes, SC34-2782.

Updates for kernel 4.17 and 4.18

This edition (SC33-8411-35) contains changes related to the 4.17 and 4.18 kernel releases.

New information

- Linux in LPAR mode can now process device configuration data that is stored on the (SE), see Chapter 3, “Device auto-configuration for Linux in LPAR mode,” on page 19.
- The DASD device driver supports a new option to prevent automatic removal of defective paths, see “Setting defective channel paths offline automatically” on page 169.
- You can now use SMC sockets for RDMA connections with InfiniBand, see Chapter 24, “RDMA over Converged Ethernet,” on page 353.
- You can now use the zkey-cryptsetup command to manage secure AES keys of volumes that are encrypted with LUKS2, see Pervasive Encryption for Data Volumes, SC34-2782.

Changed Information

- None.

This revision also includes maintenance and editorial changes.

Deleted Information

- None.
Updates for kernel 4.16

This edition (SC33-8411-34) contains changes related to the 4.16 kernel release.

New information

• If the qeth device driver can detect the required layer for z/VM NICs, it sets the layer for you, see “Layer 2 and layer 3” on page 250.

• The zkey command can now manage keys in a repository, see Pervasive Encryption for Data Volumes, SC34-2782.

Changed Information

• None.

This revision also includes maintenance and editorial changes.

Deleted Information

• None.
This document describes the device drivers available for Linux kernel 4.19 to control devices and attachments for IBM Z (Z) and IBM LinuxONE™ (LinuxONE) servers. It also describes commands and parameters for configuring Linux on Z.

As of kernel version 4.1, Linux no longer supports 31-bit Z mainframes.

IBM® z/Architecture® hardware on which you can run Linux kernel 4.1 and later includes IBM z14 (z14), IBM z14 Model ZR1, (z14 ZR1), IBM z13® (z13®), IBM z13s® (z13s®), zEnterprise® BC12 (zBC12), IBM zEnterprise EC12 (zEC12), IBM zEnterprise 114 (z114), IBM zEnterprise 196 (z196), and IBM LinuxONE servers.

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

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

For 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 Z and LinuxONE library on the developerWorks® website at www.ibm.com/developerworks/linux/linux390/documentation_dev.html.

The first part of this document contains general and overview information for the z/Architecture specific 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 Z.

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

Most of the information in this document is intended for system administrators who want to configure Linux on Z.
The following general assumptions are made about your background knowledge:

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

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

**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, on GitHub, or are commonly available tools.

Your Linux distribution might provide additional utilities for working with IBM 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:


---

**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.

**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 or a stand-alone dump utility. See also “IPL and booting” on page 79.

**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:

```
#
```
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 142). Each device name is associated with a minor number (see Figure 1 on page 3).

Figure 1: Minor numbers and device names

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

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 on page 4 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:

```bash
# 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 144 and “Examples for udev-created tape device nodes” on page 227 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**

4 Linux on Z and LinuxONE: Device Drivers, Features, and Commands Development stream (Kernel 4.19)
When a network device is defined, it is associated with a real or virtual network adapter (see Figure 3 on page 4). You can configure the adapter properties for a particular network device through the device representation in sysfs (see “Device directories” on page 9).

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

**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 on page 5 summarizes the base names that network device drivers use for interfaces that are associated with real hardware.

<table>
<thead>
<tr>
<th>Base name</th>
<th>Interface type</th>
<th>Device driver module</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>eth</td>
<td>Ethernet</td>
<td>qeth, lcs</td>
<td>OSA-Express features</td>
</tr>
<tr>
<td>eth</td>
<td>Ethernet</td>
<td>mlx4_en</td>
<td>RoCE Express feature</td>
</tr>
<tr>
<td>osn</td>
<td>ESCON/CDLC bridge</td>
<td>qeth</td>
<td>OSA-Express features</td>
</tr>
<tr>
<td>ctc</td>
<td>Channel-to-Channel</td>
<td>ctcm</td>
<td>ESCON channel card, FICON® channel card</td>
</tr>
<tr>
<td>mpc</td>
<td>Channel-to-Channel</td>
<td>ctcm</td>
<td>ESCON channel card</td>
</tr>
</tbody>
</table>

Table 2 on page 5 summarizes the base names that network device drivers use for interfaces that are associated with virtual hardware:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Interface type</th>
<th>Device driver module</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsi</td>
<td>HiperSockets™, virtual NIC</td>
<td>qeth</td>
<td>Real HiperSockets or virtual NIC type HiperSockets coupled to a guest LAN</td>
</tr>
<tr>
<td>eth</td>
<td>virtual NIC</td>
<td>qeth</td>
<td>QDIO virtual NIC coupled to a guest LAN or virtual switch</td>
</tr>
<tr>
<td>ctc</td>
<td>virtual Channel-to-Channel</td>
<td>ctcm</td>
<td>virtual CTCA</td>
</tr>
<tr>
<td>mpc</td>
<td>virtual Channel-to-Channel</td>
<td>ctcm</td>
<td>virtual CTCA</td>
</tr>
<tr>
<td>iucv</td>
<td>IUCV</td>
<td>netiucv</td>
<td>IUCV authorizations are required</td>
</tr>
</tbody>
</table>

This table is intended as an overview only. For details about which version of a particular hardware is supported by a device driver, see the applicable section about the device driver.

Multiple network device drivers 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 to keep track of the mapping differs depending on the network device driver.

For qeth, you can use the lsqeth command (see “lsqeth - List qeth-based network devices” on page 669) 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 network devices drivers that are based on ccwgroup devices (for example, qeth and lcs devices). How to perform a particular step can be different for the different device drivers.

The main steps are:

1. Create a network device by combining suitable subchannels into a group device. The device driver then creates directories that represent the device in sysfs.
2. Configure the device through its attributes in sysfs. See “Device views in sysfs” on page 10. Some devices have attributes that can or must be set later when the device is online or when the connection is active.
3. Set the device online. This step associates the device with an interface name and thus makes the device known to the Linux network stack. For devices that are associated with a physical network adapter it also initializes the adapter for the network interface.
4. Configure and activate the interface. This step adds interface properties like IP addresses, netmasks, and MTU to the network interface and moves the network interface into state "up". The interface is then ready for user space (socket) programs to run connections and transfer data across it.

To configure devices, use the chzdev command, see Chapter 29, “Persistent device configuration,” on page 377.
Chapter 2. Devices in sysfs

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

Device categories

The /sys/devices directory includes several device categories that are specific to z/Architecture. Figure 4 on page 7 illustrates a part of sysfs.

Figure 4: sysfs

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

**AP devices**
- are adjunct processors used for cryptographic operations.

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

<table>
<thead>
<tr>
<th>Device driver</th>
<th>Category</th>
<th>sysfs directories</th>
</tr>
</thead>
<tbody>
<tr>
<td>3215 console</td>
<td>CCW</td>
<td>/sys/bus/ccw/drivers/3215</td>
</tr>
<tr>
<td>3270 console</td>
<td>CCW</td>
<td>/sys/bus/ccw/drivers/3270</td>
</tr>
<tr>
<td>DASD</td>
<td>CCW</td>
<td>/sys/bus/ccw/drivers/dasd-eckd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/sys/bus/ccw/drivers/dasd-fba</td>
</tr>
<tr>
<td>SCSI-over-Fibre Channel</td>
<td>CCW</td>
<td>/sys/bus/ccw/drivers/zfcp</td>
</tr>
<tr>
<td>Storage class memory supporting Flash Express</td>
<td>SCM</td>
<td>/sys/bus/scm/</td>
</tr>
<tr>
<td>Channel-attached tape</td>
<td>CCW</td>
<td>/sys/bus/ccw/drivers/tape_34xx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/sys/bus/ccw/drivers/tape_3590</td>
</tr>
<tr>
<td>Cryptographic</td>
<td>AP</td>
<td>/sys/bus/ap/drivers/cpixcccard</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/sys/bus/ap/drivers/cpixccqueue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/sys/bus/ap/drivers/cex2acard</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/sys/bus/ap/drivers/cex2acqueue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/sys/bus/ap/drivers/cex4card</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/sys/bus/ap/drivers/cex4queue</td>
</tr>
<tr>
<td>DCSS</td>
<td>n/a</td>
<td>/sys/devices/dcsslblk</td>
</tr>
<tr>
<td>XPRAM</td>
<td>n/a</td>
<td>/sys/devices/system/xpram</td>
</tr>
<tr>
<td>z/VM recording</td>
<td>IUCV</td>
<td>/sys/bus/iucv/drivers/vmlogrdr</td>
</tr>
<tr>
<td>qeth (OSA-Express features and HiperSockets)</td>
<td>CCW group</td>
<td>/sys/bus/ccwgroup/drivers/qeth</td>
</tr>
<tr>
<td>LCS</td>
<td>CCW group</td>
<td>/sys/bus/ccwgroup/drivers/lcs</td>
</tr>
<tr>
<td>CTCM</td>
<td>CCW group</td>
<td>/sys/bus/ccwgroup/drivers/ctcm</td>
</tr>
<tr>
<td>NETIUCV</td>
<td>IUCV</td>
<td>/sys/bus/iucv/drivers/netiucv</td>
</tr>
<tr>
<td>10GbE RoCE Express devices for Mellanox ConnectX-3 EN (mlx4_en)</td>
<td>PCI</td>
<td>sys/bus/pci/drivers/mlx4_core</td>
</tr>
<tr>
<td>10 GbE RoCE Express2 devices for Mellanox ConnectX-4 EN (mlx5_core)</td>
<td>PCI</td>
<td>sys/bus/pci/drivers/mlx5_core</td>
</tr>
</tbody>
</table>
Some device drivers do not relate to physical devices that are connected through the channel subsystem. Their representation in sysfs differs from the CCW and CCW group devices, for example, the 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 10 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 540 for details.

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

- **availability**
  
  Indicates whether the device can be used. The following values are possible:

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

  - **boxed**
    
    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 154).

  - **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>
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 chzdev or the chccwdev command (see “chzdev - Configure IBM Z devices” on page 589 and “chccwdev - Set CCW device attributes” on page 576) to set attributes.

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

Working with newly available devices

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

About this task

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

Procedure

Use the following steps before you work with a newly available device to avoid such errors:
1. Attach the device, for example, with a z/VM CP ATTACH command.
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 576) 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” on page 11
- “Device category view” on page 11
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 8).

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

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

**Examples**

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

**Device category view**

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

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

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

where:

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

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

**Note:** DCSSs 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.
Device view

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

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

where:

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

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

Examples

• This example shows the path for a qeth device.
  /sys/devices/qeth/0.0.a100

• This example shows the path for a DCSS block device.
  /sys/devices/dcssblk/mydcss

Channel subsystem view

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

The channel subsystem view is of the form:
/sys/devices/css0/<subchannel>

where:

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

I/O subchannels show the devices in relation to their respective subchannel sets and subchannels. An I/O subchannel is of the form:
/sys/devices/css0/<subchannel>/<device_bus_id>

where:

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

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

Examples

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

• This example shows a CCW device with device number 0xb200 that is associated with a subchannel 0x0001 in subchannel set 1.
  /sys/devices/css0/0.1.0001/0.1.b200
• The entries for a group device show as separate subchannels. If a CCW group device uses three subchannels 0x0002, 0x0003, and 0x0004 the subchannel information could be:

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

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

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

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

## Subchannel attributes

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

Subchannels have two common attributes:

**type**

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

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

**modalias**

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

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

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

**chpids**

is a list of the channel-path identifiers (CHPIDs) through with the device is connected. See also “Channel path ID information” on page 14.

**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**

Issue a command of this form:

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

<CHPID>  
is a two digit hexadecimal CHPID.

<value>  
is either on or off.

Examples

• To set a CHPID 0x4a logically offline issue:

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

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

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

• To set the same CHPID logically online issue:

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

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

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

Configuring a CHPID on LPAR

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

About this task

The following configuration changes are supported:

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

Procedure

Issue a command of this form:

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

where:

<CHPID>  
is a two digit hexadecimal CHPID.

<value>  
is either 1 or 0.

To query and set the configure value using commands, see “chchp - Change channel path status” on page 578 and “lschp - List channel paths” on page 654.

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:

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

- To set the same CHPID to configured issue:

  ```bash
  # 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:

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

**Finding the physical channel associated with a CHPID**

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

**About this task**

A CHPID is associated with either a physical port or with an internal connection defined inside the mainframe, such as HiperSockets. See Figure 5 on page 16. You can determine the PCHID or internal channel ID number that is associated with a CHPID number.

![Diagram](image)

*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:

```bash
# 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 “-”.

```bash
# 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    -    -       0700
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:

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

Mainframes as of z14 can store device configuration data for Linux in LPAR mode on the Support Element (SE).

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

Making devices available to Linux

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

Defining devices to an LPAR

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

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

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

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

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

Controlling device availability on Linux

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

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

To enforce the current cio_ignore list, you can use the \texttt{rd.zdev=no-auto} kernel parameter to disregard auto-configuration for devices on Linux.

Configuring devices on Linux

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

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

Persistent configuration
  The configuration to be applied when the Linux instance is booted.
DPM only: Auto-configuration

The configuration as specified on the HMC interface.

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

Overriding the auto-configuration

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

For devices that come online early in the boot process, use the zdev:early device attribute to ensure that this persistent configuration is available at this early stage (see “chzdev - Configure IBM Z devices” on page 589).

Managing auto-configuration data

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

Displaying auto-configuration data

The lszdev command can display auto-configuration data.

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

Example:

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

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

Example:

```
# lszdev
TYPE         ID                          ON   PERS    NAMES
dasd-eckd    0.0.ec30                    yes  yes    dasda
            0.0.ec31                    yes  auto   dasdb
dasd-eckd    0.0.ec32                    yes  no     dasdc
qeth         0.0.f5f0:0.0.f5f1:0.0.f5f2  yes  no     enccw0.0.f5f0
```

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

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

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

Modifying the auto-configuration

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

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

Example:

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

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

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

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

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

Example:

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

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

Example:

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

**Example:**

```
@ lszdev --by-attr zdev:early=1
 TYPE     ID                                             ON  PERS NAMES
--- --- ------------------------ ------ ------ -------
 dasd-fba 0.0.e030                                       yes yes  dasda
 zfcp-lun 0.0.1911:0x50050763070845e3:0x4082409f00000000 no  yes
```
Chapter 4. Kernel and module parameters

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

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

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

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

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 66, “Selected kernel parameters,” on page 757. 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 60 for details.

A boot configuration can include up to 895 characters of kernel parameters. See also “How kernel parameters from different sources are combined” on page 26.
**Running zipl in configuration-file mode**

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

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

As shown in Figure 6 on page 24, there are three sources of kernel parameters for `zipl` in configuration-file mode.

![Figure 6: Sources of kernel parameters for zipl in configuration-file mode](image)

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 60 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 7 on page 24, there are two sources of kernel parameters for `zipl` in command-line mode.

![Figure 7: Sources of kernel parameters for zipl in command-line mode](image)

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 60 for details about the `zipl` command modes.
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 85 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 83 and “Example for a DASD menu configuration (LPAR)” on page 88 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, and the z/VM reader.
For details, see the subsection of “Booting Linux in a z/VM guest virtual machine” on page 81 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 84 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 31 on page 90.

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 119 and “Perform an IPL or dump operation from a SCSI device” on page 120.

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 27).

How kernel parameters from different sources are combined

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

1. Kernel parameters that have been included in the boot configuration (see “Including kernel parameters in a boot configuration” on page 23).
   The kernel parameters in the boot configuration cannot exceed 895 characters. If more then 895 characters are specified, the excessive characters are truncated.
2. Kernel parameters that you specify through the HMC or through z/VM interfaces (see “Specifying kernel parameters when booting Linux” on page 25).
   For DASD boot devices you can specify up to 64 characters (z/VM only); for SCSI boot devices you can specify up to 3452 characters.

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

Multiple specifications for the same parameter

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

Conflicting kernel parameters

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

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

Examples for kernel parameters
Typical parameters that are used for booting Linux on Z configure the console and the root file system.

**conmode=<mode>, condev=<cuu>, console=<name>**
- to set up the Linux console. See “Console kernel parameter syntax” on page 41 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 8, “Suspending and resuming Linux,” on page 99).

**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 28).

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 95 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 96 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. Optional: Confirm that the module of interest is loaded by issuing a command of this form:

   # lsmod | grep <module_name>

   where `<module_name>` is the name of the module.

2. Optional: Get an overview of the parameters for the module by issuing a command of this form:

   # modinfo <module_name>

3. 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

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

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

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

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

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

  # cat /sys/module/ap/parameters/domain
  1
Part 2. Booting and shutdown

These device drivers and features are useful in the context of booting and shutting down instances of Linux on 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
Chapter 5. Console device drivers

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

The only interface to a Linux instance in an LPAR before the boot process is completed is the Hardware Management Console (HMC), see Figure 8 on page 33. 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.

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 9 on page 34 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.

---

What you should know about the console device drivers
The console concepts, naming conventions, and terminology overview help you to understand the tasks you might have to perform with console and terminal devices.

Console terminology
Terminal and console have special meanings in Linux.

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

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

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

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

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

**Before you have a Linux terminal - the zipl boot menu**
Do not confuse the zipl boot menu with a Linux terminal.

Depending on your setup, a zipl boot menu might be displayed when you perform an IPL. The zipl boot menu is part of the boot loader that loads the Linux kernel and is displayed before a Linux terminal is set up. The zipl boot menu is very limited in its functions. For example, there is no way to specify uppercase letters because all input is converted to lowercase characters. For more details about booting Linux, see Chapter 7, “Booting Linux,” on page 79. For more information about the zipl boot menu, see Chapter 6, “Initial program loader for IBM Z - zipl,” on page 59.

**Device and console names**
Each terminal device driver can provide a single console device.

Table 4 on page 35 lists the terminal device drivers and the corresponding device names and console names.

<table>
<thead>
<tr>
<th>Device driver</th>
<th>Device name</th>
<th>Console name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCLP line-mode terminal device driver</td>
<td>sclp_line0</td>
<td>ttyS0</td>
</tr>
<tr>
<td>SCLP VT220 terminal device driver</td>
<td>ttysclp0</td>
<td>ttyS1</td>
</tr>
<tr>
<td>3215 line-mode terminal device driver</td>
<td>ttyS0</td>
<td>ttyS0</td>
</tr>
<tr>
<td>3270 terminal device driver</td>
<td>3270/tty1 to 3270/tty&lt;N&gt;</td>
<td>tty3270</td>
</tr>
<tr>
<td>z/VM IUCV HVC device driver</td>
<td>hvc0 to hvc7</td>
<td>hvc0</td>
</tr>
</tbody>
</table>

As shown in Table 4 on page 35, 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 41.

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 5 on page 36 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 5: Terminal device nodes

<table>
<thead>
<tr>
<th>Device driver</th>
<th>On LPAR</th>
<th>On z/VM</th>
<th>Major</th>
<th>Minor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCLP line-mode terminal device</td>
<td>/dev/sclp_line0</td>
<td>/dev/sclp_line0</td>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>driver</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCLP VT220 terminal device</td>
<td>/dev/ttysclp0</td>
<td>/dev/ttysclp0</td>
<td>4</td>
<td>65</td>
</tr>
<tr>
<td>driver</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3215 line-mode terminal device</td>
<td>n/a</td>
<td>/dev/ttyS0</td>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>driver</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3270 terminal device driver</td>
<td>/dev/3270/tty1</td>
<td>/dev/3270/tty1</td>
<td>227</td>
<td>1 - &lt;N&gt;</td>
</tr>
<tr>
<td>to /dev/3270/tty&lt;N&gt;</td>
<td></td>
<td>/dev/3270/tty1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z/VM IUCV HVC device driver</td>
<td>n/a</td>
<td>/dev/hvc0 to /dev/hvc7</td>
<td>229</td>
<td>0 - 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 6 on page 36 summarizes when to expect which terminal mode.

Table 6: Terminal modes

<table>
<thead>
<tr>
<th>Accessed through</th>
<th>Environment</th>
<th>Device driver</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>LPAR</td>
<td>SCLP line-mode terminal device driver</td>
<td>Line mode</td>
</tr>
<tr>
<td>Messages applet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>on the HMC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z/VM emulation</td>
<td>z/VM</td>
<td>SCLP line-mode terminal device driver</td>
<td>Line mode</td>
</tr>
<tr>
<td>Operating System</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Messages applet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z/VM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrated ASCII</td>
<td>z/VM or LPAR</td>
<td>SCLP VT220 terminal device driver</td>
<td>Full-screen mode</td>
</tr>
<tr>
<td>Console applet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>on the HMC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3270 terminal</td>
<td>z/VM with</td>
<td>3215 line-mode terminal device driver</td>
<td>Line mode</td>
</tr>
<tr>
<td>hardware or</td>
<td>CONMODE=3215</td>
<td></td>
<td></td>
</tr>
<tr>
<td>emulation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 6: Terminal modes (continued)

<table>
<thead>
<tr>
<th>Accessed through</th>
<th>Environment</th>
<th>Device driver</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>3270 terminal hardware or</td>
<td>z/VM with</td>
<td>3270 terminal device driver</td>
<td>Block mode</td>
</tr>
<tr>
<td>emulation</td>
<td>CONMODE=3270</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iucvconn program</td>
<td>z/VM</td>
<td>z/VM IUCV HVC device driver</td>
<td>Full-screen mode</td>
</tr>
</tbody>
</table>

The 3270 terminal device driver provides three different views. See “Switching the views of the 3270 terminal device driver” on page 51 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 10 on page 37 shows the possible terminal devices for Linux instances that run directly in an LPAR.

Figure 10: 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 11 on page 38).
Use the CP ATTACH SYSASCII command to attach the ASCII system console to your z/VM guest virtual machine.

**Using a 3270 terminal emulation**

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

Figure 11 on page 38 illustrates how z/VM can handle the 3270 communication.

**CONMODE=3270**

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

You can use the conmode= kernel parameter to make the kernel issue the corresponding z/VM CP command.

**CONMODE=3215**

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

You can use the conmode= kernel parameter to make the kernel issue the corresponding z/VM CP command.

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

For information about the conmode= kernel parameter, see “Console kernel parameter syntax” on page 41.

**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.

![Diagram: Accessing terminal devices from a peer Linux instance](image)

*Figure 13: Accessing terminal devices from a peer Linux instance*

As illustrated in Figure 13 on page 39, 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 14 on page 40 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)**
- **S/390 character device drivers (depends on S390)**
  - Support for locally attached 3270 terminals
  - Support for tty input/output on 3270 terminals
  - Support for full-screen applications on 3270 terminals
  - Support for console on 3270 terminal
  - Support for 3215 line mode terminal
  - Support for console on 3215 line mode terminal
  - Support for SCLP line mode terminal
  - Support for console on SCLP line mode terminal
  - Support for SCLP VT220-compatible terminal
  - Support for console on SCLP VT220-compatible terminal

**Figure 14: Console kernel configuration menu options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIG_HVC_IUCV</td>
<td>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.</td>
</tr>
<tr>
<td>CONFIG_TN3270</td>
<td>Adds the 3270 terminal device driver. This device driver supports IBM 3270 terminal hardware and 3270 terminal emulations.</td>
</tr>
<tr>
<td>CONFIG_TN3270_TTY</td>
<td>Adds the terminal input and output support to the 3270 terminal device driver.</td>
</tr>
<tr>
<td>CONFIG_TN3270_FS</td>
<td>Adds limited support for full-screen applications the 3270 terminal device driver.</td>
</tr>
<tr>
<td>CONFIG_TN3270_CONSOLE</td>
<td>Adds the operating system messages view to the 3270 terminal device driver.</td>
</tr>
<tr>
<td>CONFIG_TN3215</td>
<td>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.</td>
</tr>
<tr>
<td>CONFIG_TN3215_CONSOLE</td>
<td>Adds support for kernel messages to the 3215 line-mode terminal device driver.</td>
</tr>
<tr>
<td>CONFIG_SCLP_TTY</td>
<td>Adds the SCLP line-mode terminal device driver. This device driver supports the Operating System Messages applet on the HMC.</td>
</tr>
<tr>
<td>CONFIG_SCLP_CONSOLE</td>
<td>Adds support for kernel messages to the SCLP line-mode terminal device driver.</td>
</tr>
<tr>
<td>CONFIG_SCLP_VT220_TTY</td>
<td>Adds the VT220-compatible SCLP VT220 terminal device driver. This device driver supports the Integrated ASCII console applet on the HMC.</td>
</tr>
<tr>
<td>CONFIG_SCLP_VT220_CONSOLE</td>
<td>Adds support for kernel messages to the SCLP VT220 terminal device driver.</td>
</tr>
</tbody>
</table>

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.

**Console kernel parameter syntax**

![Diagram of console kernel parameter syntax]

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 7 on page 42 shows how the device driver that is used by default depends on the environment.
Table 7: Default device driver for the line-mode terminal device

<table>
<thead>
<tr>
<th>Mode</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPAR</td>
<td>SCLP line-mode terminal device driver</td>
</tr>
<tr>
<td>z/VM</td>
<td>3215 line-mode terminal device driver or 3270 terminal device driver, depending on the z/VM guest's console settings (the CONMODE field in the output of #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.</td>
</tr>
</tbody>
</table>

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

**sclp or hwc**

specifies the SCLP line-mode terminal device driver.

You need this specification if you want to use the z/VM CP VINPUT command ("Using a z/VM emulation of the HMC Operating System Messages applet" on page 55).

**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 initialization procedures run. 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 4 on page 35.

**condev=<devno>**

specifies the CCW device to be chosen from several defined 3270 consoles. By default the first device that is found is chosen. Omit this parameter if there is only one device, or if any device is acceptable.

**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.
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 `chiuvcvalw` 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 iucvcon**

Because the iucvcon 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 346 for details about setting up the z/VM guest virtual machine.
For information about accessing Linux through the iucvty 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 iucvty and iucvconn commands.

**Setting up a line-mode terminal**

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

The preferred user access to a running instance of Linux on Z is through a user login that runs, for example, in an ssh session. See “Terminal modes” on page 36 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:

```bash
# export TERM=dumb
```

**Setting up a full-screen mode terminal**

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

See “Terminal modes” on page 36 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:

```bash
# 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:

```bash
# 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 41).
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 35). 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 44. 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 35 for the device node names. If an existing entry uses a name that you cannot map to the names of “Device nodes” on page 35, 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 35). 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 44. 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]
respm
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]
respm
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 49). 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 41.

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.

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 35). 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.
- `%t` is a variable that the ttyrun program resolves to the device node that is specified for `<dev>`.

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

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 41). 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 35). 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 44.

**%t**

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

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

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 41). 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 41).

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:

   ```bash
   # 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:

   ```bash
   # 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:

```bash
# 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:

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

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

- “Using the terminal applets on the HMC” on page 50
- “Accessing terminal devices over z/VM IUCV” on page 50
- “Switching the views of the 3270 terminal device driver” on page 51
- “Setting a CCW terminal device online or offline” on page 52
- “Entering control and special characters on line-mode terminals” on page 53
- “Using the magic sysrequest feature” on page 53
- “Using a z/VM emulation of the HMC Operating System Messages applet” on page 55
- “Using a 3270 terminal in 3215 mode” on page 57

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 guest virtual machines must be configured such that IUCV communication is permitted between them.

Procedure

Perform these steps to access an HVC terminal device over z/VM IUCV:
1. Open a terminal session on the Linux instance where the iucvconn program is installed.

2. Enter a command of this form:

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

where:

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

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

**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 15 on page 51).

![Figure 15: Switching views of the 3270 terminal device driver](image)

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 39). 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*, SC24-6268 for more information about the DEF GRAF and DIAL commands.

**Procedure**

You can use the `chccwdv` command (see “chccwdv - Set CCW device attributes” on page 576) 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:
  ```bash
  # chccwdv -e 0.0.7b01
  ```
  Alternatively issue:
  ```bash
  # echo 1 > /sys/bus/ccw/drivers/3270/0.0.7b01/online
  ```

- To set a CCW terminal device `0.0.7b01` offline, issue:
  ```bash
  # chccwdv -d 0.0.7b01
  ```
  Alternatively issue:
  ```bash
  # 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 8 on page 53 summarizes how you can use the caret character (^) to enter some control characters and to enter strings without appended newline characters.

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

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

Using the magic sysrequest feature

Whether magic sysrequest functions are available and how to call them depends on your terminal.

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 the VT220 terminal or on hvc0, enter the single character Ctrl +o followed by the character for the particular function.
  You can call the magic sysrequest functions from the hvc0 terminal device if it is present and is activated to receive Linux kernel messages.
- To call the magic sysrequest functions on a line-mode terminal, enter the 2 characters "^-" (caret and hyphen) followed by a third character that specifies the particular function.

Table 9 on page 53 provides an overview of the commands for the magic sysrequest functions:

<table>
<thead>
<tr>
<th>On line-mode terminals, enter</th>
<th>On hvc0 and the VT220 terminal, enter</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>^-b</td>
<td>Ctrl+o b</td>
<td>Re-IPL immediately (see “lsreipl - List IPL and re-IPL settings” on page 670).</td>
</tr>
<tr>
<td>^-c</td>
<td>Ctrl+o c</td>
<td>Crash through a forced kernel panic.</td>
</tr>
<tr>
<td>^-s</td>
<td>Ctrl+o s</td>
<td>Emergency sync all file systems.</td>
</tr>
<tr>
<td>^-u</td>
<td>Ctrl+o u</td>
<td>Emergency remount all mounted file systems read-only.</td>
</tr>
<tr>
<td>^-t</td>
<td>Ctrl+o t</td>
<td>Show task info.</td>
</tr>
<tr>
<td>^-m</td>
<td>Ctrl+o m</td>
<td>Show memory.</td>
</tr>
<tr>
<td>On line-mode terminals, enter</td>
<td>On hvc0 and the VT220 terminal, enter</td>
<td>To</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------</td>
<td>----</td>
</tr>
<tr>
<td>^- followed by a digit (0 - 9)</td>
<td>Ctrl+o followed by a digit (0 - 9)</td>
<td>Set the console log level.</td>
</tr>
<tr>
<td>^-e</td>
<td>Ctrl+o e</td>
<td>Send the TERM signal to end all tasks except init.</td>
</tr>
<tr>
<td>^-i</td>
<td>Ctrl+o i</td>
<td>Send the KILL signal to end all tasks except init.</td>
</tr>
<tr>
<td>^-p</td>
<td>Ctrl+o p</td>
<td>See “Obtaining details about the CPU-measurement facilities” on page 548.</td>
</tr>
</tbody>
</table>

**Note:** In Table 9 on page 53 `Ctrl+o` means pressing while holding down the control key.

Table 9 on page 53 lists the main magic sysrequest functions that are known to work on . For a more comprehensive list of functions, see `Documentation/sysrq.txt` in the Linux source tree. Some of the listed functions might not work on your system.

**Activating and deactivating the magic sysrequest feature**

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

**Procedure**

Issue the following command to activate the magic sysrequest feature:

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

Issue the following command to deactivate the magic sysrequest feature:

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

**Triggering magic sysrequest functions from procfs**

You can trigger the magic sysrequest functions through procfs.

**Procedure**

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

You can use this interface even if the magic sysrequest feature is not activated as described in “Activating and deactivating the magic sysrequest feature” on page 54.

**Example**

To set the console log level to 9, enter:

```bash
# 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 41).

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

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

Priority and non-priority commands

VINPUT commands require a VMSG (non-priority) or PVMSG (priority) specification. Operating systems that accept this specification, process priority commands with a higher priority than non-priority commands.

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

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

Example

The specifications:

#CP VINPUT VMSG LS -L

and

#CP VINPUT PVMSG LS -L

are equivalent.

Case conversion

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

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

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

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

```bash
#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:

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

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

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 57, 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.

```bash
#cp vinput pvmsg echo ""%#Number signs start bash comments"" #like this one
CP VINPUT PVMSG ECHO ""%#NUMBER SIGNS START BASH COMMENTS"" LIKE THIS ONE
HCPMD001E Unknown CP command: LIKE
```

The escape character prevents the number sign from being interpreted as an end-of-line character:
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 10 on page 57 shows the most commonly used settings:

<table>
<thead>
<tr>
<th>Character</th>
<th>Symbol</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>LINEND</td>
<td>The end of line character. With this character, you can enter several logical lines at once.</td>
</tr>
<tr>
<td></td>
<td>TABCHAR</td>
<td>The logical tab character.</td>
</tr>
<tr>
<td>@</td>
<td>CHARDEL</td>
<td>The character delete symbol deletes the preceding character.</td>
</tr>
<tr>
<td></td>
<td>LINEDEL</td>
<td>The line delete symbol deletes everything back to and including the previous LINEND symbol or the start of the input. &quot;[&quot; is common for ASCII terminals and &quot;;&quot; for EBCDIC terminals.</td>
</tr>
<tr>
<td>&quot;</td>
<td>ESCAPE</td>
<td>The escape character. With this character, you can enter a line-edit symbol as a normal character.</td>
</tr>
</tbody>
</table>

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 10 on page 57 with the opening square bracket character (\[) as the "delete line" character.

- To specify a tab character specify:

```
"|
```

- To specify a double quotation mark character, specify:

```
"
```

- If you type the character string:

```
#CP HALT#CP ZIPL 190[#CP IPL 18290 PARM vmpoff="#MSG OP REBOOT"#IPL 290"
```

the actual commands that are received by CP are:

```
CP HALT
CP IPL 290 PARM vmpoff="#MSG OP REBOOT#IPL 290"
```
Chapter 6. Initial program loader for IBM Z - zipl

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

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 for Linux on z/VM and Linux in LPAR mode:

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

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

Usage

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

zipl base functions

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

Table 11: zipl base functions

<table>
<thead>
<tr>
<th>Base function</th>
<th>Command line short option</th>
<th>Command line long option</th>
<th>Configuration file option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install a boot loader</td>
<td>-i</td>
<td>--image</td>
<td>image=</td>
</tr>
<tr>
<td>Prepare a DASD, SCSI, or tape dump device</td>
<td>-d</td>
<td>--dumpto</td>
<td>dumpto=</td>
</tr>
<tr>
<td>Prepare a list of ECKD volumes for a multi-volume dump</td>
<td>-M</td>
<td>--mvdump</td>
<td>mvdump=</td>
</tr>
<tr>
<td>Install a menu configuration</td>
<td>-m</td>
<td>--menu</td>
<td>(None)</td>
</tr>
</tbody>
</table>
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 70), 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 62
- “Preparing a DASD, SCSI, or tape dump device” on page 67
- “Preparing a multi-volume dump on ECKD DASD” on page 69

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 74.

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 74).
3. In a boot configuration only.
4. In a boot configuration or a menu configuration only.

Figure 16: zipl syntax overview

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 64 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 71 for a summary of the short and long command line options and their configuration file equivalents.

**Examples**

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

  ```
  # zipl
  ```

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

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

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

  ```
  # zipl myconf
  ```

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

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

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

  ```
  # zipl --dry-run -c /etc/myxmp.conf myconf
  ```
Preparing a boot device

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

```
zipl -i <image> -t <directory> -T <tape_node> -r <ramdisk>,<initrd_addr> -p <parmfile>,0x1000,<parm_addr> -P <parameters> -k auto
```

**Notes:**
1. Additional parameters that are used only if `-t` specifies a logical device as a target. See “Using base device parameters” on page 65.

*Figure 17: zipl command line syntax for preparing a boot device*

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 64 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.

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

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

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

Figure 18 on page 63 summarizes how to specify a boot configuration within a configuration file section.
Required specifications are shown in bold. See “Configuration file structure” on page 74 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 18: 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
parmfile=/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 23.

Preparing a logical device as a boot device

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

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

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

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

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

![Figure 19: Definition of base device](image)

A base device can have the following mappings:

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

The **zipl** command needs the device node of the base device and information about the physical characteristics of the underlying real devices. For most logical boot devices, a helper script automatically provides all the required information to zipl for you (see “Using a helper script” on page 65).

If you decide not to use the supplied helper script, or want to write your own helper script, you can use parameters to supply the base device information to zipl, see “Using base device parameters” on page 65 and “Writing your own helper script” on page 67.

**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 62 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 62.

```
--targetbase <targetbase_node>
--targettype LDL, CDL
--targetgeometry <cylinders>,<heads>,<sectors>
--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`. 

---

Initial program loader for IBM Z - zipl 65
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 20 on page 66 shows how to specify this information in a configuration file.

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

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

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

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

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

• The script must accept the name of the target directory as an argument. From this specification, it must determine a suitable base device. See “Using base device parameters” on page 65.

• The script must write the following base device `<parameter>=<value>` pairs to stdout as ASCII text. Each pair must be written on a separate line.

  - `targetbase=<targetbase_node>`
  - `targettype=<type>` where type can be LDL, CDL, FBA, or SCSI.
  - `targetgeometry=<cylinders>,<heads>,<sectors>` (For LDL and CDL only)
  - `targetblocksize=<blocksize>`
  - `targetoffset=<offset>`

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

• The script must be named `zipl_helper.<device>` where `<device>` is the device name as specified in `/proc/devices`.

• The script must be in `/lib/s390-tools`.

Preparing a 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.

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 71 for a summary of the parameters. The summary includes the long options that you can use on the command line.

Figure 22 on page 68 summarizes how to specify a DASD, SCSI, or tape dump configuration in a configuration file. See “Configuration file structure” on page 74 for a more comprehensive discussion of the configuration file.

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

*Figure 22: 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/36005076303ff4010000000000000020c0-part1` as a dump device:

```
# zipl -d /dev/mapper/36005076303ff4010000000000000020c0-part1
```

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

```
[dumpscai]
dumpto=/dev/mapper/36005076303ff4010000000000000020c0-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.

```
[54x733]Preparing a multi-volume dump on ECKD DASD
```

Figure 23: zipl command line syntax for preparing devices for a multi-volume dump

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

-M `<dump_device_list>`

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

Optionally, you can also specify:

-f or --force

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

-n

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

<size>

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

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

DASD or tape dump devices are not formatted with a file system so no target directory can be specified. See Using the Dump Tools, SC33-8412 for details about processing these dumps.

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

Figure 24 on page 69 summarizes how to specify a multi-volume DASD dump configuration in a configuration file. See “Configuration file structure” on page 74 for a more comprehensive discussion of the configuration file.

```
[<section_name>]
  mvdump=<dump_device_list>,<size>
```

Figure 24: zipl syntax for preparing DASD devices for a multi-volume dump - configuration file mode

Example

The following command prepares two DASD partitions /dev/dasdc1, /dev/dasdd1 for a multi-volume dump and suppresses confirmation prompts that require an operator response:

```
# zipl -M mvdump.conf -n
```
where the `mvdump.conf` file contains the two partitions, separated by line breaks:

```
/dev/dasdc1
/dev/dasdd1
```

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

```
[multi_volume_dump]
mvdump=mvdump.conf
```

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

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

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

## Installing a menu configuration

Use `zipl` with the `-m` (`--menu`) command-line option to install a menu configuration.

To prepare a menu configuration, you need a configuration file that includes at least one menu section (see “Menu configurations” on page 76) or with a default section that supports an automatic menu (see “Default section” on page 74).

```
zipl -m <menu_name>
  -c /etc/zipl.conf
  -c <config_file>
  -a
```

Notes:
1. You can change the default configuration file with the ZIPLCONF environment variable.

*Figure 25: zipl syntax for installing a menu configuration*

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 26 on page 77*, 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.

<table>
<thead>
<tr>
<th>Command line short option</th>
<th>Command line long option</th>
<th>Configuration file option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>--add-files</td>
<td>n/a</td>
<td>Causes kernel image, kernel parameter file, and initial RAM disk to be added to the bootstrap file in the target directory rather than being referenced from this file.</td>
</tr>
<tr>
<td>-c &lt;config_file&gt;</td>
<td>--config=&lt;config_file&gt;</td>
<td>n/a</td>
<td>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 bootstrap file that is created in the target directory.</td>
</tr>
<tr>
<td>-d &lt;dump_device&gt;[,&lt;size&gt;]</td>
<td>--dumpto=&lt;dump_device&gt;[,&lt;size&gt;]</td>
<td>n/a</td>
<td>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 67 and Using the Dump Tools, SC33-8412.</td>
</tr>
<tr>
<td>-h</td>
<td>--help</td>
<td>n/a</td>
<td>Displays help information.</td>
</tr>
<tr>
<td>-i &lt;image&gt;[,&lt;image_addr&gt;]</td>
<td>--image=&lt;image&gt;[,&lt;image_addr&gt;]</td>
<td>n/a</td>
<td>Specifies the location of the Linux kernel image on the file system and, optionally, in memory after IPL. The default memory address is 0x10000.</td>
</tr>
</tbody>
</table>

See “Preparing a boot device” on page 62 for details.
<table>
<thead>
<tr>
<th>Command line short option</th>
<th>Command line long option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-k auto</td>
<td>--kdump=auto</td>
<td>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 62 for details.</td>
</tr>
<tr>
<td>-m &lt;menu_name&gt;</td>
<td>--menu=&lt;menu_name&gt;</td>
<td>Specifies the name of the menu that defines a menu configuration in the configuration file (see “Menu configurations” on page 76).</td>
</tr>
<tr>
<td>-M &lt;dump_device_list&gt;[,&lt;size&gt;]</td>
<td>--mvdump=&lt;dump_device_list&gt;[,&lt;size&gt;]</td>
<td>Specifies a file with a list of DASD partitions to which a dump is to be written after IPL. The optional size specification limits the amount of memory to be dumped. The value is a decimal number that can optionally be suffixed with K for kilobytes, M for megabytes, or G for gigabytes. The value is rounded to the next megabyte boundary. If you limit the dump size below the amount of memory that is used by the system to be dumped, the resulting dump is incomplete. If no limit is provided, all of the available physical memory is dumped. See “Preparing a multi-volume dump on ECKD DASD” on page 69 and Using the Dump Tools, SC33-8412 for details.</td>
</tr>
<tr>
<td>-n</td>
<td>--noninteractive</td>
<td>Suppresses all confirmation prompts (for example, when preparing a DASD or tape dump device).</td>
</tr>
<tr>
<td>-p &lt;parmfile&gt;[,&lt;parm_addr&gt;]</td>
<td>--parmfile=&lt;parmfile&gt;[,&lt;parm_addr&gt;]</td>
<td>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 64. The optional &lt;parm_addr&gt; specifies the memory address where the combined kernel parameter list is to be loaded at IPL time.</td>
</tr>
</tbody>
</table>
## Command line short option
### Command line long option

<table>
<thead>
<tr>
<th>Configuration file option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-P &lt;parameters&gt;</td>
<td>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 (&quot;). You can specify multiple sources of kernel parameters. For more information, see “How kernel parameters from different sources are combined” on page 64.</td>
</tr>
<tr>
<td>-r &lt;ramdisk&gt;[,&lt;initrd_addr&gt;]</td>
<td>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.</td>
</tr>
<tr>
<td>-t &lt;directory&gt;</td>
<td>Specifies the target directory where zipl creates boot-relevant files. The boot loader is installed on the disk that contains the target directory.</td>
</tr>
<tr>
<td>none</td>
<td>For logical boot devices, specifies the device node of the base device, either by using the standard device name or in form of the major and minor number, separated by a colon (:). See “Using base device parameters” on page 65 for details.</td>
</tr>
<tr>
<td>none</td>
<td>For logical boot devices, specifies the bytes per block of the base device. See “Using base device parameters” on page 65 for details.</td>
</tr>
<tr>
<td>none</td>
<td>For logical boot devices that map to ECKD type base devices, specifies the disk geometry of the base device in cylinders, heads, and sectors. See “Using base device parameters” on page 65 for details.</td>
</tr>
<tr>
<td>none</td>
<td>For logical boot devices, specifies the offset in blocks between the start of the physical device and the start of the logical device. See “Using base device parameters” on page 65 for details.</td>
</tr>
<tr>
<td>none</td>
<td>For logical boot devices, specifies the device type of the base device. See “Using base device parameters” on page 65 for details.</td>
</tr>
</tbody>
</table>
### Command line short option

**Command line long option**

### Configuration file option

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-T &lt;tape_node&gt;</td>
<td>Specifies the tape device where zipl installs the boot loader code.</td>
</tr>
<tr>
<td>--tape=&lt;tape_node&gt;</td>
<td></td>
</tr>
<tr>
<td>tape=&lt;tape_node&gt;</td>
<td></td>
</tr>
<tr>
<td>-v</td>
<td>Prints version information.</td>
</tr>
<tr>
<td>--version</td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>-V</td>
<td>Provides more detailed command output.</td>
</tr>
<tr>
<td>--verbose</td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

If you call zipl in configuration file mode without specifying a configuration file, the default /etc/zipl.conf is used. You can change the default configuration file with the environment variable ZIPLCONF.

### Configuration file structure

A configuration file 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 76.

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 64).

**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 62 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 67.

- **mvdump=<dump_device_list>**
  Defines a multi-volume DASD dump configuration. See “Preparing a multi-volume dump on ECKD DASD” on page 69 for details.

Additional parameters might be required for logical boot devices (see “Preparing a logical device as a boot device” on page 64).
Menu configurations

For DASD and SCSI devices, you can define a menu configuration. A menu configuration has a section identifier that consists of a menu name with a leading colon.

The identifier is followed by one or more lines with references to IPL configurations in the same 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 64).

Example

Figure 26 on page 77 shows a sample configuration file that defines multiple configuration sections and two menu configurations.
Figure 26: Sample `/etc/zipl.conf` file

The following commands assume that the configuration file of the sample is the default configuration file.

- Call `zipl` to use the default configuration file settings:

  ```
  # zipl
  ```

  **Result:** `zipl` reads the default option from the `[defaultboot]` section and selects the `:menu1` section. It then installs a menu configuration with a boot configuration and a SCSI dump configuration.

- Call `zipl` to install a menu configuration (see also “Installing a menu configuration” on page 70):

  ```
  # zipl -m menu2
  ```
Result: zipl selects the :menu2 section. It then installs a menu configuration with two DASD boot configurations. “Example for a DASD menu configuration on z/VM” on page 83 and “Example for a DASD menu configuration (LPAR)” on page 88 illustrate what this menu looks like when it is displayed.

- Call zipl to install a boot loader for boot configuration [boot2]:

  # zipl boot2


- 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/36005076303ff4d0100000000000020c0-part1.
Chapter 7. Booting Linux

The options and requirements you have for booting Linux depend on your platform, LPAR or z/VM, and on your boot medium.

For booting Linux as a KVM guest, see Device Drivers, Features, and Commands for Linux as a KVM Guest, SC34-2754.

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, SC24-6287, the chapter about creating your first z/VM guest virtual machine for Linux and installing Linux.

IPL and booting

On IBM Z, you usually start booting Linux by performing an Initial Program Load (IPL).

Figure 27 on page 79 summarizes the main steps of the 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 92).

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 6, “Initial program loader for IBM Z - zipl,” on page 59.

**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 12 on page 80 provides an overview of the possibilities:

<table>
<thead>
<tr>
<th>Table 12: Boot media</th>
</tr>
</thead>
<tbody>
<tr>
<td>DASD</td>
</tr>
<tr>
<td>z/VM guest</td>
</tr>
<tr>
<td>LPAR</td>
</tr>
</tbody>
</table>

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. The z/VM reader is 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 83 and “Example for a DASD menu configuration (LPAR)” on page 88). 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 76 for information about defining menu configurations.

**Boot data**

To boot Linux, you generally need a kernel image, boot loader code, kernel parameters, and an initial RAM disk image.

For sequential I/O boot devices, z/VM reader and tape, the order in which this data is provided is significant. For random access devices, there is no required order.
Kernel image

You can use an existing kernel image or compile your own kernel.


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 4, “Kernel and module parameters,” on page 23, Chapter 6, “Initial program loader for IBM Z - zipl,” on page 59, 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.

For more general information about z/VM guest environments for Linux, see z/VM: Getting Started with Linux on System z, SC24-6287.
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:

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> clear parm <kernel_parameters>
   ```

   where

   `<devno>`
   is the device number of the boot device as seen by the guest virtual machine.

   `parm <kernel_parameters>`
   is an optional 64-byte string of kernel parameters to be concatenated to the end of the existing kernel parameters that are used by your boot configuration (see “Preparing a boot device” on page 62 for information about the boot configuration).

   See also “Specifying kernel parameters when booting Linux” on page 25.

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 62).

Procedure
Perform these steps to start the boot process:
1. Establish a CMS or CP session with the z/VM guest virtual machine where you want to boot Linux.
2. Ensure that the boot device is accessible to your z/VM guest virtual machine.
3. Issue a command of this form:

   ```
   #cp i <devno> clear loadparm <n> parm <kernel_parameters>
   ```

where:

- `<devno>`
  
specifies the device number of the boot device as seen by the guest.

- `loadparm <n>`
  
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” on page 83). These additional kernel parameters are appended to the parameters you might have provided in a parameter file. The combined parameter string must not exceed 895 bytes.

  See “Menu configurations” on page 76 for more details about menu configurations.

- `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 62 for information about the boot configuration).

  See also “Specifying kernel parameters when booting Linux” on page 25.

**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 26 on page 77 is displayed on the z/VM guest virtual machine console:

```
00: zIPL interactive boot menu
00: 0. default (boot1)
00: 1. boot1
00: 2. boot3
00: Note: VM users please use '#cp vi vmsg <input>'
00: Please choose (default will boot in 30 seconds): #cp vi vmsg 2
```

You choose a configuration by specifying the configuration number. For example, to boot configuration boot3 specify:

```
#cp vi vmsg 2
```

You can also specify additional kernel parameters by appending them to the configuration number. For example, you can specify:

```
#cp vi vmsg 2 maxcpus=1
```

These parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration when booting Linux.
Booting 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 62). 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 76.

   Example: To select a configuration with configuration number 2 from a menu configuration:
   
   ```
   #cp set loaddev bootprog 2
   ```

5. Optional: Specify kernel parameters.

   ```
   #cp set loaddev scpdata <APPEND|NEW> '<kernel_parameters>'
   ```

   where:
   
   - `<kernel_parameters>` specifies a set of kernel parameters to be stored as system control program data (SCPDATA). When booting Linux, these kernel parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration.

   `<kernel_parameters>` must contain ASCII characters only. If characters other than ASCII characters are present, the boot process ignores the SCPDATA.
<kernel_parameters> as entered from a CMS or CP session is interpreted as lowercase on Linux. If you require uppercase characters in the kernel parameters, run the SET LOADDEV command from a REXX script instead. In the REXX script, use the "address command" statement. See z/VM: REXX/VM Reference, SC24-6314 and z/VM: REXX/VM User's Guide, SC24-6315 for details.

**Optional: APPEND**

appends kernel parameters to existing SCPDATA. This is the default.

**Optional: NEW**

replaces existing SCPDATA.

**Examples:**

- To append kernel parameter noresume to the current SCPDATA:

  ```bash
  #cp set loaddev scpdata 'noresume'
  ```

- To replace the current SCPDATA with the kernel parameters resume=/dev/sda2 and no_console_suspend:

  ```bash
  #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:

```bash
#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, SC24-6268 for more information about the SET LOADDEV command.

**Booting from the z/VM reader**

Boot Linux by issuing the IPL command with the z/VM reader as the IPL device. You first must transfer the boot data to the reader.

**Before you begin**

You need the following files, all in record format fixed 80:

- Linux kernel image
- 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 62 for information about the boot configuration).

   See also “Specifying kernel parameters when booting Linux” on page 25.

---

**Booting Linux in LPAR mode**

You can boot Linux in LPAR mode from a Hardware Management Console (HMC) or Support Element (SE).

**About this task**

The following description refers to an HMC, but the same steps also apply to an SE.

**Booting from DASD**

Use the SE or HMC to boot Linux in LPAR mode from a DASD boot device.

**Before you begin**

You need a boot device that is prepared with zipl (see “Preparing a boot device” on page 62).
**Procedure**

Perform these steps to boot from a DASD 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, click **Load** (which is located in the **Recovery** area). See Figure 28 on page 87.

![Hardware Management Console](image-url)

*Figure 28: Load task on the HMC*

4. Select load type **Normal** as shown in Figure 29 on page 88.
5. Enter the device number of the DASD boot device in the **Load address** field.

   To IPL from a subchannel set other than 0, specify five digits: The subchannel set ID followed by the device number, for example 1E711.

6. If the boot configuration is part of a zipl created menu configuration, 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 88). These additional kernel parameters are appended to the parameters you might have provided in a parameter file. The combined parameter string must not exceed 895 bytes.

   See “Menu configurations” on page 76 for more details about menu configurations.

7. 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.

8. Click **OK** to start the boot process.

**Example for a DASD menu configuration (LPAR)**

This example illustrates how menu2 in the sample configuration file in Figure 26 on page 77 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
```

These parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration when booting Linux.

**What to do next**
Check the output on the preferred console (see “Console kernel parameter syntax” on page 41) to monitor the boot progress.

**Booting from SCSI**
Use the SE or HMC to boot Linux in LPAR from a SCSI boot device.

**Before you begin**
- You need a boot device that is prepared with zipl (see “Preparing a boot device” on page 62).
- You must have the SCSI IPL feature (FC9904) installed.

**Procedure**
Perform these steps to boot from a 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 and click **Load** (in the **Recovery** area) as shown in Figure 30 on page 90.
4. Select load type **SCSI** as shown in Figure 31 on page 90.

A SCSI device can be a disk or an FC-attached CD-ROM or DVD drive.

5. Enter the device number of the FCP channel through which the SCSI disk is accessed in the **Load address** field.

6. Enter the WWPN of the SCSI disk in the **World wide port name** field.
7. Enter the LUN of the SCSI disk in the **Logical unit number** field.
8. If the boot configuration is part of a zipl created menu configuration, type the configuration number that identifies your SCSI boot configuration within the menu in the **Boot program selector** field. Configuration number 0 specifies the default configuration.

   See “Menu configurations” on page 76 for more details about menu configurations.

9. Type kernel parameters in the **Operating system specific load parameters** field.

   These parameters are concatenated to the end of the existing kernel parameters that are used by your boot configuration when booting Linux.

   Use ASCII characters only. If you enter characters other than ASCII characters, the boot process ignores the data in the **Operating system specific load parameters** field.

10. Accept the defaults for the remaining fields.
11. Click **OK** to start the boot process.

**What to do next**

Check the output on the preferred console (see “Console kernel parameter syntax” on page 41) to monitor the boot progress.

**Booting from tape**

You can boot Linux in LPAR mode from tape.

**Before you begin**

You need a boot device that is prepared with zipl (see “Preparing a boot device” on page 62).

**Procedure**

Perform these steps to boot from a tape 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** as shown in Figure 32 on page 92.
4. Select the load type **Normal** (see Figure 29 on page 88).
5. Enter the device number of the tape boot device in the **Load address** field.
6. 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 41) to monitor the boot progress.

**Loading Linux from removable media or from an FTP server**

Instead of a boot loader, you can use SE functions to copy the Linux kernel image to your LPAR memory. After the Linux kernel is loaded, Linux is started using restart PSW.

**Before you begin**

You need installation data that includes a special file with installation information (with extension "ins"). This file can be in different locations:

- On a disk that is inserted in the CD-ROM or DVD drive of the system where the HMC runs
- In the file system of an FTP server that you can access through FTP from your HMC system

The .ins file contains a mapping of the location of installation data on the disk or FTP server and the memory locations where the data is to be copied.

**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, click **Load from Removable Media or Server**, which is located in the **Recovery** area (see Figure 33 on page 93).

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 34 on page 94).
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 an initial installation from removable media at the HMC
  a. Select Hardware Management Console CD / DVD and assign for operating system use (see Figure 34 on page 94).
  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 removable media.

The installation CD or DVD must hold a distribution that supports an installation from the HMC.

- 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 35 on page 94).

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 670. 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.
- **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.

```
# 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 25.

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 26.

If the device is FCP, a number of additional files are present (also see Chapter 13, “SCSI-over-Fibre Channel device driver,” on page 177 for details):

- **device**: Contains the bus ID of the FCP device that is used for IPL, for example:
loadparm
Contains up to 8 characters for the loadparm.

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 84 and “Booting from DASD” on page 86).

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 26.

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 IBM Z hardware must have SCSI 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 581.

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 or 0.1.5119.

loadparm
Up to eight characters for the loadparm used to select the boot configuration in the zipl menu (if
available).

parm
A 64-byte string of kernel parameters that is concatenated to the boot command-line. The PARM
parameter can be set only for Linux on z/VM. See also “Specifying kernel parameters when booting
Linux” on page 25.

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 26.

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.

loadparm
Up to eight characters for the loadparm.

scp_data
Kernel parameters to be used for the next FCP re-IPL. See also “Specifying kernel parameters when
booting Linux” on page 25.

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 26.

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.
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 25.

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 26.

Note: You cannot load Linux kernel 4.13 or later from an NSS. The NSS must contain a Linux distribution with shared kernel support or another mainframe operating system, for example, CMS.

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
```

• 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 8. Suspending and resuming Linux

With suspend and resume support, you can stop a running instance of Linux on Z 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.

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 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.
• 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 475 and “Setting the access mode” on page 473.
• 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.

- 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.

```
resume=<device_node>
no_console_suspend
noresume
```

*Figure 36: suspend and resume kernel parameter syntax*

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 \texttt{pri=} parameter to specify the swap partitions in \texttt{/etc/fstab} with different priorities. See the \texttt{swapon} man page for details.

The following example shows two swap partitions with different priorities:

\begin{verbatim}
# cat /etc/fstab
.../dev/dasdb1 swap swap pri=-1 0 0
/dev/dasdc1 swap swap pri=-2 0 0
\end{verbatim}

In the example, the partition to be used for the resume data is \texttt{/dev/dasdc1}.

You can check your current swap configuration by reading \texttt{/proc/swaps}.

\begin{verbatim}
# cat /proc/swaps
Filename    Type         Size   Used   Priority
/dev/dasdb1  partition    7212136 71056   -1
/dev/dasdc1  partition    7212136   0      -2
\end{verbatim}

**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 “\texttt{cioignore} - List devices to be ignored” on page 758 and “\texttt{cioignore} - Manage the I/O exclusion list” on page 596).

**Suspending a Linux instance**

Suspend a Linux instance by writing to the \texttt{/sys/power/state} sysfs attribute.

\textbf{Before you begin}

\begin{itemize}
  \item \textbf{Attention:} Suspend only Linux instances for which you specified the \texttt{resume=} kernel parameter. Without this parameter, you cannot resume the suspended Linux instance.
\end{itemize}

\textbf{Procedure}

Enter the following command to suspend a Linux instance:

\begin{verbatim}
# echo disk > /sys/power/state
\end{verbatim}

\textbf{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 53 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.

\textbf{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 iucvty 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 systemd, Upstart, or inittab.

#### Using inittab

For distributions that use inittab, edit `/etc/inittab`.

Replace the line

```plaintext
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

with

```plaintext
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

```plaintext
exec /sbin/shutdown -t3 -r now "Control-Alt-Delete pressed"
```

with

```plaintext
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.

#### Using systemd

For distributions that use systemd, create a symbolic link from the systemd target that handles SIGNAL SHUTDOWN to the systemd target that handles suspend.

Issue the following command to create the symbolic link:
# ln -sf /usr/lib/systemd/system/suspend.target /etc/systemd/system/ctrl-alt-del.target

This command replaces /etc/systemd/system/ctrl-alt-del.target if it exists.
Chapter 9. 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>
<thead>
<tr>
<th>Trigger</th>
<th>Command or condition</th>
<th>Default shutdown action</th>
</tr>
</thead>
<tbody>
<tr>
<td>halt</td>
<td>Linux shutdown -H command</td>
<td>stop</td>
</tr>
<tr>
<td>poff</td>
<td>Linux poweroff or shutdown -P command</td>
<td>stop</td>
</tr>
<tr>
<td>reboot</td>
<td>Linux reboot or shutdown -r command</td>
<td>reipl</td>
</tr>
</tbody>
</table>
| 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 14 on page 105.

<table>
<thead>
<tr>
<th>Action</th>
<th>Explanation</th>
<th>See also</th>
</tr>
</thead>
</table>
| 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 95 |
| reipl  | Performs an IPL according to the specifications in /sys/firmware/reipl. | “Rebooting from an alternative source” on page 96 |
| 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 106 |

Use lsshut to find out which shutdown action is configured for each shutdown trigger, see “lsshut - List the current system shutdown actions” on page 673.

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 585.
- For restart and panic use dumpconf, see Using the Dump Tools, SC33-8412.
kdump for restart and panic

If kdump is set up for a Linux instance, kdump is started to create a dump, regardless of the shutdown actions that are specified for restart and panic. With kdump, these settings act as a backup that is used only if kdump fails.

Note: kdump is not a shutdown action that you can set as a sysfs attribute value for a shutdown trigger. See Using the Dump Tools, SC33-8412 about how to set up kdump.

Override for halt

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.

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 37: 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.

In sysfs, two attributes are required to set a z/VM CP command as a shutdown action for a trigger <trigger>:

- /sys/firmware/shutdown_actions/on_<trigger> must be set to vmcmd.
- /sys/firmware/vmcmd/on_<trigger> specifies the z/VM CP command.
The values of the attributes in the /sys/firmware/vmcmd directory must conform to these rules:

- The value must be a valid z/VM CP command.
- The commands, including any z/VM user IDs or device numbers, must be specified with uppercase characters.
- Commands that include blanks must be delimited by double quotation marks (".
- The value must not exceed 127 characters.

You can specify multiple z/VM CP commands that are separated by the newline character "\n". Each newline is counted as one character. When writing values with multiple commands, use this syntax to ensure that the newline character is processed correctly:

```
# echo -e <cmd1>\n<cmd2>... | cat > /sys/firmware/vmcmd/on_<trigger>
```

where `<cmd1>`\n`<cmd2>`... are two or more z/VM CP commands and `on_<trigger>` is one of the attributes in the vmcmd directory.

The -e echo option and redirect through cat are required because of the newline character.

**Example for a single z/VM CP command**

Issue the following command to configure the z/VM CP LOGOFF command as the shutdown action for the poff shutdown trigger.

```
# chshut poff vmcmd "LOGOFF"
```

Alternatively, you can issue the following commands to directly write the shutdown configuration to sysfs:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_poff
# echo LOGOFF > /sys/firmware/vmcmd/on_poff
```

Figure 38 on page 107 illustrates the relationship of the sysfs attributes for this example.

**Example for multiple z/VM CP commands**

Issue the following command to configure two z/VM CP commands as the shutdown action for the poff shutdown trigger. First a message is sent to user OPERATOR, and then the LOGOFF command is issued.

```
# chshut poff vmcmd "MSG OPERATOR Going down" vmcmd "LOGOFF"
```
Alternatively, you can issue the following commands to directly write the shutdown configuration to sysfs:

```
# echo vmcmd > /sys/firmware/shutdown_actions/on_poff
# echo -e "MSG OPERATOR Going down\nLOGOFF" | cat > /sys/firmware/vmcmd/on_poff
```
Chapter 10. The diag288 watchdog device driver

The diag288 watchdog device driver provides Linux watchdog applications with access to the watchdog timer on IBM Z.

You can use the diag288 watchdog in these environments:

- Linux on z/VM
- Linux in LPAR mode as of z13s and z13 with the enhancements of February 2016.
- Linux as a KVM guest (see Device Drivers, Features, and Commands for Linux as a KVM Guest, SC34-2754)

The diag288 watchdog device driver provides the following features:

- Access to the watchdog timer on Z.
- An API for watchdog applications (see “External programming interfaces ” on page 113).

Watchdog applications can be used to set up automated restart mechanisms. Watchdog-based restart mechanisms are an alternative to a networked heartbeat with STONITH (see “STONITH support (snipl for STONITH)” on page 132).

Watchdog applications that communicate directly with the 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 IBM 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 39 on page 109).

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 113.

**Linux on z/VM only:** Loading or saving a DCSS can take a long time during which the virtual machine does not respond, depending on the size of the DCSS. As a result, a watchdog might time out and restart the guest. You are advised not to use the watchdog in combination with loading or saving DCSSs.

See also the generic watchdog documentation in 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 diagram](image)

**Figure 40: 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 9, “Shutdown actions,” on page 105).

For Linux on z/VM, you can use the `diag288_wdt.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 parameters diagram](image)
Notes:

1 diag288_wdt.cmd= and diag288_wdt.conceal= apply only to Linux on z/VM and are ignored for Linux in LPAR mode.

Figure 41: watchdog kernel parameter syntax

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 9, “Shutdown actions,” on page 105).

Any other specification dissociates the timeout action from the restart shutdown trigger. Instead, the specification is issued by CP and must adhere to these rules:

• It must be a single valid CP command
• It must not exceed 230 characters
• It must be enclosed by quotation marks if it contains any blanks or newline characters

The specification is converted from ASCII to uppercase EBCDIC.

For details about CP commands, see z/VM: CP Commands and Utilities Reference, SC24-6268.

On an running instance of Linux on z/VM, you can write to /sys/module/diag288_wdt/parameters/cmd to replace the command you specify when loading the module. Through this sysfs interface, you can also specify multiple commands to be issued, see “Example for Linux on z/VM” on page 111 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_wdt.conceal=1

enables the protected application environment where the guest is protected from unexpectedly entering CP READ. Do not enable the protected environment for guests with multiprocessor configurations. The protected application facility supports only virtual uniprocessor systems.

For details, see the "SET CONCEAL" section of z/VM: CP Commands and Utilities Reference, SC24-6268.

<nowayout_flag>

determines what happens when the watchdog device node is closed by the watchdog application.

If the flag is set to 1, 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.

```bash
diag288_wdt.cmd="ipl b1a0" diag288_wdt.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.

```bash
/bin/echo -en "MSG * WATCHDOG FIRED\nVMDUMP\nIPL" > /sys/module/diag288_wdt/parameters/cmd
```

Use the echo version at `/bin/echo`. The built-in echo command from bash might not process the newline characters as intended.

To verify that your commands have been accepted, issue:

```bash
# cat /sys/module/diag288_wdt/parameters/cmd
MSG * WATCHDOG FIRED
VMDUMP
IPL
```

**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.

```bash
modprobe diag288_wdt cmd="SYSTEM RESTART"
```

**Notes:**

1. `cmd=` and `conceal=` apply only to Linux on z/VM and are ignored for Linux in LPAR mode.

*Figure 42: watchdog module parameter syntax*

The variables have the same meaning as the corresponding kernel parameters with prefix `diag288_wdt` in “Watchdog kernel parameters” on page 110.

### 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.

```bash
# 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:

   ```bash
   # 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.

   ```bash
   # 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 and the supported IOCTLs, see the Documentation/watchdog/watchdog-api.txt file in the Linux source tree.

The default watchdog timeout is 30 seconds, the minimum timeout that can be set through the IOCTL `WDIOC_SETTIMEOUT` is 15 seconds.
Chapter 11. Remotely controlling virtual hardware - snipl

snipl is a command-line tool for remotely controlling virtual IBM Z hardware.
You can use snipl to activate and deactivate virtual IBM 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.

This information applies to simple network IPL (snipl) version 3.1.0. You can obtain the snipl package at www.ibm.com/developerworks/linux/linux390/snipl.html.

With snipl version 3.0.0, the default behavior changed. snipl now uses SNMPv3 to encrypt communication with LPAR hypervisors (SE or HMC), and OpenSSL for communication with z/VM hypervisors.

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 with disabled encryption, 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 IBM 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 "hwmcapi" 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.
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” on page 116 summarizes snip1 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 snip1 command in LPAR mode always requires a main option, access data, and, with one exception, specifications for one or more LPARs.

Where:

- `<image_name>` specifies an LPAR. If snip1 directly accesses the SE, this is the LPAR name as defined in the hardware setup.

If snip1 accesses the SE through an HMC, the specification has the format `<mainframe_system>-<lpar_name>` where `<mainframe_system>` is the name that identifies the mainframe on the HMC. If you are using a snip1 configuration file that defines an alias for an LPAR, you can specify the alias.

**SE Example:** `lpar204`

**HMC Example:** `z02-lpar204`

A snip1 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” on page 117.

- `-a, -d, -r, -o, -g` are described in “Activate, deactivate, reset, stop, or get status information” on page 118.
-l is described in “Perform an IPL operation from a CCW device” on page 119.

-s, -D are described in “Perform an IPL or dump operation from a SCSI device” on page 120.

-x is described in “List LPARs” on page 122.

-i is described in “Emulate the Operating Systems Messages applet” on page 123.

-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.

- 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 use IPv6 or IPv4 connections.

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.

-P or --promptpassword
prompts for a password in protected entry mode.

-u <user> or --userid <user>
specifies an SNMPv3 user identifier that is authorized to access an HMC or SE. This parameter can be omitted if it is specified in the configuration file.

-e or --noencryption
specifies that no encryption is used when connecting to the server. A user name is not allowed if encryption is disabled. This parameter can also be specified through a configuration file.

-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 126.

```
--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.

```
snipl <image_name> |par-access-data
```

Notes:
1. If not specified, the HMC or SE default profile for the specified LPAR is used.

*Figure 44: LPAR mode: -a, -d, -r, -o, -g options*

Where:

```
<image_name>
```
see “Overview for LPAR mode” on page 116.

```
|par-access-data|
```
see “Specifying access data for LPAR mode” on page 117.

- **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 -e -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 SZ02LP03 on a mainframe system that is identified as SZ02 on an HMC with an IP address 2001:0db8::11a0:

```
# snipl SZ02-SZ02LP03 -L 2001:0db8::11a0 -e -P -g
Enter password:
Warning : No default configuration file could be found/opened.
status of SZ02
SZ02LP033: 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 120.

![Diagram](image)

**Figure 45: 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 116.

- `lpar-access-data`
  see “Specifying access data for LPAR mode” on page 117.

- `-l` or `--load`
  performs an IPL for the specified LPARs.

- `-F` or `--force`
  unconditionally forces the IPL operation.

- `-A <load_address>` or `--address_load <load_address>`
  specifies the hexadecimal four-digit device number of the IPL device. To use a device from a subchannel set other than 0, specify five digits: The subchannel set ID followed by the device number, for example 15199. The default is subchannel set 0. If the -A 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.

Examples:

• The following command performs an IPL from a CCW device with bus ID 0.0.5119 for an LPAR SZ02LP03 on a mainframe system that is identified as SZ02 on an HMC with an IP address 2001:0db8::11a0:

```
# snipl SZ02-SZ02LP03 -L 2001:0db8::11a0 -e -P -l -A 5119
Enter password:
Warning: No default configuration file could be found/opened.
processing......
SZ02
SZ02LP03: acknowledged.
```

• To perform an IPL from a CCW device in subchannel set 1 with the bus ID 0.1.5119 for an LPAR SZ03LP00:

```
% snipl SZ03LP00 -L 192.0.2.4 -P -e -l -A 15119
```

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 119.
Remote controlling virtual hardware - snipl

Figure 46: LPAR mode: SCSI IPL or dump

Where:

<iimage_name></iimage_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 116.

<|lpar-access-data|>
see “Specifying access data for LPAR mode” on page 117.

-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 <load_address> 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.

Note: The IPL device must be on subchannel set 0.

--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 25).

--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 -e -P -s -A 3d0f --wwpn_scsiload 500507630303c562 \ 
--lun_scsiload 4010404900000000
```

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 mainframe.

```
Figure 47: LPAR mode: list
```

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 116).

* `|lpar-access-data|`
  see “Specifying access data for LPAR mode” on page 117.

* `-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 -e -P -x
```

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.

```
snipl <image_name> lpar-access-data -i --msgtimeout 5000 --msgtimeout <interval> --msgfilename <name>
```

*Figure 48: 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 116).

- `lpar-access-data` see “Specifying access data for LPAR mode” on page 117.

- `-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 -e -P -i -M SZ01LP02.transcript
```

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*, SC24-6327.

**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.
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.

Figure 49: snipl command syntax (z/VM mode)

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 VSMERVE service machine through which the specified z/VM guest virtual machines are controlled. You can use IPv6 or IPv4 connections.

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 VSMERVE service machine. This option can be omitted if defined in the configuration file.

-e or --noencryption
specifies that no encryption is used when connecting to the server. This parameter can also be specified through a configuration file. To use encryption, you require a configuration file with an SSL fingerprint defined.

-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 VSMERVE 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 126.
--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” on page 126). 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.

Examples

The following command logs on two z/VM guest virtual machines:

```bash
# snipl sndlnx04 sndlnx05 -V sandbox.www.example.com -e -z 44444 -u sndadm01 -p pw42play -a
Warning : No default configuration file could be found/opened.
* ImageActivate : Image sndlnx04 Request Successful
* ImageActivate : Image sndlnx05 Request Successful
```

The following command logs on a z/VM guest virtual machine:

```bash
# snipl vm04lnxd -V 2001:0db8::1a:0015 -e -z 37899 -u vm04main -p mainpw -a
Warning : No default configuration file could be found/opened.
* ImageActivate : Image vm04lnxd 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 117 or “Command-line syntax (z/VM mode)” on page 124 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 15 on page 127 summarizes the keywords for the configuration file and the command-line equivalents for LPAR mode and z/VM mode.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value for LPAR mode</th>
<th>Value for z/VM mode</th>
<th>Command-line equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>Starts a configuration file section by specifying the IP address or host name of an HMC or SE.</td>
<td>Starts a configuration file section by specifying the IP address or host name of a SMAPI request server or VSMServe service machine.</td>
<td>(See note “1” on page 128)</td>
</tr>
<tr>
<td></td>
<td>You can use IPv6 or IPv4 connections.</td>
<td>You can use IPv6 or IPv4 connections.</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>LPAR</td>
<td>VM</td>
<td>(See note “1” on page 128)</td>
</tr>
<tr>
<td></td>
<td>(required)</td>
<td>(required)</td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>An SNMPv3 user identifier that is authorized for the HMC or SE.</td>
<td>A z/VM user ID that is authorized for the SMAPI request server or VSMServe service machine.</td>
<td>-u or --user</td>
</tr>
<tr>
<td></td>
<td>(See note “2” on page 128)</td>
<td>(See note “2” on page 128)</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>The value for community in the SNMPv2 (unencrypted) settings of the HMC or SE.</td>
<td>The password for the z/VM user ID specified with the user keyword.</td>
<td>-p or --password</td>
</tr>
<tr>
<td></td>
<td>(See note “3” on page 128)</td>
<td>(See note “2” on page 128)</td>
<td></td>
</tr>
<tr>
<td>encryption</td>
<td>&quot;no&quot; specifies an SNMPv2 unencrypted connection.</td>
<td>&quot;no&quot; specifies unencrypted connection to the SMAPI request server.</td>
<td>-e or --noencryption</td>
</tr>
<tr>
<td></td>
<td>&quot;yes&quot; specifies an SNMPv3 encrypted connection.</td>
<td>&quot;yes&quot; specifies use of the OpenSSL protocol when connecting to the SMAPI request server.</td>
<td></td>
</tr>
<tr>
<td>sslfingerprint</td>
<td>n/a</td>
<td>If encryption is enabled, the fingerprint mechanism is used to detect man-in-the-middle attacks. Specified in the configuration file fingerprint value must be equal to the server certificate fingerprint for each new sn IPL connection. The sslfingerprint connection parameter can be specified only in a configuration file.</td>
<td></td>
</tr>
<tr>
<td>port</td>
<td>n/a</td>
<td>Required if the server keyword specifies the IP address or host name of a SMAPI request server.</td>
<td>-z or --port</td>
</tr>
<tr>
<td>Keyword</td>
<td>Value for LPAR mode</td>
<td>Value for z/VM mode</td>
<td>Command-line equivalent</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>image</td>
<td>An LPAR name as defined in the mainframe hardware configuration.</td>
<td>A z/VM user ID that specifies a target z/VM guest virtual machine.</td>
<td>A list of one or more items that are separated by blanks and specified without a switch.</td>
</tr>
<tr>
<td></td>
<td>If the server keyword specifies an HMC, the specification begins with the name that identifies the mainframe on the HMC, followed by a hyphen (-), followed by the LPAR name.</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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. The user keyword 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 50 on page 129 shows a configuration file example with multiple sections, including sections for LPAR mode and for z/VM mode.
Examples

- 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.

Assuming that the configuration file of Figure 50 on page 129 is available at /etc/xcf

```
# snipl sndlnx01 tutor -a -f /etc/xcf
Server sandbox.www.example.com from config file /etc/xcf is used
* ImageActivate : Image sndlnx01 Request Successful
* ImageActivate : Image sndlnx03 Image Already Active
```

Assuming that the configuration file of Figure 50 on page 129 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
* 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 -u sz01adm -l -P -A 5000
Enter password:
Warning : No default configuration file could be found/opened.
SZ01LP03: acknowledged.
```

Assuming that the configuration file of Figure 50 on page 129 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
SZ01LP03: acknowledged.
```

Assuming that the configuration file of Figure 50 on page 129 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
SZ01LP03: acknowledged.
```

• Assuming that the configuration file of Figure 50 on page 129 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
```

• The following command logs on a z/VM guest virtual machine:

```
# snipl vm04lnxd -V 2001:0db8::1a:0015 -e -z 37899 -u vm04main -p mainpw -a
Warning : No default configuration file could be found/opened.
* ImageActivate : Image vm04lnxd Request Successful
```

Assuming that the configuration file of Figure 50 on page 129 is used by default, an equivalent command would be:

```
# snipl vm04lnxd -a
Enter password:
Server 2001:0db8::1a:0015 from config file /etc/snipl.conf is used
* ImageActivate : Image vm04lnxd Request Successful
```

---

**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:
An unknown command option was specified.

A command option with an invalid value was specified.

A command option was specified more than once.

Conflicting command options was specified.

No command option was specified.

No SE, HMC, SMAPI request server or VSMERVE service machine was specified on the command line or through a configuration file.

No LPAR or z/VM guest virtual machine was specified.

No z/VM user ID was specified on the command line or through a configuration file.

No password was specified on the command line or through a configuration file.

A specified LPAR or z/VM guest virtual machine does not exist on the specified SE or z/VM system.

More than one LPAR was specified for option --dialog.

No user name was specified in the command line or configuration file. User name is required when encryption is selected.

User name must not be specified when encryption is disabled.

The following return codes indicate setup errors or program errors:

An error occurred while loading one of the systems management API libraries libhwmcaapi.so or libvmsmapi.so.

Operation --dialog encounters a problem while starting another process.

Operation --dialog encounters a problem with stdin attribute setting.

A response from the HMC or SE could not be interpreted.

The response buffer is too small for a response from the HMC or SE.

A storage allocation failure occurred.

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 126.
- 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 115 and “Setting up snipl for z/VM mode” on page 123.

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 126. 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.

On z/VM, you must use a configuration file containing a SSL fingerprint for an encrypted connection.

Figure 51: stonith syntax (simplified)

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 126).

For LPAR mode the following keywords are required:

- server
- type
- user (or encryption=no)
- 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 VVSMSERVE service machine)
- type
- user
- password
- image

snipl_file <parameters>
specifies a configuration file (see “The snipl configuration file” on page 126). 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 vm04lnxd:

  stonith -t lic_vps -p "snipl_param server=2001:0db8::1a:0015,type=vm,user=vm04main,password=mypasswd,encryption=no,image=vm04lnxd" -T reset vm04lnxd

  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 vm04lnxd
Part 3. Storage

Linux on Z includes several storage device drivers that are specific to z/Architecture.

**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
Chapter 12. 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 13, “SCSI-over-Fibre Channel device driver,” on page 177).

Features

The DASD device driver supports a wide range of disk devices and disk functions.

- The DASD device driver has no dependencies on the adapter hardware that is used to physically connect the DASDs to the IBM Z hardware. You can use any adapter that is supported by the IBM Z hardware (see www.ibm.com/systems/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 16 on page 137:

<table>
<thead>
<tr>
<th>Device format</th>
<th>Control unit type</th>
<th>Device type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECKD</td>
<td>1750</td>
<td>3380 and 3390</td>
</tr>
<tr>
<td>ECKD</td>
<td>2107</td>
<td>3380 and 3390</td>
</tr>
<tr>
<td>ECKD</td>
<td>2105</td>
<td>3380 and 3390</td>
</tr>
<tr>
<td>ECKD</td>
<td>3990</td>
<td>3380 and 3390</td>
</tr>
<tr>
<td>ECKD</td>
<td>9343</td>
<td>9345</td>
</tr>
<tr>
<td>ECKD</td>
<td>3880</td>
<td>3390</td>
</tr>
<tr>
<td>FBA</td>
<td>6310</td>
<td>9336</td>
</tr>
<tr>
<td>FBA</td>
<td>3880</td>
<td>3370</td>
</tr>
</tbody>
</table>

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 “IBM Z compatible disk layout” on page 139 for details.
- The DASD device driver provides an option for extended error reporting for ECKD devices. Extended error reporting can support high availability setups.
- The DASD device driver supports parallel access volume (PAV) and HyperPAV on storage devices that provide this feature. 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. Use
the dasdstat command to check whether a DASD uses PAV, see “Scenario: Verifying that PAV and HPF are used” on page 164.

• The DASD device driver supports High Performance FICON, including multitrack requests, on storage devices that provide this feature. Use the dasdstat command to check whether a DASD uses High Performance FICON, see “Scenario: Verifying that PAV and HPF are used” on page 164.

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

Linux on Z includes the IBM label partitioning scheme that preserves the Z IPL, label, and VTOC records. With this partitioning scheme, Linux can share a disk with other mainframe operating systems. For example, a traditional mainframe operating system can handle backup and restore for a partition that is used by Linux.

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

• “IBM Z compatible disk layout” on page 139
• “Linux disk layout” on page 141
• “CMS disk layout” on page 141

DASD partitions

Partitioning 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 extent 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.

IBM Z compatible disk layout

With the compatible disk layout, a DASD can have up to three partitions that can be accessed by traditional mainframe operating systems.

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

Figure 52 on page 139 illustrates a DASD with the compatible disk layout.

```
Figure 52: Compatible disk layout

The IPL records, volume label (VOL1), and VTOC of disks with the compatible disk layout are on the first two tracks of the disks. These tracks are not intended for use by Linux applications. Using the tracks can result in data loss.

Linux can address the device as a whole as `/dev/dasd<x>`, where `<x>` can be one to four letters that identify the individual DASD (see “DASD naming scheme” on page 142).

Disks with the compatible disk layout can have one to three partitions. Linux addresses the first partition as `/dev/dasd<x>1`, the second as `/dev/dasd<x>2`, and the third as `/dev/dasd<x>3`.

You use the `dasdfmt` command (see “dasdfmt - Format a DASD” on page 615) to format a disk with the compatible disk layout. You use the `fdasd` command (see “fdasd - Partition a DASD” on page 633) 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.

DASD device driver 139
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 Lnnnnn (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 Z, like other mainframe operating systems, uses a Volume Table Of Contents (VTOC).

The VTOC contains pointers to the location of every data set on the volume. These data sets form the Linux partitions.

The VTOC is on the second track (cylinder 0, track 1). It contains a number of records, each written in a separate data set control block (DSCB). The number of records depends on the size of the volume:

- One DSCB that describes the VTOC itself (format 4)
- One DSCB that is required by other operating systems but is not used by Linux. `fdasd` sets it to 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 IBM Z tools such as VM/DITTO. A Linux DASD with physical device number 0x0193, volume label "LNX001", and three partitions might be displayed like this example:
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 155 for how to enable DIAG.

Figure 53 on page 141 illustrates a disk with the Linux disk layout.

![Linux disk layout](image)

**Figure 53: Linux disk layout**

DASDs with the Linux disk layout either have an LNX1 label or are not labeled. The first records of the device are reserved for IPL records and the volume label, and are not intended for use by Linux applications. All remaining records are grouped into a single partition. You cannot have more than a single partition on a DASD that is formatted in the Linux disk layout.

Linux can address the device as a whole as `/dev/dasd<x>`, where `<x>` can be one to four letters that identify the individual DASD (see “DASD naming scheme” on page 142). Linux can access the partition as `/dev/dasd<x>1`.

You use the `dasdfmt` command (see “dasdfmt - Format a DASD” on page 615) to format a disk with the Linux disk layout.

**CMS disk layout**

The CMS disk layout applies only to Linux on z/VM. The disks are formatted with z/VM tools.

Both ECKD or FBA type DASD can have the CMS disk layout. DASD partitions that are formatted with this layout cannot be accessed by traditional mainframe operating systems. Apart from accessing the disks as ECKD or FBA devices, you can also access them using the DASD DIAG access method.

Figure 54 on page 142 illustrates two variants of the CMS disk layout.
The first variant contains IPL records, a volume label (CMS1), and a CMS data area. Linux treats DASD like this equivalent to a DASD with the Linux disk layout, where the CMS data area serves as the Linux partition.

The second variant is a CMS reserved volume. In this variant, the DASD was reserved by a CMS RESERVE fn ft fm command. In addition to the IPL records and the volume label, DASD with the CMS disk layout also have CMS metadata. The CMS reserved file serves as the Linux partition.

For both variants of the CMS disk layout, you can have only a single Linux partition. The IPL record, volume label and (where applicable) the CMS metadata, are not intended for use by Linux applications.

Addressing the device and partition is the same for both variants. Linux can address the device as a whole as /dev/dasd<x>, where <x> can be one to four letters that identify the individual DASD (see “DASD naming scheme” on page 142). Linux can access the partition as /dev/dasd<x>1.

“Enabling the DASD device driver to use the DIAG access method” on page 155 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>
<thead>
<tr>
<th>Disk layout</th>
<th>ECKD device format</th>
<th>FBA device format</th>
<th>DIAG access method support (z/VM only)</th>
<th>Maximum number of partitions</th>
<th>Formatting tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compatible disk layout</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>3</td>
<td>dasdfmt</td>
</tr>
<tr>
<td>Linux disk layout</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>1</td>
<td>dasdfmt</td>
</tr>
<tr>
<td>CMS (z/VM only)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>1</td>
<td>z/VM tools</td>
</tr>
</tbody>
</table>

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 18 on page 143 shows how the device names map to the available minor numbers.

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

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 142).

Example

The following nodes use the form /dev/<device_name> for the device nodes. The assignment of minor numbers is according to Table 18 on page 143.
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 139).

If you assign the same VOLSER to multiple devices, Linux can still access each device through its standard device node. However, only one of the devices can be accessed through the VOLSER-based device node. Thus, the node is ambiguous and might lead to unintentional data access.

Furthermore, if the VOLSER on the device that is addressed by the node is changed, the previously hidden device is not automatically addressed instead. To reassign the node, you must reboot Linux or force the kernel to reread the partition tables from disks, for example, by issuing:

```
# blockdev --rereadpt /dev/dasdz
```

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-uid/<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 144).
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 55 on page 146 gives an overview of the available DASD kernel configuration options and the corresponding modules.

---

**Figure 55: 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.

```
<device_bus_id>
<from_device_bus_id> - <to_device_bus_id>
```

*Figure 56: 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 142 for the naming scheme.

  The device names are assigned in order of ascending subchannel numbers. Auto-detection can yield confusing results if you change your I/O configuration and reboot, or if your Linux instance runs as a z/VM guest because the devices might appear with different names and minor numbers after rebooting.

- **probeonly**
  causes the DASD device driver to reject any "open" syscall with EPERM.

- **autodetect,probeonly**
  causes the DASD device driver to assign device names and minor numbers as for auto-detect. All devices regardless of whether they are accessible as DASD return EPERM to any "open" requests.

- **nopav**
  suppresses parallel access volume (PAV and HyperPAV) enablement for Linux instances that run in LPAR mode. The nopav keyword has no effect for Linux on z/VM.

- **nofcx**
  suppresses accessing the storage server with the I/O subsystem in transport mode (also known as High Performance FICON).

- **<device_bus_id>**
  specifies a single DASD.
<from_device_bus_id>-<to_device_bus_id>
specifies the first and last DASD in a range. All DASD devices with bus IDs in the range are selected. The device bus-IDs <from_device_bus_id> and <to_device_bus_id> need not correspond to actual DASD.

(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 159).

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 19 on page 148 shows the resulting allocation of device names and minor numbers:

<table>
<thead>
<tr>
<th>Minor</th>
<th>Name</th>
<th>To access</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>dasda</td>
<td>device 0.0.7000 as a whole</td>
</tr>
<tr>
<td>1</td>
<td>dasda1</td>
<td>the first partition on 0.0.7000</td>
</tr>
<tr>
<td>2</td>
<td>dasda2</td>
<td>the second partition on 0.0.7000</td>
</tr>
<tr>
<td>3</td>
<td>dasda3</td>
<td>the third partition on 0.0.7000</td>
</tr>
<tr>
<td>4</td>
<td>dasdb</td>
<td>device 0.0.7001 as a whole</td>
</tr>
<tr>
<td>5</td>
<td>dasdb1</td>
<td>the first partition on 0.0.7001</td>
</tr>
<tr>
<td>6</td>
<td>dasdb2</td>
<td>the second partition on 0.0.7001</td>
</tr>
<tr>
<td>7</td>
<td>dasdb3</td>
<td>the third partition on 0.0.7001</td>
</tr>
<tr>
<td>8</td>
<td>dasdc</td>
<td>device 0.0.7002 as a whole</td>
</tr>
<tr>
<td>9</td>
<td>dasdc1</td>
<td>the first partition on 0.0.7002</td>
</tr>
<tr>
<td>10</td>
<td>dasdc2</td>
<td>the second partition on 0.0.7002</td>
</tr>
<tr>
<td>11</td>
<td>dasdc3</td>
<td>the third partition on 0.0.7002</td>
</tr>
</tbody>
</table>
Table 19: Example mapping of device names and minor numbers to devices (continued)

<table>
<thead>
<tr>
<th>Minor</th>
<th>Name</th>
<th>To access</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>dasdd</td>
<td>device 0.0.7005 as a whole</td>
</tr>
<tr>
<td>13</td>
<td>dasdd1</td>
<td>the first partition on 0.0.7005 (read-only)</td>
</tr>
<tr>
<td>14</td>
<td>dasdd2</td>
<td>the second partition on 0.0.7005 (read-only)</td>
</tr>
<tr>
<td>15</td>
<td>dasdd3</td>
<td>the third partition on 0.0.7005 (read-only)</td>
</tr>
<tr>
<td>16</td>
<td>dasde</td>
<td>device 0.0.7006 as a whole</td>
</tr>
<tr>
<td>17</td>
<td>dasde1</td>
<td>the first partition on 0.0.7006</td>
</tr>
<tr>
<td>18</td>
<td>dasde2</td>
<td>the second partition on 0.0.7006</td>
</tr>
<tr>
<td>19</td>
<td>dasde3</td>
<td>the third partition on 0.0.7006</td>
</tr>
</tbody>
</table>

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.

Figure 57: DASD module parameter syntax

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 147).

If the extended error reporting feature is compiled into this module (see “Building a kernel with the DASD device driver” on page 146), 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:

```bash
# 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:

```bash
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 19 on page 148.

The following example specifies that High Performance FICON are to be suppressed for all DASDs:

```bash
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 146)
- Your distribution might create the required device node for you, for example, with udev.

To check if there is already a node issue:

```bash
# 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 10 to avoid errors when you are working with devices that have become available to a running Linux instance.

- “Preparing an ECKD type DASD for use” on page 152
- “Preparing an FBA-type DASD for use” on page 153
- “Accessing DASD by force” on page 154
- “Enabling the DASD device driver to use the DIAG access method” on page 155
- “Using extended error reporting for ECKD type DASD” on page 156
- “Setting a DASD online or offline” on page 157
- “Enabling and disabling logging” on page 158
- “Enabling and disabling immediate failure of I/O requests” on page 159
- “Setting the timeout for I/O requests” on page 160
- “Working with DASD statistics in debugfs” on page 161
- “Accessing full ECKD tracks” on page 165
- “Handling lost device reservations” on page 167
- “Reading and resetting the reservation state” on page 168
- “Setting defective channel paths offline automatically” on page 169
- “Querying the HPF setting of a channel path” on page 170
- “Checking for access by other operating system instances” on page 171
- “Displaying DASD information” on page 172
Preparing an ECKD type DASD for use

Before you can use an ECKD type DASD as a disk for Linux on Z, you must format it with a suitable disk layout. You must then create a file system or define a swap space.

Before you begin

- The 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 662) to find out if the device is online.
   If necessary, set the device online, see “Setting a DASD online or offline” on page 157.

2. Format the device with the `dasdfmt` command (see “dasdfmt - Format a DASD” on page 615 for details). The formatting process can take hours for large DASDs.
   If you want to use the CMS disk layout, and your DASD is already formatted with the CMS disk layout, skip this step.

   Tips:
   - Use the largest possible block size, ideally 4096; the net capacity of an ECKD DASD decreases for smaller block sizes. For example, a DASD formatted with a block size of 512 byte has only half of the net capacity of the same DASD formatted with a block size of 4096 byte.
   - For DASDs that have previously been formatted with the cd1 or ld1 disk layout, use the dasdfmt quick format mode.
   - Use the -p option to display a progress bar.

   Example: Assuming that /dev/dasdzzz is a valid device node for 0.0.b100:

```
# dasdfmt -b 4096 -p /dev/dasdzzz
```

3. Proceed according to your chosen disk layout:

   - If you have formatted your DASD with the Linux disk layout or the CMS disk layout, skip this step and continue with step “4” on page 152. 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 633 for details).

   Example: To start the partitioning tool in interactive mode for partitioning a device /dev/dasdzzz issue:

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

   If you create three partitions for a DASD /dev/dasdzzz, the device nodes for the partitions are /dev/dasdzzz1, /dev/dasdzzz2, and /dev/dasdzzz3.

   Result: fdasd creates the partitions and updates the partition table (see “VTOC” on page 140).

4. Depending on the intended use of each partition, create a file system on the partition or define it as a swap space.
Either create a file system 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 disk for Linux on Z, 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 18 on page 143 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:**
   ```bash
   # 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.
   **Tip:** Mount file systems on FBA devices that are backed by z/VM VDISKs with the discard mount option. This option frees memory when data is deleted from the device.
   **Examples:**
   - To mount a file system in a partition `/dev/dasdzzy1`, issue:
     ```bash
     # mount /dev/dasdzzy1 /mnt
     ```
   - To mount a VDISK-backed file system in a partition `/dev/dasdzzy1`, and use the discard option to free memory when data is deleted, issue:
     ```bash
     # mount -o discard /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 155 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 10).

**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:
   ```bash
   # 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.
   ```bash
   # 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 733.

### 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 146).
- 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 157).
**Note:** When switching between an enabled and a disabled DIAG access method on FBA-type DASD, first reinitialize the DASD, for example, with CMS format or by overwriting any previous content. Switching without initialization might cause data-integrity problems.

For more details about DIAG, see *z/VM: CP Programming Services*, SC24-6272.

**Example**

In this example, the DIAG access method is enabled for a DASD with device number 0xb100.

1. Ensure that the driver is loaded (only applicable when compiled as module):
   ```bash
   # modprobe dasd_diag_mod
   ```

2. Identify the sysfs CCW-device directory for the device in question and change to that directory:
   ```bash
   # cd /sys/bus/ccw/devices/0.0.b100/
   ```

3. Ensure that the device is offline:
   ```bash
   # echo 0 > online
   ```

4. Enable the DIAG access method for this device by writing '1' to the use_diag sysfs attribute:
   ```bash
   # echo 1 > use_diag
   ```

5. Use the online attribute to set the device online:
   ```bash
   # 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 146).
- A misc character device (see “Assuring that a device node exists for extended error reporting” on page 150).
- 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:

```bash
# 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:
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 150). The device supports these file operations:

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

close
You can close the node as usual.

read
Blocking read and non-blocking read are supported. When a record is partially read and then purged, the next read returns an I/O error -EIO.

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

Setting a DASD online or offline

Use the chzdev command, the chccwdev command or the online sysfs attribute of the device to set DASDs online or offline.

About this task

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

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 chzdev command (“chzdev - Configure IBM Z devices” on page 589) to set a DASD online or offline.

Alternatively, use the chccwdev command, or write 1 to the device's sysfs online attribute to set it online or 0 to set it offline. In contrast to the sysfs attribute, the chccwdev command triggers a cio_settle for you and waits for the cio_settle to complete.

Outstanding I/O requests are canceled when you set a device offline. To wait indefinitely for outstanding I/O requests to complete before setting the device offline, use the chccwdev option --safeoffline or the sysfs attribute safe_offline. The chzdev command uses safe offline (if available), unless you specify the --force option.

Examples

• To set a DASD with device bus-ID 0.0.b100 online, issue:

# chzdev  -e dasd 0.0.b100

This setting persists across re-boots. For more details, see Chapter 29, “Persistent device configuration,” on page 377.
To apply this setting to the running system only, use the chzdev command with the -a option or one of the following commands:

# chccwdev  -e dasd 0.0.b100

DASD device driver 157
• To set a DASD with device bus-ID 0.0.b100 offline, issue:

```
# echo 1 > /sys/bus/ccw/devices/0.0.b100/online
```

This setting persists across re-boots. For more details, see Chapter 29, “Persistent device configuration,” on page 377.

To apply this setting to the running system only, use the chzdev command with the -a option or one of the following commands:

```
# chzdev -d dasd 0.0.b100
```

• 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 instance of Linux on Z, 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 147). 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 17).

**Attention:** Do not detach a device that is still being used by Linux. Detaching devices might cause the system to hang or crash. Ensure that you unmount a device and set it offline before you detach it.

See “Working with newly available devices” on page 10 to avoid errors when working with devices that have become available to a running Linux instance.

Be careful to avoid errors when working with devices that have become available to a running Linux instance.

**Enabling and disabling logging**

Use the dasd= kernel or module parameter or use the erplog sysfs attribute to enable or disable error recovery processing (ERP) logging.

**Procedure**

You can enable and disable error recovery processing (ERP) logging on a running system. There are two methods:
• Use the dasd= parameter when you load the base module of the DASD device driver.

  **Example:**
  
  To define a device range (0.0.7000-0.0.7005) and enable 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:**
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:
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 146).
- 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 620).

About this task

The DASD performance data is contained in the following subdirectories of <mountpoint>/dasd, where <mountpoint> is the mount point of debugfs:

- A directory global that represents all available DASDs taken together.
- For each DASD, one directory with the name of the DASD block device with which the DASD is known to the DASD device driver (for example, dasda, dasdb, and dasdc).
- For each CCW device that corresponds to a DASD, a directory with the bus ID as the name.

Block devices that are not set up for PAV or HyperPAV map to exactly one CCW device and the corresponding directories contain the same statistics.

With PAV or HyperPAV, a bus ID can represent a base device or an alias device. Each base device is associated with a particular block device. The alias devices are not permanently associated with the same block device. At any one time, a DASD block device is associated with one or more CCW devices. Statistics that are based on bus ID, therefore, show more detail for PAV and HyperPAV setups.

Each of these directories contains a file statistics that you can use to perform these tasks:

- Start and stop data gathering.
- Reset statistics counters.
- Read statistics.

To control data gathering at the scope of a directory in <mountpoint>/dasd, issue a command of this form:

```
# echo <max_wait> > /sys/bus/ccw/devices/<device_bus_id>/expires
# echo <total_max> > /sys/bus/ccw/devices/<device_bus_id>/timeout
```

where:

- `<max_wait>`
  is the new maximum response time, in seconds, for the storage server. The value must be a positive integer.

- `<total_max>`
  is the new maximum total time in seconds. The value must be a positive integer or 0. 0 disables this timeout setting.

- `<device_bus_id>`
  is the device bus-ID of the DASD.

Example:

```
# echo 60 > /sys/bus/ccw/devices/0.0.7008/exppires
# 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.
# 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 debugfs 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_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_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_hpp**
  - 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_hpp**
  - 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, divided 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.

**Scenario: Verifying that PAV and HPF are used**
Use the dasdstat command to display DASD performance statistics, including statistics about Parallel Access Volume (PAV) and High Performance FICON (HPF).

**Procedure**
1. Enable DASD statistics for the device of interest.

  **Example:**
2. Assure that I/O requests are directed to the device.

Hints:

- Access a partition, rather than the whole device, to avoid directing the I/O request towards the first 2 tracks of a CDL formatted DASD. Requests to the first 2 tracks of a CDL formatted DASD are exceptional in that they never use High Performance FICON.
- Assure that a significant I/O load is applied to the device. PAV aliases are used only if multiple I/O requests for the device are processed simultaneously.

Example:

```
# dd if=/dev/dasdc1 of=/dev/null bs=4k count=256
```

3. Look for PAV and HPF in the statistics.

Example:

```
# dasdstat dasdc
```

```
statistics data for statistic: dasdc
start time of data collection: Fri Dec 11 14:22:18 CET 2015
7 dasd I/O requests
with 4000 sectors(512B each)
3 requests used a PAV alias device
7 requests used HPF
```

In the example, dasdc uses both Parallel Access Volume and High Performance FICON.

Accessing full ECKD tracks

In raw-track access mode, the DASD device driver accesses full ECKD tracks, including record zero and the count and key data fields.

Before you begin

- This section applies to ECKD type DASD only.
- The DASD 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.
Handling lost device reservations

A DASD reservation by your Linux instance can be lost if another system unconditionally reserves this DASD.

About this task

This other system then has exclusive I/O access to the DASD for the duration of the unconditional reservation. Such unconditional reservations can be useful for handling error situations where:

- Your Linux instance cannot gracefully release the DASD.
- Another system requires access to the DASD, for example, to perform recovery actions.

After the DASD is released by the other system, your Linux instance might process pending I/O requests and write faulty data to the DASD. How to prevent pending I/O requests from being processed depends on the reservation policy. There are two reservation policies:

**ignore**

All I/O operations for the DASD are blocked until the DASD is released by the second system. When using this policy, reboot your Linux instance before the other system releases the DASD. This policy is the default.

**fail**

All I/O operations are returned as failed until the DASD is set offline or until the reservation state is reset. When using this policy, set the DASD offline and back online after the problem is resolved. See “Reading and resetting the reservation state” on page 168 about resetting the reservation state to resume operations.

Procedure

Set the reservation policy with a command of this form:

```bash
# 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 167).

- **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 167). 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 167) and only while the value of the `last_known_reservation_state` attribute is lost.

To find out the reservation state of a DASD issue a command of this form:

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

where `<device_bus_id>` specifies the DASD.

**Example**

The command in this example queries the reservation state of a DASD with bus ID 0.0.7009.

```
# cat /sys/bus/ccw/devices/0.0.7009/last_known_reservation_state
reserved
```

**Setting defective channel paths offline automatically**

Control the automatic removal of defective channel paths through the `path_threshold`, `path_interval`, and `path_autodisable` sysfs attributes.

**About this task**

A channel control check (CCC) is caused by any machine malfunction that affects channel-subsystem controls. An interface control check (IFCC) indicates that an incorrect signal occurred on the channel path. Usually, these errors can be recovered automatically.

However, IFCC or CCC errors that occur frequently on a particular channel path indicate a defective channel path. Error recovery processing on defective channel paths can result in performance degradation. If at least one operational channel path remains, overall device performance might improve if a defective channel path is excluded from I/O.

By default, automatic path removal is enabled with an error threshold of 256 and a reset interval of 300 s (5 minutes). Accordingly, a channel path is set offline automatically, when the error count reaches 256 and if at least one other channel path remains. If 300 seconds elapse without an error, the error count is reset to 0.

You can change the error threshold and reset interval, or you can prevent automatic removal of channel paths altogether.

**Procedure**

- To specify the number of errors that must occur before the channel path is taken offline, issue a command of this form:

```
# echo <no_of_errors> > /sys/bus/ccw/devices/<device_bus_id>/path_threshold
```

where `/sys/bus/ccw/devices/<device_bus_id>` represents the device in sysfs and `<no_of_errors>` is an integer that specifies the error threshold.

To disable detecting defective paths, and to suppress messages about IFCC or CCC errors, set `<no_of_errors>` to 0.

- To specify the time that must elapse without errors to trigger a counter reset, issue a command of this form:

```
# echo <time> > /sys/bus/ccw/devices/<device_bus_id>/path_interval
```

where `<time>` is the reset interval in seconds.
To prevent defective paths from being set offline automatically, issue a command of this form:

```
# echo <flag> > /sys/bus/ccw/devices/<device_bus_id>/path_autodisable
```

where `<flag>` can be "1" to enable automatic path removal, or "0" to prevent automatic path removal. By default, automatic path removal is enabled.

**Examples**

- Setting 512 for the error threshold and 6 minutes (360 s) for the reset interval:

```
echo 512 > /sys/bus/ccw/devices/0.0.4711/path_threshold
echo 360 > /sys/bus/ccw/devices/0.0.4711/path_interval
```

According to this example, a channel path is automatically removed if a count of 512 IFCCs or CCCs is reached. Any 6-minute interval without a IFCCs or CCCs causes the counter to be reset to zero.
- Preventing automatic removal of defective channel paths:

```
# echo 0 > /sys/bus/ccw/devices/0.0.4711/path_autodisable
```

In this example, messages about defective paths are issued according to the settings for the error threshold and the reset interval, but defective paths are not removed automatically.

**What to do next**

After repairing a channel path that was set offline, set it online again by using the `tunedasd` command with the `-p` option. See “tunedasd - Adjust low-level DASD settings” on page 733 for details.

**Querying the HPF setting of a channel path**

Query the High Performance FICON (HPF) state of a channel path through the `hpf` sysfs attribute. The HPF function can be lost if the device cannot provide the function, or if the channel path is not able to do HPF.

**About this task**

The HPF channel-path is deactivated if an HPF error occurs indicating that HPF is not available if there are other channel paths available. If no other channel paths are available, the path remains operational with HPF deactivated.

If the device loses HPF functionality, HPF is disabled for all channel paths defined for the device.

**Procedure**

To query the HPF function for a channel path, issue a command of this form:

```
# lsdasd -l <device_bus_id>
```

Alternatively, you can query the sysfs attribute directly:

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

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

**Example**

To query the availability of HPF for a device 0.0.4711, issue:
lsdasd -l 0.0.4711

0.0.4711/dasdc/94:8
  status:                           active
  type:                             ECKD
  hpf:                              i

This example indicates that HPF is enabled for the device.

Alternatively, read from the hpf sysfs attribute:

```
cat /sys/bus/ccw/devices/0.0.4712/hpf
```

This example indicates that HPF is disabled for device 0.0.4712.

**What to do next**

You can now reset the paths to the device. You can use the tunedasd command to reset all or one channel path.

To re-validate all paths for one device and if possible reset HPF:

```
# tunedasd --path_reset_all /dev/dasdc
Reseting all chpids for device </dev/dasdc> ...
Done.
```

See “tunedasd - Adjust low-level DASD settings” on page 733 for details.

You can also use sysfs to reset a path. sysfs expects a path mask. For example to reset CHPID 44, you can use tunedasd:

```
tunedasd -p 44 /dev/dasde
```

This would be the same as specifying the following in sysfs:

```
echo 08 > /sys/bus/ccw/devices/0.0.9330/path_reset
```

Both commands will reset CHPID 44 (path mask 08).

**Checking for access by other operating system instances**

Query if a DASD volume is online to another operating system instance by reading the `host_access_count` attribute.

**Before you begin**

To query the number of operating system instances that use the DASD device, the DASD must be online.

**About this task**

Storage servers that support this feature know about the online status of the device on all attached operating system instances in an LPAR (so called hosts). If a DASD device is set online it might potentially be used on another operating system instance. This information can help to reduce the chance for outages or possible data corruption due to concurrent access to DASD volumes from different operating system instances.

**Procedure**

To check whether a DASD device is being used by other operating system instances, issue a command of this form:
where /sys/bus/ccw/devices/<device_bus_id> represents the device in sysfs.

For example, to query how many operating system instances have access to a device 0.0.bf45, issue:

```bash
# cat /sys/bus/ccw/devices/0.0.bf45/host_access_count
13
```

In the example, 13 operating system instances have access to the device, including the current Linux instance.

**What to do next**

To see details for each host connected to the DASD device, use the `lsd sad` command with the `--host-access-list` option. For more information and an example, see “lsd sad - List DASD devices” on page 662.

**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 `lsd sad -l` (see “lsd sad - List DASD devices” on page 662) to display summary information about the device settings and the device geometry of multiple DASDs.
- Use `dasdview` (see “dasdview - Display DASD structure” on page 622) to display details about the contents of a particular DASD.
- Read information about a particular DASD from sysfs, as described in this section.

The sysfs representation of a DASD is a directory of the form `/sys/bus/ccw/devices/<device_bus_id>`, where `<device_bus_id>` is the bus ID of the DASD. This sysfs directory contains a number of attributes with information about the DASD.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>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.</td>
</tr>
<tr>
<td>discipline</td>
<td>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.</td>
</tr>
<tr>
<td>eer_enabled</td>
<td>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 156).</td>
</tr>
<tr>
<td>erplog</td>
<td>1 if error recovery processing (ERP) logging is enabled, 0 if ERP logging is not enabled (see “Enabling and disabling logging” on page 158).</td>
</tr>
<tr>
<td>Attribute</td>
<td>Explanation</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>expires</td>
<td>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 160).</td>
</tr>
<tr>
<td>failfast</td>
<td>1 if I/O operations are returned as failed immediately when the last path to the DASD is lost. 0 if a wait period for a path to return expires before an I/O operation is returned as failed. See “Enabling and disabling immediate failure of I/O requests” on page 159.</td>
</tr>
<tr>
<td>host_access_count</td>
<td>Shows how many operating system instances have access to the device. See “Checking for access by other operating system instances” on page 171.</td>
</tr>
<tr>
<td>hpf</td>
<td>1 if High Performance FICON is available for the device. See “Querying the HPF setting of a channel path” on page 170.</td>
</tr>
</tbody>
</table>
| 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 168 for details. |
<p>| online                       | 1 if the DASD is online, 0 if it is offline (see “Setting a DASD online or offline” on page 157).                                           |
| path_autodisable             |                                                                                                                                             |
| path_interval                | Control the automatic removal of defective channel path (see “Setting defective channel paths offline automatically” on page 169)            |
| path_threshold               |                                                                                                                                             |
| 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 165).               |
| 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 167 for details. |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| 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 160).  
| 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 155). 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.  

174 Linux on Z and LinuxONE: Device Drivers, Features, and Commands Development stream (Kernel 4.19)
There are some more attributes that are common to all CCW devices (see “Device attributes” on page 9).

**Procedure**

Issue a command of this form to read an attribute:

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

where `<attribute>` is one of the attributes of Table 20 on page 172.

**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/host_access_count
1
# cat /sys/bus/ccw/devices/0.0.b100/hpf
1
# cat /sys/bus/ccw/devices/0.0.b100/last_known_reservation_state
reserved
# cat /sys/bus/ccw/devices/0.0.b100/online
1
# cat /sys/bus/ccw/devices/0.0.b100/path_autodisable
1
# cat /sys/bus/ccw/devices/0.0.b100/path_interval
300
# cat /sys/bus/ccw/devices/0.0.b100/path_threshold
256
# cat /sys/bus/ccw/devices/0.0.b100/raw_track_access
0
# cat /sys/bus/ccw/devices/0.0.b100/readonly
1
# cat /sys/bus/ccw/devices/0.0.b100/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 13. SCSI-over-Fibre Channel device driver

The SCSI-over-Fibre Channel device driver for Linux on Z (zfcp device driver) supports virtual QDIO-based SCSI-over-Fibre Channel adapters (FCP devices) and attached SCSI devices (LUNs).

Z adapter hardware typically provides multiple channels, with one port each. You can configure a channel to use the Fibre Channel Protocol (FCP). This FCP channel is then virtualized into multiple FCP devices. Thus, an FCP device is a virtual QDIO-based SCSI-over-Fibre Channel adapter with a single port.

A single physical port supports multiple FCP devices. Using N_Port ID virtualization (NPIV) you can define virtual ports and establish a one-to-one mapping between your FCP devices and virtual ports (see “N_Port ID Virtualization for FCP channels” on page 182).

On Linux, an FCP device is represented by a CCW device that is listed under /sys/bus/ccw/drivers/zfcp. Do not confuse FCP devices with SCSI devices. A SCSI device is identified by a LUN.

Features

The zfcp device driver supports a wide range of SCSI devices, various hardware adapters, specific topologies, and specific features that depend on the Z hardware.

• Linux on 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)
  – FICON Express16S (as of z13)
  – FICON Express16S+ (as of z14)

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 Z.

• The zfcp device driver supports switched fabric and point-to-point topologies. You can use either of these topologies, provided that it is supported by your storage server.

• 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.

To find out whether a combination of device, Linux distribution, and IBM mainframe is supported, see the individual interoperability matrix for each storage device. The interoperability matrices are available at:

www.ibm.com/systems/support/storage/ssic/interoperability.wss

For information about 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 58 on page 178 illustrates how the device drivers work together.

![Diagram showing the relationship between Linux, SCSI stack, and zfcp device driver]

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 201.
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 687 for more details about the command.

See also “Mapping the representations of a SCSI device in sysfs” on page 203.

**SCSI disk device nodes**

User space programs access SCSI disks through device nodes.

Device names for SCSI disks 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 z Systems*, 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>-fc-<wwpn>-lun-<lun>
```

for whole SCSI device and

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

for the `<n>`th partition, where `<wwpn>` is the worldwide port number of the target port and `<lun>` is the logical unit number that represents the target SCSI device.

**Note:** The format of these udev-created device nodes has changed and now matches the common code format. Device nodes of the prior form, `ccw-<device_bus_id>-zfcp-<wwpn>:<lun>` or `ccw-<device_bus_id>-zfcp-<wwpn>:<lun>-part<n>`, are no longer created.
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-fc-0x500507630300c562-1un-0x401040ea00000000
- /dev/disk/by-id/scsi-36005076303ffc562000000000000010ea
- /dev/disk/by-id/wwn-0x6005076303ffc562000000000000010ea

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-fc-0x500507630300c562-1un-0x401040ea00000000-part1
- /dev/disk/by-id/scsi-36005076303ffc562000000000000010ea-part1
- /dev/disk/by-uuid/7eaf9c95-55ac-4e5e-8f18-065b313e63ca
- /dev/disk/by-id/wwn-0x6005076303ffc562000000000000010ea-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-fc-0x500507630300c562-1un-0x401040ea00000000-part2
- /dev/disk/by-id/scsi-36005076303ffc562000000000000010ea-part2
- /dev/disk/by-uuid/b4a818c8-747c-40a2-bfa2-acaa3ef70ead
- /dev/disk/by-id/wwn-0x6005076303ffc562000000000000010ea-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 block device that corresponds to a SCSI disk” on page 209 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 block device that corresponds to a SCSI disk” on page 209.

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

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

zfcp HBA API (FC-HBA) support
The zfcp host bus adapter API (HBA API) provides an interface for HBA management clients that run on Z. As shown in Figure 59 on page 182, the zfcp HBA API support includes a user space library.
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 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 186.

N_Prot 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 189 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 z Systems, SC33-8413.

N_Prot 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 60 on page 183 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 60: 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)
- `CONFIG_SCSI_FC_ATTRS`

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, or GPT disk layout support.
  
  Partitioning is only possible if PC-BIOS disk layout support, or GPT disk layout support, is compiled into the kernel. For partitions greater than 2 TB, you must use the GPT disk layout.

  - SCSI tapes support
  - SCSI media changer support
  - SCSI CD-ROM and ISO 9660 file system
  - SCSI generic support
  - Block Layer 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. For information about specifying kernel parameters, see “Specifying kernel parameters” on page 23.
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.

Notes:
1 For experimental use only. Do not use on production systems.

Figure 61: zfcp kernel parameter syntax

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.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.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.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.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 dbfsize is increased to 6 pages:
    zfcp.dbfsize=6
    This results in six pages being used for each area and FCP device.

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.port_scan_ratelimit=<limit>
    sets the minimum delay, in milliseconds, between automatic port scans of your Linux instance. The
    default value is 60000 milliseconds. To turn off the rate limit, specify 0. Use this parameter to avoid
    frequent scans, while you still ensure that a scan is conducted eventually.

zfcp.port_scan_backoff=<delay>
    sets additional random delay, in milliseconds, in which the port scans of your Linux instance are
    spread. The default value is 500 milliseconds. To turn off the random delay, specify 0. In an
    installation with multiple Linux instances, use this attribute for every Linux instance to spread scans to
    avoid potential multiple simultaneous scans.

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 FCP device is set online and when user-triggered
    writes to the sysfs attribute port_rescan occur.
    Note: As of kernel 4.0 and the introduction of port_scan_backoff and port_scan_ratelimit,
    use them rather than no_auto_port_rescan.

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 210.

Example
Use the following kernel parameter to enable end-to-end data consistency checking:

```
zfcp.dif=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. For more information about specifying parameters, see “Specifying module parameters” on page 28.

```
modprobe zfcp
device = <device_bus_id>,<wwpn>,<fcp_lun>
allow_lun_scan = 1
allow_lun_scan = <value>
datarouter = 1
datarouter = 0
dbflevel = 3
dbflevel = <level>
dbfsize = 4
dbfsize = <pages>
dif = 0
port_scan_ratelimit = 60000
port_scan_ratelimit = <limit>
port_scan_backoff = 500
port_scan_backoff = <limit>
no_auto_port_rescan = 0
no_auto_port_rescan = 1
queue_depth = 32
queue_depth = <depth>
```

Notes:
1 For experimental use only. Do not use on production systems.

Figure 62: zfcp module parameter syntax

The variables have the same meaning as the corresponding kernel parameters with prefix zfcp. in “zfcp device driver kernel parameters” on page 183.

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:
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 libzfcp-hbaapi-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 libzfcpbhaapi 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/zfcp-hbaapi-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 218.

Working with FCP devices
Set an FCP device online before you attempt to perform any other tasks.

Working with FCP devices comprises the following tasks:

• “Setting an FCP device online or offline” on page 188
• “Displaying FCP channel and device information” on page 189
• “Recovering a failed FCP device” on page 193
• “Finding out whether NPIV is in use” on page 193
• “Logging I/O subchannel status information” on page 194
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 10 to avoid errors when you work with devices that have become available to a running Linux instance.

**Important:** Configuration changes can directly or indirectly affect information that is required to mount the root file system. Such changes require an update of the initramfs, followed by a re-write of the boot record.

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**

For a persistent configuration, use the chzdev command.

For a non-persistent configuration, you can use:

- The chzdev command with the -a option.
- The chccwdev command.
- The sysfs attribute online. Write 1 to an FCP device's online attribute to set it online, or 0 to set it offline.

**Examples**

- For a persistent configuration, to set an FCP device with bus ID 0.0.3d0c online, issue:

  ```shell
  # chzdev -e zfcp-host 0.0.3d0c
  ```

  This setting persists across re-boots. For more details, see Chapter 29, “Persistent device configuration,” on page 377. To apply this setting to the running system only, use the chzdev command with the -a option:
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>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>card_version</td>
<td>Version number that identifies a particular hardware feature.</td>
</tr>
<tr>
<td>hardware_version</td>
<td>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.</td>
</tr>
<tr>
<td>lic_version</td>
<td>Microcode level.</td>
</tr>
<tr>
<td>peer_wwnn</td>
<td>WWNN of peer for a point-to-point connection.</td>
</tr>
<tr>
<td>peer_wwpn</td>
<td>WWPN of peer for a point-to-point connection.</td>
</tr>
<tr>
<td>peer_d_id</td>
<td>Destination ID of the peer for a point-to-point connection.</td>
</tr>
</tbody>
</table>
### Table 22: Attributes with FCP device information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_recovery</td>
<td>Shows if the FCP device is in recovery (0 or 1).</td>
</tr>
</tbody>
</table>

For the attributes availability, cmb_enable, and cutype, see “Device directories” on page 9. The status attribute is reserved.

### Table 23: Relevant transport class attributes, fc_host attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxframe_size</td>
<td>Maximum frame size.</td>
</tr>
<tr>
<td>node_name</td>
<td>Worldwide node name (WWNN).</td>
</tr>
<tr>
<td>permanent_port_name</td>
<td>WWPN associated with the physical port of the FCP channel.</td>
</tr>
<tr>
<td>port_id</td>
<td>A unique ID (N_Port_ID) assigned by the fabric. In an NPIV setup, each virtual port is assigned a different port_id.</td>
</tr>
<tr>
<td>port_name</td>
<td>WWPN associated with the FCP device. If N_Port ID Virtualization is not available, the WWPN of the physical port (see permanent_port_name).</td>
</tr>
<tr>
<td>port_type</td>
<td>The port type indicates the topology of the port.</td>
</tr>
<tr>
<td>serial_number</td>
<td>The 32-byte serial number of the adapter hardware that provides the FCP channel.</td>
</tr>
<tr>
<td>speed</td>
<td>Speed of FC link.</td>
</tr>
<tr>
<td>supported_classes</td>
<td>Supported FC service class.</td>
</tr>
<tr>
<td>supported_speeds</td>
<td>Supported speeds.</td>
</tr>
<tr>
<td>symbolic_name</td>
<td>The symbolic port name that is registered with the FC name server.</td>
</tr>
<tr>
<td>tgid_bind_type</td>
<td>Target binding type.</td>
</tr>
</tbody>
</table>

### Table 24: Relevant transport class attributes, fc_host statistics

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset_statistics</td>
<td>Writeable attribute to reset statistic counters.</td>
</tr>
<tr>
<td>seconds_since_last_reset</td>
<td>Seconds since last reset of statistic counters.</td>
</tr>
<tr>
<td>tx_frames</td>
<td>Transmitted FC frames.</td>
</tr>
<tr>
<td>tx_words</td>
<td>Transmitted FC words.</td>
</tr>
<tr>
<td>rx_frames</td>
<td>Received FC frames.</td>
</tr>
<tr>
<td>rx_words</td>
<td>Received FC words.</td>
</tr>
<tr>
<td>lip_count</td>
<td>Number of LIP sequences.</td>
</tr>
<tr>
<td>nos_count</td>
<td>Number of NOS sequences.</td>
</tr>
<tr>
<td>error_frames</td>
<td>Number of frames that are received in error.</td>
</tr>
</tbody>
</table>
### Table 24: Relevant transport class attributes, fc_host statistics (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>dumped_frames</td>
<td>Number of frames that are lost because of lack of host resources.</td>
</tr>
<tr>
<td>link_failure_count</td>
<td>Link failure count.</td>
</tr>
<tr>
<td>loss_of_sync_count</td>
<td>Loss of synchronization count.</td>
</tr>
<tr>
<td>loss_of_signal_count</td>
<td>Loss of signal count.</td>
</tr>
<tr>
<td>prim_seq_protocol_err_count</td>
<td>Primitive sequence protocol error count.</td>
</tr>
<tr>
<td>invalid_tx_word_count</td>
<td>Invalid transmission word count.</td>
</tr>
<tr>
<td>invalid_crc_count</td>
<td>Invalid CRC count.</td>
</tr>
<tr>
<td>fcp_input_requests</td>
<td>Number of FCP operations with data input.</td>
</tr>
<tr>
<td>fcp_output_requests</td>
<td>Number of FCP operations with data output.</td>
</tr>
<tr>
<td>fcp_control_requests</td>
<td>Number of FCP operations without data movement.</td>
</tr>
<tr>
<td>fcp_input_megabytes</td>
<td>Megabytes of FCP data input.</td>
</tr>
<tr>
<td>fcp_output_megabytes</td>
<td>Megabytes of FCP data output.</td>
</tr>
</tbody>
</table>

### Procedure

Use the `cat` command to read an attribute.

- Issue a command of this form to read an attribute:

  ```bash
  # 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 21 on page 189 or Table 22 on page 190.

- To read attributes of the associated Fibre Channel host use:

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

  where:

  `<host_name>` is the ID of the Fibre Channel host.

  `<attribute>` is one of the attributes in Table 23 on page 190.

- To read statistics attributes of the FCP channel associated with this Fibre Channel host, use:

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

  where:

  `<host_name>` is the ID of the Fibre Channel host.

  `<attribute>` is one of the attributes in Table 24 on page 190.
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 687) to display attributes of an FCP channel:

  # lszfcp -b 0.0.3d0c -a
  0.0.3d0c host0
  Bus = "ccw"
  availability = "good"
  card_version = "0x0007"
  cmb_enable = "0"
  cutype = "1731/03"
  devtype = "1732/03"
  failed = "0"
  hardware_version = "0x00000000"
  in_recovery = "0"
  lic_version = "0x00000302"
  modalias = "ccw:tl1731m03dt1732dm03"
  online = "1"
  peer_d_id = "0x000000"
  peer_wnnn = "0x0000000000000000"
  peer_wwpn = "0x0000000000000000"
  status = "0x5400040b"
  uevent = "DRIVER=zfcp"
  Class = "fc_host"
  active_fc4s = "0x00 0x00 0x01 0x00 ..."
  dev_loss_tmo = "60"
  maxframe_size = "2112 bytes"
  node_name = "0x5005076400c7ec87"
  permanent_port_name = "0xc05076ffd6801981"
  port_id = "0x671a29"
  port_name = "0xc05076ffd6801e10"
  port_state = "Online"
  port_type = "NPIV VPOR"
  serial_number = "IBM0200000007EC07"
  speed = "16 Gbit"
  supported_classes = "Class 2, Class 3"
  supported_fc4s = "0x00 0x00 0x01 0x00 ..."
  supported_speeds = "4 Gbit, 8 Gbit, 16 Gbit"
  symbolic_name = "IBM type serial PCHID: 0198 NPIV UlpId: 05600300 DEVNO: 0.0.3d0c NAME:
  hostname.domain"
  tgtid_bind_type = "wwpn (World Wide Port Name)"
  Class = "scsi_host"
  active_mode = "Initiator"
  can_queue = "4096"
  cmd_per_lun = "0"
  eh_deadline = "off"
  host_busy = "0"
  megabytes = "15 0"
  proc_name = "zfcp"
  prot_capabilities = "0"
  prot_guard_type = "0"
  queue_full = "0 2357653"
  requests = "815 0 66"
  seconds_active = "11"
  sg_prot_tablesize = "0"
  sg_tablesize = "574"
  state = "running"
  supported_mode = "Initiator"
  unchecked_isa_dma = "0"
  unique_id = "6400"
  use_blk_mq = "0"
  utilization = "0 0 0"
**Recovering a failed FCP device**

Failed FCP devices are automatically recovered by the zfcp device driver. You can read the `in_recovery` attribute to check whether recovery is under way.

**Before you begin**
The FCP device must be online.

**Procedure**
Perform these steps to find out the recovery status of an FCP device and, if needed, start or restart recovery:

1. Issue a command of this form:

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

   The value is 1 if recovery is under way and 0 otherwise. If the value is 0 for a non-operational FCP device, recovery might have failed. Alternatively, the device driver might have failed to detect that the FCP device is malfunctioning.

2. To find out whether recovery failed, read the `failed` attribute. Issue a command of this form:

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

   The value is 1 if recovery failed and 0 otherwise.

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

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

**Example**
In the following example, an FCP device with a device bus-ID 0.0.3d0c is malfunctioning. The first command reveals that recovery is not already under way. The second command manually starts recovery for the FCP device:

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

**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.
Issue a command of the following form:

```
# cat /sys/bus/ccw/drivers/zfcp/<device_bus_id>/host*/fc_host/host*/port_type
```

For example:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.3d0c/host*/fc_host/host*/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 687) to list the FCP device attributes.

**Example**

```
# lszfcp -b 0.0.3d0c -a
0.0.3d0c host0
Bus = "ccw"
  availability = "good"
...
Class = "fc_host"
...
node_name           = "0x5005076400c7ec87"
permanent_port_name = "0xc05076ffd6801981"
port_id             = "0x671a29"
port_name           = "0xc05076ffd6801e10"
port_state          = "Online"
port_type           = "NPIV VPORT"
...
symbolic_name       = "IBM type serial PCHID: 0198 NPIV UlpId: 05600300 DEVNO: 0.0.3d0c NAME:
hostname.domain"
...
```

The `port_type` attribute directly indicates that NPIV is used. The example also shows that `permanent_port_name` is different from `port_name` and neither is NULL. The example also shows the `symbolic_name` attribute that shows the symbolic port name that was registered on the FC name server.

**Logging I/O subchannel status information**

When severe errors occur for an FCP device, the FCP device driver triggers a set of log entries with I/O subchannel status information.

The log entries are available through the SE Console Actions Work Area with the View Console Logs function. In the list of logs, these entries have the prefix `1F00`. The content of the entries is intended for support specialists.

**Working with target ports**

You can scan for ports, display port information, recover a port, or remove a port.

Working with target ports comprises the following tasks:

- “Scanning for ports” on page 194
- “Controlling automatic port scanning” on page 195
- “Displaying port information” on page 197
- “Recovering a failed port” on page 199
- “Removing ports” on page 200

**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 186.

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 FCP device configuration or if you are unsure whether all ports were added to sysfs.

**Procedure**

Issue a command of this form:

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

where `<device_bus_id>` specifies the FCP device through which the target ports are attached.

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

**Example**

In this example, a port with WWPN 0x500507630303c562 is already configured for an FCP device with bus ID 0.0.3d0c. An additional target port with WWPN 0x500507630300c562 is automatically configured by triggering a port scan.

```
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
# echo 1 > /sys/bus/ccw/drivers/zfcp/0.0.3d0c/port_rescan
# ls /sys/bus/ccw/drivers/zfcp/0.0.3d0c/0x*
0x500507630303c562
0x500507630300c562
```

**Controlling automatic port scanning**

Automatic port scanning includes two zfcp parameters that improve the behaviour of Linux instances in SANs. These zfcp parameters are set to default values that work well for most installations. If needed, you can fine-tune the frequency and timing of automatic port scans with the zfcp parameters port_scan_backoff and port_scan_ratelimit. You can enable automatic port scanning with the zfcp parameter no_auto_port_rescan=0. This value is the default.

**About this task**

In a large installation, where many Linux instances receive the same notifications of SAN changes, multiple instances might trigger scans simultaneously and too frequently. See Figure 63 on page 195

![Figure 63: Numerous port scans in a Linux installation](image)

These scans might put unnecessary load on the name server function of fabric switches and potentially result in late or inconclusive results.
You can avoid excessive scanning, yet still ensure that a port scan is eventually conducted. You can control port scanning with the zfcp parameters:

**port_scan_ratelimit**

sets the minimum delay, in milliseconds, between automatic port scans of your Linux instance. The default value is 60000 milliseconds. To turn off the rate limit, specify 0.

**port_scan_backoff**

sets an additional random delay, in milliseconds, in which the port scans of your Linux instance are spread. In an installation with multiple Linux instances, use this zfcp parameter for every Linux instance to spread scans to avoid potential multiple simultaneous scans. The default value is 500 milliseconds. To turn off the random delay, specify 0.

Use kernel or module parameters (see “zfcp device driver kernel parameters” on page 183). On a running Linux system, you can also query or set these values by using the sysfs attributes with the same names.

Using `port_scan_ratelimit` reduces the number of scans, as shown in Figure 64 on page 196.

![Figure 64: Port scan behavior with scan rate limit.](image)

However, if the rate limit is set to the same value, the scans can still occur almost simultaneously, as for FCP device A and B in Linux 1.

Using `port_scan_backoff` and `port_scan_ratelimit` together delays port scans even further and avoids simultaneous scans, as shown in Figure 65 on page 196. In the figure, FCP devices A and B in Linux 1 have the same rate limit and the same backoff values. The random element in the backoff value causes the scans to occur at slightly different times.

![Figure 65: Port scan behavior with backoff and scan rate limit.](image)

**Procedure**

Use `port_scan_backoff` and `port_scan_ratelimit` together or separately to tune the behavior of port scanning:

- To avoid too frequent scanning, set a minimum wait time between two consecutive scans for the same Linux instance. Use the `port_scan_ratelimit` sysfs attribute.

  By default, `port_scan_ratelimit` is turned on and has a value of 60000 milliseconds.

  For example, to specify an attribute value of 12 seconds, issue:

  ```bash
  # echo 12000 > /sys/module/zfcp/parameters/port_scan_ratelimit
  ```
• To further spread scans over a certain time and thus avoid multiple simultaneous scans, set the `port_scan_backoff` sysfs attribute.

By default, `port_scan_backoff` is turned on and has a value of 500 milliseconds.

For example, to query the setting, issue a command of this form:

```
# cat /sys/module/zfcp/parameters/port_scan_backoff
560
```

To set the attribute to 1 second, issue:

```
# echo 1000 > /sys/module/zfcp/parameters/port_scan_backoff
```

**Results**

The automatic port scans are delayed by the values specified. If a SAN notification is received during the rate limit time, a port scan is conducted immediately after the delay time passed.

Setting the attributes in sysfs is a useful method on a running system where you want to make dynamic changes. If you want to make the changes persistent across IPLs, you can:

• Use the kernel or module parameter.
• Use the appropriate tool or configuration file that is provided by your distribution.

Depending on the port event, one or more of the three zfcp parameters are evaluated to schedule a port scan. For example, port scans that are triggered manually through sysfs are not delayed. Table 25 on page 197 shows which events evaluate which zfcp parameters.

<table>
<thead>
<tr>
<th>Event</th>
<th>no_auto_port_rescan</th>
<th>port_scan_backoff</th>
<th>port_scan_ratelimit</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCP device resume</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>User sets FCP device online</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>User initiates a port scan</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>User starts FCP device recovery</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Automatic FCP device recovery</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SAN change notification</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Displaying port information**

For each target port, there is a number of read-only sysfs attributes with port information.

**About this task**

Table 26 on page 197 and Table 27 on page 198 summarize the relevant attributes.

**Table 26: zfcp-specific attributes with port information within the FCP device sysfs tree**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>access_denied</td>
<td>This attribute is obsolete. The value is always 0.</td>
</tr>
</tbody>
</table>
Table 26: zfcp-specific attributes with port information within the FCP device sysfs tree (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_recovery</td>
<td>Shows if port is in recovery (0 or 1)</td>
</tr>
</tbody>
</table>

Table 27: Transport class attributes with port information

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

Procedure

Use the `cat` command to read an attribute.

- Issue a command of this form to read a zfcp-specific attribute:

  ```bash
  # 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 197.

- To read transport class attributes of the associated target port, use a command of this form:

  ```bash
  # cat /sys/class/fc_remote_ports/<rport_name>/<attribute>
  ``

  where:
  
  `<rport_name>`
  
  is the name of the remote port.
  
  `<attribute>`
  
  is one of the attributes in Table 27 on page 198.

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 219.

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

```bash
# lszfcp -p 0x500507630300c562 -a
0.0.3d0c/0x500507630300c562 rport-0:0-0
Class = "fc_remote_ports"
  dev_loss_tmo        = "2147483647"
  fast_io_fail_tmo    = "5"
  maxframe_size       = "2112 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:

   ```bash
   # 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:

   ```bash
   # 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:

   ```bash
   # 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/0x500507630303c562/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 201) 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 186.

**Procedure**
Issue a command of this form:

```bash
# 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.

```bash
# 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 201
- “Mapping the representations of a SCSI device in sysfs” on page 203
Configuring SCSI devices

FCP devices that use NPIV mode detect the LUNs automatically and no configuring is necessary. If needed, configure the LUN manually.

You can configure the LUN manually using the chzdev command or by writing 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 183), the LUNs are configured for you. In this case, no FCP LUN entries are created under /sys/bus/ccw/drivers/zfcp/<device_bus_id>/wwpn.

To find out whether an FCP device is using NPIV mode, check the port_type attribute. For example:

```
# cat /sys/bus/ccw/drivers/zfcp/0.0.1901/host*/fc_host/host*/port_type
NPIV_VPORT
```

To find out whether automatic LUN scanning is enabled, check the current setting of the module parameter zfcp.allow_lun_scan. The example below shows automatic LUN scanning as turned on.

```
# cat /sys/module/zfcp/parameters/allow_lun_scan
Y
```

Automatically attached SCSI devices

FCP devices that use NPIV mode, and for which you did not disable automatic LUN scanning, 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 a SCSI device does not exist 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:** The following procedure ignores distribution specifics and might interfere with configuration tools provided by your distribution. Always use distribution-specific configuration tools if available.

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 4, “Kernel and module parameters,” on page 23

---

- “Displaying information about SCSI devices” on page 207
- “Finding the major and minor numbers for a block device that corresponds to a SCSI disk” on page 209
- “Setting the queue depth” on page 210
- “Recovering failed SCSI devices” on page 211
- “Updating the information about SCSI devices” on page 212
- “Setting the SCSI command timeout” on page 212
- “Controlling the SCSI device state” on page 213
- “Removing SCSI devices” on page 214
Procedure
If your FCP device does not use NPIV mode, or if you have disabled automatic LUN scanning, proceed as follows:

• Use the chzdev command.

To enable a FCP device and create a persistent configuration, issue:

```bash
# chzdev -e zfcp-lun <device_bus_id>:<wwpn>:<fcp_lun>
```

where:

- `<device_bus_id>` specifies the FCP device.
- `<wwpn>` is the WWPN of the target port.
- `<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.

This command starts a process with multiple steps:

1. It creates a directory in `/sys/bus/ccw/drivers/zfcp/<device_bus_id>/` 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 203).

• To enable a FCP device for the current configuration only, you can use either chzdev or sysfs:

  - To use chzdev with the `-a` option, issue a command of this form:
    ```bash
    # chzdev -e -a zfcp-lun <device_bus_id>:<wwpn>:<fcp_lun>
    ```

  - To use sysfs, configure a SCSI device for a target port and write the device's LUN to the port's `unit_add` attribute. Issue a command of this form:
    ```bash
    # echo <fcp_lun> > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/unit_add
    ```

Examples
For a persistent configuration, use chzdev. In this example, an FCP device with bus ID 0.0.3dc0 is enabled. The WWPN of the target port is 0x500507630300c562. A SCSI device with LUN 0x4010403300000000 is added to the port.

```bash
# chzdev -e zfcp-lun 0.0.3dc0:0x500507630300c562:0x4010403300000000
```

This creates a configuration that is persistent across boots.

For changes to the current configuration only, use chzdev with the `-a` option. In this example, the same configuration as in the previous example is made, but will not be persistent across re-boots:

```bash
# chzdev -e -a zfcp-lun 0.0.3dc0:0x500507630300c562:0x4010403300000000
```

For the current configuration only, you can also use sysfs. In this example, a target port with WWPN 0x500507630300c562 is attached through an FCP device with bus ID 0.0.3d0c. A SCSI device with LUN
0x4010403200000000 is already configured for the port. An additional SCSI device with LUN 0x4010403300000000 is added to the port.

```bash
# 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
0x4010403000000000
```

What to do next

To check whether a SCSI device is registered for the configured LUN, check for a directory with the name of the LUN in `/sys/bus/scsi/devices`. If there is no SCSI device for this LUN, the LUN is not valid in the storage system, or the FCP device is offline in Linux.

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

Mapping the representations of a SCSI device in sysfs

Each SCSI device that is configured is represented by multiple directories in sysfs, in particular, within the SCSI branch. Only manually configured LUNs are also represented within the zfcp branch.

You can find the FCP device bus-ID, the target WWPN, and the FCP LUN triplet that corresponds to a SCSI device in two ways: By traversing the sysfs directory tree or by using commands.

**Note:** The zfcp-specific sysfs attributes hba_id, wwpn, and fcp_lun are deprecated. Use the methods described here instead to find the addressing of a SCSI device.

About this task

The directory in the sysfs SCSI branch has the following form:

```
/sys/bus/scsi/devices/<scsi_host_no>:0:<scsi_id>:<scsi_lun>
```

where:

- `<scsi_host_no>` is the SCSI host number that corresponds to the FCP device.
- `<scsi_id>` is the SCSI ID of the target port.
- `<scsi_lun>` is the LUN of the SCSI device.

The value for `<scsi_lun>` depends on the storage device. Often, it is a single-digit number, but for some storage devices it has numerous digits.

For manually configured FCP LUNs, see “Manually configured FCP LUNs and their SCSI devices” on page 201 for details about the directory in the zfcp branch.

Before you begin

You must identify the SCSI device in sysfs. For example, use `readlink` to find the path in sysfs with all symbolic links resolved:

```bash
# readlink -e /sys/bus/scsi/devices/2:0:1:1074741413
/sys/devices/css0/0.0.00a/0.0.1800/host2/rport-2:0-1/target2:0:1/2:0:1:1074741413
```

Using sysfs

**Note:** Do not assume a stable sysfs structure. The following procedure accommodates changes in sysfs.
This example shows how you can traverse the directory tree to find the FCP device bus-ID, the target WWPN, and the FCP LUN that correspond to a SCSI device name. The example assumes:

**SCSI device**  
2:0:1:1074741413

**FCP LUN**  
0x40a5400f00000000

**target WWPN**  
0x50050763030bd327

**FCP device bus-ID**  
0.0.1800

1. **Obtain the hexadecimal FCP LUN.**
   a. Start at the SCSI device directory or anywhere in the subtree below the SCSI device. Ascend the sysfs tree until you find the SCSI device. To do this, test every subdirectory for a symbolic link named "subsystem" that points to a relative directory path whose last entry is `scsi`. Search for the symbolic link named "subsystem":

   ```bash
   # ls -dl subsystem
   lrwxrwxrwx 1 root root 0 Oct 19 16:08 subsystem -> ../../../../../../bus/scsi
   ```

   The subsystem symbolic link points to a directory tree where the last subdirectory is `scsi`.
   b. Confirm that this is a SCSI device by reading the `DEVTYPE` line within the `uevent` attribute. The value must be "scsi_device".

   ```bash
   # grep "^DEVTYPE=" uevent
   DEVTYPE=scsi_device
   ```

   The last part of the current directory name is then the decimal SCSI LUN, for example, assuming you have found this directory:

   ```bash
   # pwd
   /sys/devices/css0/0.0.000a/0.0.1800/host2/rport-2:0-1/target2:0:1/2:0:1:1074741413
   ```

   Here, the SCSI LUN is 1074741413.
   c. **Transform the SCSI LUN to the FCP LUN as follows:**

<table>
<thead>
<tr>
<th>Step</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take decimal LUN in decimal notation:</td>
<td>1074741413</td>
</tr>
<tr>
<td>Convert to hexadecimal notation:</td>
<td>0x400f40a5</td>
</tr>
<tr>
<td>Pad with 0 from the left to obtain a 64-bit value:</td>
<td>0x00000000400f40a5</td>
</tr>
<tr>
<td>Divide into 16-bit blocks (LUN levels):</td>
<td>0x0000</td>
</tr>
<tr>
<td>Reverse the order of the blocks:</td>
<td>0x40a5</td>
</tr>
<tr>
<td>The resulting hexadecimal number is the FCP LUN:</td>
<td>0x40a5400f00000000</td>
</tr>
</tbody>
</table>

   The Linux kernel function `int_to_scsilun()` in `drivers/scsi/scsi_common.c` converts a decimal SCSI LUN to obtain the hexadecimal FCP LUN according to this algorithm. The conversion works in both directions.

   d. Confirm that the path includes a directory "rport-<no>". For example, assuming you have found this directory:

   ```bash
   # pwd
   /sys/devices/css0/0.0.000a/0.0.1800/host2/rport-2:0-1/target2:0:1/2:0:1:1074741413
   ```
If there is no rport directory, the transport is not fibre channel and thus not zfcp-related. Abandon the search.

Table 28 on page 205 lists the libudev functions that you can use instead of manually traversing the sysfs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>udev_device_get_parent()</td>
<td>Ascend the sysfs tree.</td>
</tr>
<tr>
<td>udev_device_getSubsystem()</td>
<td>Retrieve subsystem name.</td>
</tr>
<tr>
<td>udev_device_get_devtype()</td>
<td>Retrieve device type.</td>
</tr>
<tr>
<td>udev_device_get_syspath()</td>
<td>Check if rport is a subdirectory.</td>
</tr>
</tbody>
</table>

Table 28: Useful udev functions

2. Obtain the target WWPN.

a. Continue ascending the sysfs tree the same way until you find the SCSI target. To do this, test every subdirectory for a symbolic link named "subsystem" that points to a relative directory path whose last entry is scsi. Search for the symbolic link named "subsystem":

```bash
# ls -dl subsystem
lrwxrwxrwx 1 root root 0 Oct 19 16:08 subsystem -> ../../../bus/scsi
```

b. Confirm that this is a SCSI target by reading the DEVTYPE line within the uevent attribute. The value must be "scsi_target".

```bash
# grep "^DEVTYPE=.*scsi_target" uevent
```

For example, assuming you have found this directory:

```bash
# pwd
/sys/devices/css0/0.0.000a/0.0.1800/host2/rport-2:0-1/target2:0:1
```

c. The SCSI target has a subdirectory fc_transport. Descend this subtree until you find a subdirectory that matches the SCSI target name. In this example, you would descend to fc_transport/target2:0:1.

d. In the found target, read the port_name attribute:

```bash
# cat port_name
0x50050763030bd327
```

The value of the port_name is the target WWPN.

Table 29: Useful udev functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>udev_device_get_parent_with_subsystem_devtype(dev, &quot;scsi&quot;, &quot;scsi_target&quot;)</td>
<td>Find the SCSI target.</td>
</tr>
<tr>
<td>udev_device_new_from_subsystem_sysname (udev_device_get_udev(scsidev), &quot;fc_transport&quot;, udev_device_get_sysname(targetdev))</td>
<td>Find a matching target in the fc_transport branch.</td>
</tr>
<tr>
<td>udev_device_get_sysattr_value()</td>
<td>Read the port_name attribute.</td>
</tr>
</tbody>
</table>
3. Obtain the FCP device-bus ID. Keep ascending the sysfs tree. Search for the symbolic link "subsystem" that points to a relative path where the last subdirectory is ccw.

For example:

```
# ls -dl subsystem
lrwxrwxrwx 1 root root 0 Oct 19 16:08 subsystem -> ../../../../bus/ccw
```

Then the name of the last directory in the current path is the FCP device-bus ID, for example:

```
# pwd
/sys/devices/css0/0.0.000a/0.0.1800
```

Here, 0.0.1800 is the FCP device-bus ID.

**Using commands**

To map a SCSI device name to its corresponding FCP device bus-ID, target WWPN, and LUN, you can use one of the following commands. The example assumes:

**SCSI device**

2:0:1:1074741413

**FCP LUN**

0x40a5400f00000000

**target WWPN**

0x50050763030bd327

**FCP device bus-ID**

0.0.1800

- Use the `lszfcp` with the -D option to list the FCP device-bus ID, the target WWPN, and the FCP LUN for all SCSI devices. For example:

```
lszfcp -D
....
0.0.1800/0x50050763030bd327/0x40a5400f0000000 2:0:1:1074741413
....
```

For details about the `lszfcp` command, see "lszfcp - List zfcp devices" on page 687.

- Use the `lszdev` command on device type zfcp-lun devices, and display the ID and `ATTR:scsi_dev` columns. For example:

```
# lszdev zfcp-lun -a -c ID,ATTR:scsi_dev
ID                                             ATTR:scsi_dev
...                                             ...
0.0.1800:0x50050763030bd327:0x40a5400f0000000 2:0:1:1074741413
...
```

For details about the `lszdev` command, see "lszdev - Display IBM Z device configurations" on page 682.

- Use the `lsscsi` command with the --transport and --lunhex options in verbose mode to get information about a SCSI device:

```
# lsscsi -xttv
[2:0:1:0x40a5400f00000000] disk     fc:0x50050763030bd327,0x249900   /dev/sda
dir: /sys/bus/scsi/devices/2:0:1:1074741413 [/sys/devices/css0/0.0.000a/0.0.1800/host2
/rport-2:0-1/target2:0:1/2:0:1:1074741413]
...
```

For details about the `lsscsi` command, see the man page.

**Note:** The details of the command output is subject to change. Do not rely on the output always being exactly as shown.
Figure 66 on page 207 illustrates the sysfs structure of a SCSI device and how it corresponds to the `lszfcp` command output.

**Warning:** Do not rely on the sysfs structure in the example. The sysfs structure changes without notice.

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 30 on page 208 lists read-only attributes for manually configured FCP LUNs. The table includes those attributes that indicate whether the device access is restricted by access control software on the FCP channel. These attributes can be found in the zfcp branch of sysfs. The path has the form:

```
/sys/bus/ccw/drivers/zfcp/<device_bus_id>/wwpn/<fcp_lun>/attribute
```
Table 30: Attributes of manually configured FCP LUNs with device access information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>access_denied</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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 182). You might also use different FCP channels or target ports.</td>
</tr>
<tr>
<td>access_shared</td>
<td>This attribute is obsolete. The value is always 0.</td>
</tr>
<tr>
<td>access_readonly</td>
<td>This attribute is obsolete. The value is always 0.</td>
</tr>
<tr>
<td>in_recovery</td>
<td>Shows if unit is in recovery (0 or 1)</td>
</tr>
</tbody>
</table>

Table 31 on page 208 lists a subset of all attributes with information about the SCSI device. These attributes can be found in the SCSI branch of sysfs. The path has the form:

```
/sys/class/scsi_device/<device_name>/device/<attribute>
```

Table 31: SCSI device class attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>device_blocked</td>
<td>Flag that indicates whether the device is in blocked state (1) or not (0).</td>
</tr>
<tr>
<td>iocounterbits</td>
<td>The number of bits used for I/O counters.</td>
</tr>
<tr>
<td>iodone_cnt</td>
<td>The number of completed or rejected SCSI commands.</td>
</tr>
<tr>
<td>ioperror_cnt</td>
<td>The number of SCSI commands that completed with an error.</td>
</tr>
<tr>
<td>iorequest_cnt</td>
<td>The number of issued SCSI commands.</td>
</tr>
<tr>
<td>model</td>
<td>The model of the SCSI device, received from inquiry data.</td>
</tr>
<tr>
<td>rev</td>
<td>The revision of the SCSI device, received from inquiry data.</td>
</tr>
<tr>
<td>scsi_level</td>
<td>The SCSI revision level, received from inquiry data.</td>
</tr>
<tr>
<td>type</td>
<td>The type of the SCSI device, received from inquiry data.</td>
</tr>
<tr>
<td>vendor</td>
<td>The vendor of the SCSI device, received from inquiry data.</td>
</tr>
<tr>
<td>zfcp_access_denied</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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 182). You might also use different FCP channels or target ports.</td>
</tr>
<tr>
<td>zfcp_in_recovery</td>
<td>Shows if unit is in recovery (0 or 1).</td>
</tr>
</tbody>
</table>

Procedure

Use the `lszfcp` command (see “`lszfcp - List zfcp devices” on page 687) to display information about the associated SCSI device.
Alternatively, you can use sysfs to read the information. To read attributes of the associated SCSI device, use a command of this form:

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

where:

- `<device_name>` is the name of the associated SCSI device.
- `<attribute>` is one of the attributes in Table 31 on page 208.

**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 674).

**Examples**

- In this example, information is displayed for a manually configured FCP LUN with LUN 0x4010403200000000 that is accessed through a target port with WWPN 0x500507630300c562 and is attached through an FCP device 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 197).

- You can use `lszfcp` to display attributes of a SCSI device. The example shows the attributes listed in Table 31 on page 208 as well as other relevant attributes:

  ```
  # lszfcp -l 0x4010403200000000 -a
  0.0.3d0c/0x500507630300c562/0x4010403200000000 0:0:0:0
  Class = "scsi_device"
  ...
  device_blocked = "0"
  ...
  iocounterbits = "32"
  iodone_cnt = "0xbe"
  iorequest_cnt = "0xbe"
  ...
  model = "2107900"
  queue_depth = "32"
  queue_ramp_up_period = "120000"
  ...
  rev = ".166"
  scsi_level = "6"
  state = "running"
  timeout = "30"
  type = "0"
  uevent = "DEVTYPE=scsi_device"
  vendor = "IBM"
  ...
  zfcp_access_denied = "0"
  zfcp_failed = "0"
  zfcp_in_recovery = "0"
  zfcp_status = "0x54000000"
  ```

**Finding the major and minor numbers for a block device that corresponds to a SCSI disk**

Linux creates block devices that represent SCSI disks. You can find the major and minor numbers of a SCSI disk block device and its partitions from the device representation in the sysfs SCSI branch.

**About this task**
In Figure 67 on page 210, `<scsi_device>` represents a SCSI device with a block device as a child. The block device has a name of the form `sd<x>`, which is a standard name that the SCSI stack assigned to the SCSI disk block device. If the disk is partitioned, the block directory that follows contains directories of the form `sd<x><n>` that represent the partitions, where `<n>` is a positive integer that identifies the partition.

Both the block device 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 disk 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:

```
# cat /sys/bus/scsi/devices/<scsi_disk>/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 block device that corresponds to SCSI disk 0:0:1:1:

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

Assuming that the disk 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 183) 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:
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 module parameter `queue_depth` or its associated kernel parameter described in “zfcp device driver kernel parameters” on page 183.
- 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 `zfcp_in_recovery` attribute to check whether recovery is under way.

**Before you begin**
The FCP device must be online.

**Procedure**
Perform the following steps to check the recovery status of a failed SCSI device:

1. Check the value of the `zfcp_in_recovery` attribute. Issue the `lszfcp` command:

   ```shell
   # lszfcp -l <LUN> -a
   ```
   
   where `<LUN>` is the LUN of the associated SCSI device.

   Alternatively, you can issue a command of this form:

   ```shell
   # cat /sys/class/scsi_device/<device_name>/device/zfcp_in_recovery
   ```
   
   The value is 1 if recovery is under way and 0 otherwise. If the value is 0 for a non-operational SCSI device, recovery might have failed. Alternatively, the device driver might have failed to detect that the SCSI device is malfunctioning.

2. To find out whether recovery failed, read the `zfcp_failed` attribute. Either use the `lszfcp` command again, or issue a command of this form:
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 zfcp_failed attribute. Issue a command of this form:

```
# echo 0 > /sys/class/scsi_device/<device_name>/device/zfcp_failed
```

**Example**

In the following example, SCSI device 0:0:0:0 is malfunctioning. The first command reveals that recovery is not already under way. The second command manually starts recovery for the SCSI device:

```
# cat /sys/class/scsi_device/0:0:0:0/device/zfcp_in_recovery
0
# echo 0 > /sys/class/scsi_device/0:0:0:0/device/zfcp_failed
```

**What to do next**

If you manually configured an FCP LUN (see “Manually configured FCP LUNs and their SCSI devices” on page 201), but did not get a corresponding SCSI device, you can also use the corresponding FCP LUN sysfs attributes, in_recovery and failed, to check on recovery. See Table 30 on page 208.

**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 203.

**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 203.

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 203. 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**

Issue a command of this form:

```bash
# 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:

```bash
# 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 203 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 scanning and configure all LUNs manually, see “Manually configured FCP LUNs and their SCSI devices” on page 201.

**Procedure**

Issue a command of this form:

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

**Example**

In this example, a 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.

```bash
# 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**

Manually remove a SCSI device if your environment is not set up to use NPIV or if you disabled automatic LUN scan. For details about disabling automatic LUN scan, see “zfcp module parameters” on page 186.
Before you begin

Attention: The following procedure ignores distribution specifics and might interfere with configuration tools provided by your distribution. Always use distribution-specific configuration tools if available.

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:

• For a persistent configuration, use the chzdev command. Issue a command of this form:

```
chzdev -d zfcp-lun <device_bus_id>:<wwpn>:<fcp_lun>
```

• For the current configuration only, use the chzdev command with the -a option or sysfs.

  – To use chzdev, issue a command of this form:

```
chzdev -d -a zfcp-lun <device_bus_id>:<wwpn>:<fcp_lun>
```

  – To use sysfs, follow this procedure:

    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 203 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 201. 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.

• To remove the LUN and make the configuration change persistent, use:

```
# chzdev -d zfcp-lun 0.0.3d0c:0x500507630300c562:0x4010403200000000
```

• For a non-persistent configuration, use the chzdev command with the -a option:

```
# chzdev -d -a zfcp-lun 0.0.3d0c:0x500507630300c562:0x4010403200000000
```

Alternatively, for a non-persistent configuration, you can also use sysfs:

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:
Confirming end-to-end data consistency checking

There are different types of end-to-end data consistency checking, with dependencies on hardware and software.

**About this task**

End-to-end data consistency checking is based on a data integrity field (DIF) that is added to transferred data blocks. DIF data is used to confirm that a data block originates from the expected source and was not modified during the transfer between the storage system and the FCP device. The SCSI Block Commands (T10 SBC) standard defines several types of DIF. Linux data integrity extension (DIX) builds on DIF to extend consistency checking, for example, to the operating system, middleware, or an application.

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 183). With end-to-end data consistency checking for SCSI disks enabled, Linux automatically discovers which FCP devices and which SCSI disks support end-to-end data consistency checking. No further setup is required.

**Note:** SCSI disks 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 disks for which end-to-end data consistency checking is used, there is a sysfs directory:

```
/sys/block/sd<x>/integrity
```

In the path, sd<x> is the standard name of the block device of the SCSI disk.

End-to-end data consistency checking is used only if all of the following components support it:

**SCSI disk**
Check your storage server documentation about T10 DIF support and any restrictions.

**IBM Z hardware**
IBM Z FCP adapter hardware supports end-to-end data consistency checking as of FICON Express8.

**Hypervisor**
For Linux on z/VM, you require a z/VM version with guest support for end-to-end data consistency checking.

**FCP device**
Check your FCP adapter hardware documentation about the support and any restrictions. For example, end-to-end data consistency checking might be supported only for disks with 512-byte block size.

Read the prot_capabilities sysfs attribute of the SCSI host that is associated with an FCP device to find out about its end-to-end data consistency checking support. The following values are possible:

- **0** The FCP device does not support end-to-end data consistency checking.
- **1** The FCP device supports DIF type 1.
The FCP device supports DIX type 1.

The FCP device supports DIX type 1 with DIF type 1.

**Procedure**

Issue a command of this form:

```bash
# 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**

```bash
# cat /sys/bus/ccw/devices/0.0.1940/host0/scsi_host/host0/protCapabilities
```

**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:

   ```bash
   # 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:

   ```bash
   # 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:

   ```bash
   # 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:

   ```bash
   # lszfcp -P
   0.0.3c02/0x50050763030bc562 rport-0:0-0
   0.0.3c02/0x500507630310c562 rport-0:0-1
   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 HBA management clients that run on Linux on Z.

Table 32 on page 218 gives an overview of available packages of zfcp HBA API and the binary files that are installed with those versions.

<table>
<thead>
<tr>
<th>Package</th>
<th>Installed binary files</th>
</tr>
</thead>
<tbody>
<tr>
<td>zfcp HBA API library 2.1</td>
<td>libzfcphbaapi.so</td>
</tr>
<tr>
<td>zfcp HBA API library 2.0</td>
<td>libzfcphbaapi.so</td>
</tr>
<tr>
<td>zfcp HBA API library 1.4</td>
<td>san_disc command</td>
</tr>
<tr>
<td>zfcp HBA API library 1.3</td>
<td>libzfcphbaapi.so, san_disc command</td>
</tr>
<tr>
<td>zfcp HBA API library 1.2 and earlier</td>
<td>libzfcphbaapi.so</td>
</tr>
</tbody>
</table>

**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](http://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 4.19 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 z Systems*, 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 14. Storage-class memory device driver

Storage-class memory (SCM) is a class of data storage devices that combines properties of both storage and memory.

SCM can be implemented as Flash Express or as Virtual Flash Memory.

What you should know about storage-class memory

Storage-class memory (SCM) is accessed, in chunks called increments, through extended asynchronous data mover (EADM) subchannels.

The LPAR on which your Linux instance runs must be configured to provide SCM.

- At least one EADM subchannel must be available to the LPAR. Because SCM supports multiple concurrent I/O requests, it is advantageous to configure multiple EADM subchannels. A typical number of EADM subchannels is 64.
- One or more SCM increments must be added to the I/O configuration of the LPAR.

In Linux, each increment is represented as a block device. You can use the block device with standard Linux tools as you would use any other block device. Commonly used tools that work with block devices include: fdisk, mkfs, and mount.

Storage-class memory 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, 1sblk 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
...```

© Copyright IBM Corp. 2000, 2018
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 68 on page 222 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)
```

**Figure 68: 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.

### Setting up the storage-class memory device driver

Configure the storage-class memory device driver by using the module parameters.

```
modprobe scm_block
```

**Figure 69: Storage-class memory module parameter syntax**

- **nr_requests** specifies the number of parallel I/O requests. Set this number to the number of EADM subchannels. The default is 64.
- **nr_request_per_io** submits more concurrent I/O requests than the current limit, which is based on the number of available EADM subchannels (64). Valid values are 1 - 64. Increasing the requests increases the number of I/O requests per second, especially for requests with a small block size. The default
Working with storage-class memory increments

You can list storage-class memory increments and EADM subchannels.

- “Displaying EADM subchannels” on page 223
- “Listing storage-class memory increments” on page 223
- “Combining SCM devices with LVM” on page 224

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:

```bash
# lscss --eadm
```

<table>
<thead>
<tr>
<th>Device</th>
<th>Subchan.</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>0.0.ff00</td>
</tr>
<tr>
<td>n/a</td>
<td>0.0.ff01</td>
</tr>
<tr>
<td>n/a</td>
<td>0.0.ff02</td>
</tr>
<tr>
<td>n/a</td>
<td>0.0.ff03</td>
</tr>
<tr>
<td>n/a</td>
<td>0.0.ff04</td>
</tr>
<tr>
<td>n/a</td>
<td>0.0.ff05</td>
</tr>
<tr>
<td>n/a</td>
<td>0.0.ff06</td>
</tr>
<tr>
<td>n/a</td>
<td>0.0.ff07</td>
</tr>
</tbody>
</table>

For more information about the `lscss` command, see “lscss - List subchannels” on page 659.

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:

```bash
# lsscm
```

<table>
<thead>
<tr>
<th>SCM Increment</th>
<th>Size</th>
<th>Name</th>
<th>Rank</th>
<th>D_state</th>
<th>O_state</th>
<th>Pers</th>
<th>ResID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000000000</td>
<td>16384MB</td>
<td>scma</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0000000000000000</td>
<td>16384MB</td>
<td>scmb</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

See “lsscm - List storage-class memory increments” on page 671 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 15. Channel-attached tape device driver

The Linux on 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 13, “SCSI-over-Fibre Channel device driver,” on page 177).

Features

The tape device driver supports a range of channel-attached tape devices and functions of these devices.

• The tape device driver supports channel-attached tape drives that are compatible with IBM 3480, 3490, 3590, and 3592 magnetic tape subsystems. Various models of these device types are handled (for example, the 3490/10).
  3592 devices that emulate 3590 devices are recognized and treated as 3590 devices.
• Logical character devices for non-rewinding and rewinding modes of operation (see “Tape device modes and logical devices” on page 225)
• Control operations through mt (see “Using the mt command” on page 228)
  
• Message display support (see “tape390_display - Display messages on tape devices and load tapes” on page 731)
• Encryption support (see “tape390_crypt - Manage tape encryption” on page 727)
• 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 33 on page 225.

<table>
<thead>
<tr>
<th>Device</th>
<th>Names</th>
<th>Minor numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-rewinding character devices</td>
<td>ntimb&lt;n&gt;</td>
<td>2×&lt;n&gt;</td>
</tr>
<tr>
<td>Rewinding character devices</td>
<td>rtimb&lt;n&gt;</td>
<td>2×&lt;n&gt;+1</td>
</tr>
</tbody>
</table>
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 674) 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.

```bash
# 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 34 on page 226 summarizes the resulting names and minor numbers.

<table>
<thead>
<tr>
<th>Bus ID</th>
<th>Index (TapeNo)</th>
<th>Device</th>
<th>Device name</th>
<th>Minor number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.01a1</td>
<td>0</td>
<td>non-rewind</td>
<td>ntibm0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rewind</td>
<td>rtibm0</td>
<td>1</td>
</tr>
<tr>
<td>0.0.01a0</td>
<td>1</td>
<td>non-rewind</td>
<td>ntibm1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rewind</td>
<td>rtibm1</td>
<td>3</td>
</tr>
<tr>
<td>0.0.0172</td>
<td>2</td>
<td>non-rewind</td>
<td>ntibm2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rewind</td>
<td>rtibm2</td>
<td>5</td>
</tr>
<tr>
<td>0.0.01ac</td>
<td>not assigned</td>
<td>n/a</td>
<td>n/a</td>
<td>not assigned</td>
</tr>
</tbody>
</table>

For the online devices, the major/minor assignments can be read from their respective representations in `/sys/class`:
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 225).

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 33 on page 225.

```
# 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 7) 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 234 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 70 on page 229 summarizes the kernel configuration menu options that are relevant to the tape device driver:

<table>
<thead>
<tr>
<th>Device Drivers ---&gt;</th>
<th>Character devices ---&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/390 tape device support</td>
<td>(CONFIG_S390_TAPE)</td>
</tr>
<tr>
<td>--- S/390 tape hardware support (depends on S390_TAPE) ---</td>
<td></td>
</tr>
<tr>
<td>Support for 3480/3490 tape hardware</td>
<td>(CONFIG_S390_TAPE_34XX)</td>
</tr>
<tr>
<td>Support for 3590 tape hardware</td>
<td>(CONFIG_S390_TAPE_3590)</td>
</tr>
</tbody>
</table>

Figure 70: 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
modprobe 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 56, “Channel measurement facility,” on page 539.

For information about displaying messages on a tape device's display unit, see “tape390_display - Display messages on tape devices and load tapes” on page 731.

See “Working with newly available devices” on page 10 to avoid errors when working with devices that have become available to a running Linux instance.

- “Setting a tape device online or offline” on page 230
- “Displaying tape information” on page 231
- “Enabling compression” on page 233
- “Loading and unloading tapes” on page 234

Setting a tape device online or offline

Set a tape device online or offline with the chccwdev command or through the online sysfs attribute of the device.

About this task

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

- The non-rewind character device
- The rewind character device

At any time, the device can be online to a single Linux instance only. You must set the tape device offline to make it accessible to other Linux instances in a shared environment.

Procedure

Use the chzdev command (see “chzdev - Configure IBM Z devices” on page 589) to set a tape online or offline.

Alternatively, use the chccwdev command, or 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 227) 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 227).

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 674.

**Examples**

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

  ```
  # chzdev -e tape 015f
  ```

  This setting persists across re-boots. For more details, see Chapter 29, “Persistent device configuration,” on page 377.

  To apply this setting to the running system only, use the `chzdev` command with the `-a` option or one of the following commands:

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

  In the example, the assigned index number is 2. The standard device nodes for working with the device until it is set offline are then:

  - `/dev/ntibm2` for the non-rewinding device
  - `/dev/rtibm2` for the rewinding device

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

  ```
  # chzdev -d tape 015f
  ```

  This setting persists across re-boots. For more details, see Chapter 29, “Persistent device configuration,” on page 377.

  To apply this setting to the running system only, use the `chzdev` command with the `-a` option or one of the following commands:

  ```
  # chccwdev -d 0.0.015f
  # 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 35: Tape device attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>online</td>
<td>1 if the device is online or 0 if it is offline (see “Setting a tape device online or offline” on page 230)</td>
</tr>
<tr>
<td>cmb_enable</td>
<td>1 if channel measurement block is enabled for the physical device or 0 if it is not enabled (see Chapter 56, “Channel measurement facility,” on page 539)</td>
</tr>
<tr>
<td>cutype</td>
<td>Type and model of the control unit</td>
</tr>
<tr>
<td>devtype</td>
<td>Type and model of the physical tape device</td>
</tr>
<tr>
<td>blocksize</td>
<td>Currently used record size in bytes or 0 for auto</td>
</tr>
<tr>
<td>state</td>
<td>State of the physical tape device, either of:</td>
</tr>
</tbody>
</table>
|               | **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:                                   |
| medium_state  | 1                                                                          
|               | Cartridge is loaded into the tape device                                   |
| medium_state  | 2                                                                          
|               | No cartridge is loaded                                                    |
| medium_state  | 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:

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

where `<attribute>` is one of the attributes of Table 35 on page 232.
Example
The following listape command displays information about a tape device with bus ID 0.0.015f:

```
# listape 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 731) 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 731). 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 16. XPRAM device driver

With the XPRAM block device driver Linux can access expanded storage. XPRAM can be used as a basis for fast swap devices or 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>
<thead>
<tr>
<th>Minor</th>
<th>Name</th>
<th>To access</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>slram0</td>
<td>the first partition or the entire expanded storage if there are no partitions</td>
</tr>
<tr>
<td>1</td>
<td>slram1</td>
<td>the second partition</td>
</tr>
<tr>
<td>2</td>
<td>slram2</td>
<td>the third partition</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>&lt;n&gt;</td>
<td>slram&lt;n&gt;</td>
<td>the (&lt;n&gt;+1)th partition</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>31</td>
<td>slram31</td>
<td>the 32nd partition</td>
</tr>
</tbody>
</table>
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 235).

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 36 on page 235.

```
# 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 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 71 on page 237 shows where to find CONFIG_BLK_DEV_XPRAM in the configuration menu.
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 236 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.parts = <number_of_partitions>, <partition_size>
```

**Figure 72: 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.

  ```text
  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.

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

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

  ```text
  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.

![Diagram of module parameters](image)

*Figure 73: 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.
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.

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

See the `modprobe` man page for details about `modprobe`. 
Part 4. Networking

Linux on Z includes several network device drivers that are specific to z/Architecture.

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

Example

Figure 74: 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 mainframe to an external network.
Table 37 on page 242 lists which control units and device type combinations are supported by the network device drivers.

<table>
<thead>
<tr>
<th>Device type</th>
<th>Control unit</th>
<th>Device driver</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1732/01</td>
<td>1731/01</td>
<td>qeth</td>
<td>OSA configured as OSD</td>
</tr>
<tr>
<td>1732/02</td>
<td>1731/02</td>
<td>qeth</td>
<td>OSA configured as OSX</td>
</tr>
<tr>
<td>1732/03</td>
<td>1731/02</td>
<td>qeth</td>
<td>OSA configured as OSM</td>
</tr>
<tr>
<td>1732/05</td>
<td>1731/05</td>
<td>qeth</td>
<td>HiperSockets</td>
</tr>
<tr>
<td>1732/06</td>
<td>1731/06</td>
<td>qeth</td>
<td>OSA configured as OSN</td>
</tr>
<tr>
<td>0000/00</td>
<td>3088/08</td>
<td>ctm</td>
<td>Virtual CTC under z/VM</td>
</tr>
<tr>
<td>0000/00</td>
<td>3088/1e</td>
<td>ctm</td>
<td>FICON channel</td>
</tr>
<tr>
<td>0000/00</td>
<td>3088/1f</td>
<td>lcs</td>
<td>2216 Nways Multiaccess Connector</td>
</tr>
<tr>
<td>0000/00</td>
<td>3088/1f</td>
<td>ctm</td>
<td>ESCON channel</td>
</tr>
<tr>
<td>0000/00</td>
<td>3088/60</td>
<td>lcs</td>
<td>OSA configured as OSE (non-QDIO)</td>
</tr>
</tbody>
</table>
The qeth device driver supports a multitude of network connections, for example, connections through Open Systems Adapters (OSA), HiperSockets, guest LANs, and virtual switches.

**Real connections that use OSA-Express**

An IBM mainframe uses OSA-Express adapters, which are real LAN-adapter hardware, see Figure 75 on page 243. 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, OSM, OSN, or OSX in the hardware configuration). OSD-devices are the standard IBM Z LAN-adapters, while OSN-devices serve as NCP-adapters. For details about OSA-Express in QDIO mode, see Open Systems Adapter-Express Customer's Guide and Reference, SA22-7935.

Figure 75: 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**

An IBM mainframe uses internal connections that are called HiperSockets. These simulate QDIO network adapters and provide high-speed TCP/IP communication for operating system instances within and across LPARs. For details about HiperSockets, see *HiperSockets Implementation Guide*, SG24-6816.

**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 76 on page 244.

![Figure 76: Guest LAN](image)

**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 77 on page 245.
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 78 on page 245. You can thus connect a HiperSockets LAN to an external LAN without using a router.

For more information about bridge ports, see “Layer 2 promiscuous mode” on page 256, and Figure 82 on page 257.

For information about guest LANs, virtual switches, HiperSockets bridge ports and virtual HiperSockets, see z/VM: Connectivity, SC24-6267.

The qeth network device driver supports the IBM Z OSA-Express6S, OSA-Express5S, OSA-Express4S, OSA-Express3, OSA-Express2, and OSA-Express features and HiperSockets as shown in Table 38 on page 246 and Table 39 on page 246:
Table 38: The qeth device driver support for HiperSockets and OSA-Express features - Part 1: Mainframe CPCs as of zEnterprise

<table>
<thead>
<tr>
<th>Feature</th>
<th>z14 and z14 ZR1</th>
<th>z13 and z13s</th>
<th>zEC12 and zBC12</th>
<th>z196 and z114</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiperSockets</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>OSA-Express6S</td>
<td>Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
<tr>
<td>OSA-Express5S</td>
<td>Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet</td>
<td>Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet</td>
<td>Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet</td>
<td>Not supported</td>
</tr>
<tr>
<td>OSA-Express4S</td>
<td>1000Base-T Ethernet</td>
<td>Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet</td>
<td>Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet</td>
<td>Gigabit Ethernet 10 Gigabit Ethernet</td>
</tr>
<tr>
<td>OSA-Express3</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet</td>
<td>Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet</td>
</tr>
<tr>
<td>OSA-Express2</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Gigabit Ethernet 1000Base-T Ethernet</td>
</tr>
<tr>
<td>OSA-Express</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

Table 39: The qeth device driver support for HiperSockets and OSA-Express features - Part 2: Earlier mainframe CPCs

<table>
<thead>
<tr>
<th>Feature</th>
<th>System z10</th>
<th>System z9</th>
<th>eServer™ zSeries</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiperSockets</td>
<td>Yes</td>
<td>Yes (layer 3 only)</td>
<td>Yes (layer 3 only)</td>
</tr>
<tr>
<td>OSA-Express3</td>
<td>Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
<tr>
<td>OSA-Express2</td>
<td>Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet</td>
<td>Gigabit Ethernet 10 Gigabit Ethernet 1000Base-T Ethernet</td>
<td>Not supported</td>
</tr>
</tbody>
</table>
Table 39: The qeth device driver support for HiperSockets and OSA-Express features - Part 2: Earlier mainframe CPCs (continued)

<table>
<thead>
<tr>
<th>Feature</th>
<th>System z10</th>
<th>System z9</th>
<th>eServer™ zSeries</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSA-Express</td>
<td>Not supported</td>
<td>Fast Ethernet</td>
<td>Fast Ethernet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gigabit Ethernet</td>
<td>Gigabit Ethernet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000Base-T Ethernet</td>
<td>1000Base-T Ethernet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ATM (z900, needs RPQ)</td>
</tr>
</tbody>
</table>

**Note:** Unless otherwise indicated, OSA-Express refers to OSA-Express6S, OSA-Express5S, 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 40 on page 247 and Table 41 on page 249.

**Table 40: Real connections.**

<table>
<thead>
<tr>
<th>Function</th>
<th>OSA Layer 2</th>
<th>OSA Layer 3</th>
<th>HiperSockets Layer 2</th>
<th>HiperSockets Layer 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic device or protocol functions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPv4/multicast/broadcast</td>
<td>Yes/Yes/Yes</td>
<td>Yes/Yes/Yes</td>
<td>Yes/Yes/Yes</td>
<td>Yes/Yes/Yes</td>
</tr>
<tr>
<td>IPv6/multicast</td>
<td>Yes/Yes</td>
<td>Yes/Yes</td>
<td>Yes/Yes</td>
<td>Yes/Yes</td>
</tr>
<tr>
<td>Non-IP traffic</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>VLAN IPv4/IPv6/non IP</td>
<td>sw/sw/sw</td>
<td>hw/sw/sw</td>
<td>sw/sw/sw</td>
<td>hw/hw/No</td>
</tr>
<tr>
<td>Linux ARP</td>
<td>Yes</td>
<td>No (hw ARP)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Linux neighbor solicitation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Unique MAC address</td>
<td>Yes (random for LPAR)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Change MAC address</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Promiscuous mode</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>• Yes (for sniffer=1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• No (for sniffer=0)</td>
</tr>
<tr>
<td>MAC headers send/receive</td>
<td>Yes/Yes</td>
<td>faked/faked</td>
<td>Yes/Yes</td>
<td>faked/faked</td>
</tr>
<tr>
<td>ethtool support</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bonding</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Priority queueing</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bridge port</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
### Table 40: Real connections. (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>OSA Layer 2</th>
<th>OSA Layer 3</th>
<th>HiperSockets Layer 2</th>
<th>HiperSockets Layer 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic device or protocol functions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Offload features</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP segmentation offload (TSO)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Inbound (rx) checksum</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Outbound (tx) checksum</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>OSA/QETH specific features</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Special device driver setup for VIPA</td>
<td>No</td>
<td>required</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Special device driver setup for proxy ARP</td>
<td>No</td>
<td>required</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Special device driver setup for IP takeover</td>
<td>No</td>
<td>required</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Special device driver setup for routing IPv4/IPv6</td>
<td>No/No</td>
<td>required/required</td>
<td>No/No</td>
<td>Yes/Yes</td>
</tr>
<tr>
<td>Receive buffer count</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Direct connectivity to z/OS</td>
<td>Yes by HW</td>
<td>Yes</td>
<td>no</td>
<td>Yes</td>
</tr>
<tr>
<td>SNMP support</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multiport support</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Data connection isolation</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Problem determination</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware trace</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Function</th>
<th>Emulated OSA Layer 2</th>
<th>Emulated OSA Layer 3</th>
<th>Emulated HiperSockets Layer 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic device or protocol features</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPv4/multicast/broadcast</td>
<td>Yes/Yes/Yes</td>
<td>Yes/Yes/Yes</td>
<td>Yes/Yes/Yes</td>
</tr>
<tr>
<td>IPv6/multicast</td>
<td>Yes/Yes</td>
<td>Yes/Yes</td>
<td>No/No</td>
</tr>
<tr>
<td>Non-IP traffic</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>VLAN IPv4/IPv6/non IP</td>
<td>sw/sw/sw</td>
<td>hw/sw/No</td>
<td>hw/No/No</td>
</tr>
<tr>
<td>Linux ARP</td>
<td>Yes</td>
<td>No (hw ARP)</td>
<td>No</td>
</tr>
<tr>
<td>Linux neighbor solicitation</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Unique MAC address</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Change MAC address</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Promiscuous mode</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MAC headers send/receive</td>
<td>Yes/Yes</td>
<td>faked/faked</td>
<td>faked/faked</td>
</tr>
<tr>
<td>ethtool support</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bonding</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Priority queueing</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Offload features</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>OSA/QETH specific features</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Special device driver setup for VIPA</td>
<td>No</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>Special device driver setup for proxy ARP</td>
<td>No</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>Special device driver setup for IP takeover</td>
<td>No</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>Special device driver setup for routing IPv4/IPv6</td>
<td>No/No</td>
<td>required/required</td>
<td>required/required</td>
</tr>
<tr>
<td>Receive buffer count</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Direct connectivity to z/OS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SNMP support</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multiport support</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Data connection isolation</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Problem determination</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware trace</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 41: Virtual NICs coupled to a z/VM VSWITCH or guest LAN. (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Emulated OSA Layer 2</th>
<th>Emulated OSA Layer 3</th>
<th>Emulated HiperSockets Layer 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic device or protocol features</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 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 253 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 254 for details.

The layer 2 discipline (qeth_l2)

The layer 2 discipline supports:

- OSA devices and z/VM virtual NICs that couple to VSWITCHes or QDIO guest LANs running in layer 2 mode
- 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 265 for details.

For z/VM NICs that are coupled to a guest LAN or VSWITCH, the qeth device driver detects the required layer and configures it automatically. If a qeth device is created before the NIC is coupled, the qeth device driver defaults to layer 2.
The layer 3 discipline (qeth_l3)
The layer 3 discipline supports:

- OSA devices and z/VM virtual NICs that couple to VSWITCHes or QDIO guest LANs 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 47 on page 262. 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 79 on page 251).

Figure 79: 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 260.

**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 other network device drivers. 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 320.

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 270 and “Finding out the bus ID of a qeth interface” on page 271 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.

![MAC header diagram](image)

*Figure 80: Standard IPv4 processing*

MAC address handling as shown in Figure 80 on page 253 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 80 on page 253. 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.

For Linux as a z/VM guest, the qeth device driver obtains a MAC address for each L2 device from the z/VM host. No configuration is necessary.

For Linux in LPAR mode with a directly attached OSA adapter in QDIO mode, assign a unique MAC address. You can set a MAC address by issuing the command:

```
ip link set addr <MAC address> dev <interface>
```

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

For OSX 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.
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 81 on page 254. 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 81: MAC address handling in layer 3 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 305.

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 42: Ethernet addresses of outgoing frames

<table>
<thead>
<tr>
<th>Frame</th>
<th>Destination address</th>
<th>Source address</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4</td>
<td>FAKELL</td>
<td>Real device address</td>
</tr>
<tr>
<td>IPv6</td>
<td>Real destination address</td>
<td>Real device address</td>
</tr>
<tr>
<td>Other packets</td>
<td>Real destination address</td>
<td>Real device address</td>
</tr>
</tbody>
</table>
Incoming frames
The device driver provides Ethernet headers for all incoming frames.

If necessary information like the Ethernet source address is not available (because of the offload functionality) the value is filled with the hardcoded address FAKELL.

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

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 44 on page 255.

<table>
<thead>
<tr>
<th>Application</th>
<th>Support</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcpdump</td>
<td>Yes</td>
<td>Displays only frames, fake Ethernet information is displayed.</td>
</tr>
<tr>
<td>iptables</td>
<td>Partially</td>
<td>As long as the rule does not deal with Ethernet information of an IPv4 frame.</td>
</tr>
<tr>
<td>dhcp</td>
<td>Yes</td>
<td>Is non-IPv4 traffic. (Note that DHCP does not work for Layer 3 HiperSockets.)</td>
</tr>
</tbody>
</table>

IP addresses
The network stack of each operating system that shares an OSA-Express adapter in QDIO mode registers all its IP addresses with the adapter.

Whenever IP addresses are deleted from or added to a network stack, the device drivers download the resulting IP address list changes to the OSA-Express adapter.

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

- Handling MAC addresses and MAC headers
- ARP processing

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

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

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

Layer 2 promiscuous mode
OSA and HiperSockets ports that operate in layer 2 mode can be set up to receive all frames that are addressed to unknown MAC addresses.

On most architectures, traffic between operating systems and networks is handled by Ethernet Network Interface Controllers (NICs). NICs usually filter incoming traffic to admit only frames with destination MAC addresses that are registered with the NIC.

However, a NIC can also be configured to receive and pass to the operating system all Ethernet frames that reach it, regardless of the destination MAC address. This mode of operation is known as "promiscuous mode". Promiscuous mode is a prerequisite for configuring a NIC as a member of a Linux software bridge.

For more information about how to set up a software bridge, see the documentation that is provided by your distributor, or the bridging how-to available at http://www.tldp.org/HOWTO/BRIDGE-STP-HOWTO.

On IBM Z, you can realize a promiscuous mode for Ethernet traffic through a bridge port configuration or through Virtual Network Interface Controller (VNIC) characteristics. OSA devices can be configured as bridge ports. HiperSockets Devices can be configured either as bridge ports or with VNIC characteristics, but not both simultaneously.

VNIC characteristics
With (VNIC) characteristics, you can set and fine-tune a promiscuous mode for HiperSockets devices, (see “Advanced packet-handling configuration” on page 291).

Bridge ports
Linux can assign a bridge port role to a logical port and the adapter assigns an active state to one of the logical ports to which a role was assigned. A local port in active bridge port state receives all Ethernet frames with unknown destination MAC addresses.

Figure 82 on page 257 shows a setup with a HiperSockets bridge port and an OSA bridge port.
HiperSockets only: Permission to configure ports as bridge ports must be granted in IBM zEnterprise Unified Resource Manager (zManager).

**Differences between promiscuous mode and bridge-port roles**

Making a logical port of an OSA or HiperSockets adapter an active bridge port is similar to enabling promiscuous mode on a non-mainframe NIC that is connected to a real Ethernet switch. However, there are important differences:

**Number of ports in promiscuous mode**

- Real switches: Any number of interfaces that are connected to a real switch can be turned to promiscuous mode, and all of them then receive frames with unknown destination addresses.
- Bridge ports on Z: Although you can assign the bridge-port role to multiple ports of a single OSA or HiperSockets adapter, only one port is active and receives traffic to unknown destinations.

**Interception of traffic to other systems**

- Real switches: A port of a real switch can be configured to receive frames with both known and unknown destinations. If a NIC in promiscuous mode is connected to the port, the corresponding host receives a copy of all traffic that passes through the switch. This includes traffic that is destined to other hosts connected to this switch.
- Bridge ports on Z: Only frames with unknown destinations are passed to the operating system. It is not possible to intercept traffic addressed to systems connected to other ports of the same OSA or HiperSockets adapter.

**Limitation by the source of traffic (OSA bridge port only)**

- Real switches and HiperSockets bridge-port LAN: Frames with unknown destination MAC addresses are delivered to the promiscuous interfaces regardless of the port through which the frames enter the switch or HiperSockets adapter.
- OSA bridge port only: An active bridge port learns which MAC addresses need to be routed to the owning system by analyzing ARP and other traffic. Incoming frames are routed to the active bridge port if their destination MAC address:
  - Matches an address that is learned or registered with the bridge port
– Is not learned or registered with any of the local ports of the OSA adapter, but arrived from the physical Ethernet port

**Bridge port roles**

Linux can assign a primary or secondary role to a logical port of an OSA or a HiperSockets adapter. Only one logical port of such an adapter can be assigned the primary role, but multiple other logical ports can be assigned secondary role. When one or more logical ports of an adapter are assigned primary or secondary role, the hardware ensures that exactly one of these ports is active. The active port receives frames with unknown destination. When a port with primary role is present, it always becomes active. When only ports with secondary role are present, the hardware decides which one becomes active. Changes in the ports’ state are reported to Linux user space through udev events.

You can set a bridge port role either directly by using the bridge_role attribute or indirectly by using the bridge_reflect_promisc attribute. See “Configuring a network device as a member of a Linux bridge” on page 289.

**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 83 on page 258 summarizes the kernel configuration menu options that are relevant to the qeth device driver:

| I/O subsystem ---| QDIO support (CONFIG_QDIO) |
| Device Drivers ---| (common code option CONFIG_NETDEVICES) |
| Network device support ---| S/390 network device drivers (depends on NETDEVICES & S390) ---| (CONFIG_QETH)* |
| Gigabit Ethernet device support (CONFIG_QETH_L2) |
| qeth layer 2 device support (CONFIG_QETH_L3) |

**Figure 83: qeth kernel configuration menu options**

**CONFIG_QDIO**

This option provides the Queued Direct I/O base support for IBM 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 3 mode. 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:

```
modprobe qeth
modprobe qeth_l2
modprobe qeth_l3
```

*Figure 84: qeth module syntax*

where:

- **qeth**
  is the core module that contains common functions that are used for both layer 2 and layer 3 disciplines.

- **qeth_l2**
  is the module that contains layer 2 discipline-specific code.

- **qeth_l3**
  is the module that contains layer 3 discipline-specific code.

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

Some devices can only run in one discipline, see “Layer 2 and layer 3” on page 250. The device driver rejects any request to switch the discipline of these devices.

If the new discipline is accepted by the device driver, the old network interface is deleted. When the new discipline is set online the first time, the new network interface is created.
Removing the modules

Removing a module is not possible if there are cross dependencies between the discipline modules and the core module.

To release the dependencies from the core module to the discipline module, all devices of this discipline must be ungrouped. Now the discipline module can be removed. If all discipline modules are removed, the core module can be removed.

Working with qeth devices

Typical tasks 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 45 on page 260, Table 46 on page 261, and Table 47 on page 262 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 752).

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>
<thead>
<tr>
<th>Task</th>
<th>Corresponding attributes</th>
<th>Possible attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Creating a qeth group device” on page 263</td>
<td>group</td>
<td>n/a</td>
</tr>
<tr>
<td>“Removing a qeth group device” on page 264</td>
<td>ungroup</td>
<td>0 or 1</td>
</tr>
<tr>
<td>“Setting the layer2 attribute” on page 265</td>
<td>layer2</td>
<td>0 or 1, see “Layer 2 and layer 3” on page 250¹</td>
</tr>
<tr>
<td>“Using priority queueing” on page 266</td>
<td>priority_queueing</td>
<td>prio_queueing_vlan</td>
</tr>
<tr>
<td></td>
<td></td>
<td>prio_queueing_skb</td>
</tr>
<tr>
<td></td>
<td></td>
<td>prio_queueing_prec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>prio_queueing_tos</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no_prio_queueing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no_prio_queueing:0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no_prio_queueing:1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no_prio_queueing:2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no_prio_queueing:3</td>
</tr>
</tbody>
</table>
Table 45: qeth tasks and attributes common to layer2 and layer3. (continued)

<table>
<thead>
<tr>
<th>Task</th>
<th>Corresponding attributes</th>
<th>Possible attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Specifying the number of inbound buffers” on page 267</td>
<td>buffer_count</td>
<td>integer in the range 8 - 128. The default is 64 for OSA devices and 128 for HiperSockets devices</td>
</tr>
<tr>
<td>“Finding out the maximum frame size” on page 268</td>
<td>inbuf_size</td>
<td>n/a, read-only</td>
</tr>
<tr>
<td>“Specifying the relative port number” on page 269</td>
<td>portno</td>
<td>integer, either 0 or 1, the default is 0</td>
</tr>
<tr>
<td>“Finding out the type of your network adapter” on page 269</td>
<td>card_type</td>
<td>n/a, read-only</td>
</tr>
<tr>
<td>“Setting a device online or offline” on page 270</td>
<td>online</td>
<td>0 or 1</td>
</tr>
<tr>
<td>“Finding out the interface name of a qeth group device” on page 270</td>
<td>if_name</td>
<td>n/a, read-only</td>
</tr>
<tr>
<td>“Finding out the bus ID of a qeth interface” on page 271</td>
<td>none</td>
<td>n/a</td>
</tr>
<tr>
<td>“Activating an interface” on page 271</td>
<td>none</td>
<td>n/a</td>
</tr>
<tr>
<td>“Deactivating an interface” on page 273</td>
<td>none</td>
<td>n/a</td>
</tr>
<tr>
<td>“Recovering a device” on page 273</td>
<td>recover</td>
<td>1</td>
</tr>
<tr>
<td>“Turning inbound checksum calculations on and off” on page 274</td>
<td>none</td>
<td>n/a</td>
</tr>
<tr>
<td>“Turning outbound checksum calculations on and off” on page 275</td>
<td>none</td>
<td>n/a</td>
</tr>
<tr>
<td>“Isolating data connections” on page 275</td>
<td>isolation</td>
<td>none, drop, forward</td>
</tr>
<tr>
<td>“Starting and stopping collection of QETH performance statistics” on page 277</td>
<td>performance_stats</td>
<td>0 or 1</td>
</tr>
<tr>
<td>“Capturing a hardware trace” on page 279</td>
<td>hw_trap</td>
<td>arm, disarm</td>
</tr>
</tbody>
</table>

*¹A 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 46: qeth functions and attributes in layer 2 mode

<table>
<thead>
<tr>
<th>Function</th>
<th>Corresponding attributes</th>
<th>Possible attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Configuring a network device as a member of a Linux bridge” on page 289</td>
<td>bridge_role, bridge_state, bridge_hostnotify</td>
<td>primary, secondary, none, active, standby, inactive 0 or 1</td>
</tr>
</tbody>
</table>
Table 46: qeth functions and attributes in layer 2 mode (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Corresponding attributes</th>
<th>Possible attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Advanced packet-handling configuration” on page 291</td>
<td>vnicc/bridge_invisible</td>
<td>0 or 1</td>
</tr>
<tr>
<td></td>
<td>vnicc/flooding</td>
<td>0 or 1</td>
</tr>
<tr>
<td></td>
<td>vnicc/learning</td>
<td>0 or 1</td>
</tr>
<tr>
<td></td>
<td>vnicc/mcast_flooding</td>
<td>0 or 1</td>
</tr>
<tr>
<td></td>
<td>vnicc/rx_bcast</td>
<td>0 or 1</td>
</tr>
<tr>
<td></td>
<td>vnicc/takeover_learning</td>
<td>0 or 1</td>
</tr>
<tr>
<td></td>
<td>vnicc/takeover_setvmac</td>
<td>integer in the range 60 - 86400</td>
</tr>
<tr>
<td></td>
<td>vnicc/learning_timeout</td>
<td>the default is 600</td>
</tr>
</tbody>
</table>

Table 47: qeth tasks and attributes in layer 3 mode.

<table>
<thead>
<tr>
<th>Task</th>
<th>Corresponding attributes</th>
<th>Possible attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Setting up a Linux router” on page 280</td>
<td>route4</td>
<td>primary_router</td>
</tr>
<tr>
<td></td>
<td>route6</td>
<td>secondary_router</td>
</tr>
<tr>
<td></td>
<td></td>
<td>primary_connector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>secondary_connector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>multicast_router</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no_router</td>
</tr>
<tr>
<td>“Enabling and disabling TCP segmentation offload”</td>
<td>none</td>
<td>n/a</td>
</tr>
<tr>
<td>on page 282</td>
<td></td>
<td></td>
</tr>
<tr>
<td>“Faking broadcast capability” on page 283</td>
<td>fake_broadcast ¹</td>
<td>0 or 1</td>
</tr>
<tr>
<td>“Taking over IP addresses” on page 284</td>
<td>ipa_takeover/enable</td>
<td>0 or 1 or toggle</td>
</tr>
<tr>
<td></td>
<td>ipa_takeover/add4</td>
<td>IPv4 or IPv6 IP address</td>
</tr>
<tr>
<td></td>
<td>ipa_takeover/add6</td>
<td>and mask bits</td>
</tr>
<tr>
<td></td>
<td>ipa_takeover/del4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ipa_takeover/del6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ipa_takeover/invert4</td>
<td>0 or 1 or toggle</td>
</tr>
<tr>
<td></td>
<td>ipa_takeover/invert6</td>
<td></td>
</tr>
<tr>
<td>“Configuring a device for proxy ARP” on page 287</td>
<td>rxip/add4</td>
<td>IPv4 IP address</td>
</tr>
<tr>
<td></td>
<td>rxip/del4</td>
<td></td>
</tr>
<tr>
<td>Configuring a device for NDP proxy</td>
<td>rxip/add6</td>
<td>IPv6 IP address</td>
</tr>
<tr>
<td></td>
<td>rxip/del6</td>
<td></td>
</tr>
<tr>
<td>“Configuring a device for virtual IP address (VIPA)’’ on page 288</td>
<td>vipa/add4</td>
<td>IPv4 or IPv6 IP address</td>
</tr>
<tr>
<td></td>
<td>vipa/add6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vipa/del4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vipa/del6</td>
<td></td>
</tr>
<tr>
<td>“Configuring a HiperSockets device for AF_IUCV addressing” on page 288</td>
<td>hsuid</td>
<td>1 to 8 characters</td>
</tr>
<tr>
<td>“Setting up a HiperSockets network traffic analyzer” on page 307</td>
<td>sniffer</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>
### Table 47: qeth tasks and attributes in layer 3 mode. (continued)

<table>
<thead>
<tr>
<th>Task</th>
<th>Corresponding attributes</th>
<th>Possible attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tips:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Use the qethconf command instead of using the attributes for IPA, proxy ARP, and VIPA directly (see “qethconf - Configure qeth devices” on page 704).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Your distribution might also provide a distribution-specific configuration tool. See your distribution documentation for distribution-specific alternatives.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 752.

• Write the device numbers of the subchannel triplet to the sysfs group attribute to only define a group device.
Issue a command of the form:

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

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

```
/sys/bus/ccwgroup/drivers/qeth/<read_device_bus_id>
```

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

Example
In this example (see Figure 85 on page 264), a single OSA-Express CHPID in QDIO mode is used to connect a Linux instance to a network.

Mainframe configuration:

```
OSA-Express
```

```
IBM mainframe
```

```
Linux
```

```
AA00
AA01
AA02
```

```
Network
```

```
Figure 85: Mainframe configuration
```

Linux configuration:
Assuming that 0.0.aa00 is the device bus-ID that corresponds to the read subchannel:

```
# echo 0.0.aa00,0.0.aa01,0.0.aa02 > /sys/bus/ccwgroup/drivers/qeth/group
```

This command results in the creation of the following directories in sysfs:

- `/sys/bus/ccwgroup/drivers/qeth/0.0.aa00`
- `/sys/bus/ccwgroup/devices/0.0.aa00`
- `/sys/devices/qeth/0.0.aa00`

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

Removing a qeth group device
Use the `ungroup` sysfs attribute to remove a qeth group device.

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

Procedure
To remove a qeth group device, write 1 to the `ungroup` attribute.

Issue a command of the form:

```
echo 1 > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/ungroup
```
Example
This command removes device 0.0.aa00:

```
echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.aa00/ungroup
```

Setting the layer2 attribute
If the detected hardware always runs in a specific discipline, the corresponding discipline module is automatically requested.

Before you begin
• To change a configured layer2 attribute, the network interface must be shut down and the device must be set offline.
• If you are using the layer2 option within a QDIO-based 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
The qeth device driver detects the mode of the VSWITCH or LAN to which the NIC is coupled, and sets this mode on the device. The z/VM definition "Ethernet mode" is available for VSWITCHes and for guest LANs of type QDIO.

Procedure
The qeth device driver separates the configuration options in sysfs according to the device discipline. Hence the first configuration action after you group the device must be the configuration of the discipline.

To set the discipline, issue a command of the form:

```
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:
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>
<thead>
<tr>
<th>Priority flag of the skb</th>
<th>Priority queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>3</td>
</tr>
<tr>
<td>2-3</td>
<td>2</td>
</tr>
<tr>
<td>4-5</td>
<td>1</td>
</tr>
<tr>
<td>≥6</td>
<td>0</td>
</tr>
</tbody>
</table>
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_queueing
```

Possible results are:

**by VLAN headers**
- if `prio_queueing_vlan` is set.

**by skb-priority**
- if `prio_queueing_skb` is set.

**by precedence**
- if `prio_queueing_prec` is set.

**by type of service**
- if `prio_queueing_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 268 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 49 on page 268.

<table>
<thead>
<tr>
<th>inbuf_size value</th>
<th>MFS</th>
<th>MTU</th>
</tr>
</thead>
<tbody>
<tr>
<td>16k</td>
<td>16 KB</td>
<td>8 KB</td>
</tr>
<tr>
<td>24k</td>
<td>24 KB</td>
<td>16 KB</td>
</tr>
<tr>
<td>40k</td>
<td>40 KB</td>
<td>32 KB</td>
</tr>
<tr>
<td>64k</td>
<td>64 KB</td>
<td>56 KB</td>
</tr>
</tbody>
</table>

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:

```bash
# 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.

```bash
# 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:

```bash
# 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 50 on page 269 for details.

<table>
<thead>
<tr>
<th>Value of card_type</th>
<th>Adapter type</th>
<th>Link type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSD_10GIG</td>
<td>OSA card in OSD mode</td>
<td>10 Gigabit Ethernet</td>
</tr>
<tr>
<td>OSD_1000</td>
<td></td>
<td>Gigabit Ethernet, 1000BASE-T</td>
</tr>
<tr>
<td>OSD_100</td>
<td></td>
<td>Fast Ethernet</td>
</tr>
<tr>
<td>OSD_GbE_LANE</td>
<td></td>
<td>Gigabit Ethernet, LAN Emulation</td>
</tr>
<tr>
<td>OSD_FE_LANE</td>
<td></td>
<td>Fast Ethernet, LAN Emulation</td>
</tr>
<tr>
<td>OSD_ATM_LANE</td>
<td></td>
<td>ATM, LAN Emulation</td>
</tr>
<tr>
<td>OSD_Express</td>
<td></td>
<td>Unknown</td>
</tr>
<tr>
<td>OSN</td>
<td>OSA for NCP</td>
<td>ESCON/CDLC bridge or N/A</td>
</tr>
<tr>
<td>OSM</td>
<td>OSA-Express for Unified Resource Manager</td>
<td>1000BASE-T</td>
</tr>
</tbody>
</table>
Table 50: Possible values of card_type and what they mean. (continued)

<table>
<thead>
<tr>
<th>Value of card_type</th>
<th>Adapter type</th>
<th>Link type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSX</td>
<td>OSA-Express for zBX</td>
<td>10 Gigabit Ethernet</td>
</tr>
<tr>
<td>HiperSockets</td>
<td>HiperSockets, CHPID type IQD</td>
<td>N/A</td>
</tr>
<tr>
<td>Virtual NIC QDIO</td>
<td>VSWITCH or guest LAN based on OSA</td>
<td>N/A</td>
</tr>
<tr>
<td>Virtual NIC Hiper</td>
<td>Guest LAN based on HiperSockets</td>
<td>N/A</td>
</tr>
<tr>
<td>Unknown</td>
<td>Other</td>
<td></td>
</tr>
</tbody>
</table>

Example

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

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

Setting a device online or offline

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” on page 270). 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:

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

**Example**

```bash
# 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:

```bash
# 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 270).
- 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 Open Systems Adapter-Express Customer's Guide and Reference, SA22-7935 for more details about MTU size.

For HiperSockets, the maximum MTU size is restricted by the maximum frame size as announced by the Licensed Internal Code (LIC). The maximum MTU is equal to the frame size minus 8 KB. Hence, the
possible frame sizes of 16 KB, 24 KB, 40 KB, or 64 KB result in maximum corresponding MTU sizes of 8 KB, 16 KB, 32 KB, or 56 KB.

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:

  ```bash
  # 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:

  ```bash
  # 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:

  ```bash
  # ip link set dev eth0 up
  ```

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

  ```bash
  # 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 284 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 287 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 288 for details.

You can use the qethconf command (see “qethconf - Configure qeth devices” on page 704) 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
```
Configuring checksum offload operations

Some operations can be offloaded to the OSA adapter, thus relieving the burden on the host CPU. The qeth device driver supports checksum offloading for TCP and UDP network packets.

The qeth device driver supports offloading the following checksum operations on both layer 2 and layer 3:

- Inbound (receive) checksum calculations
- Outbound (send) checksum calculations

The qeth device driver also supports offloading TSO segmentation, see “Enabling and disabling TCP segmentation offload” on page 282.

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 abbreviated example shows some of the offload settings:

```
# ethtool -k eth0
Features for eth0:
  rx-checksumming: on
tx-checksumming: on
tx-checksum-ipv4: on
  tx-checksum-ip-generic: off [fixed]
tx-checksum-ipv6: on
tx-checksum-fcoe-crc: off [fixed]
tx-checksum-sctp: off [fixed]
scatter-gather: on
tx-scatter-gather: on
  tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: on
tx-tcp-segmentation: on
  tx-tcp-ecn-segmentation: off [fixed]
tx-tcp6-segmentation: off [fixed]
udp-fragmentation-offload: off [fixed]
generic-segmentation-offload: off [requested on]
generic-receive-offload: on
large-receive-offload: off [fixed]
...
```

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
### 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:** For OSA-Express3 and earlier: 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. Linux instances that share an OSA port exchange packets directly. The packets are forwarded by the OSA adapter but do not go out on the LAN and no checksum offload is performed.

**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
  ```

### 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 86 on page 276).

![Diagram](image)

*Figure 86: 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.

To check whether the switch of the adapter is in reflective relay mode, read the sysfs attribute `switch_attrs`. The attribute lists all supported forwarding modes, with the currently active mode enclosed in square brackets. For example:

```
cat /sys/devices/qeth/0.0.f5f0/switch_attrs
802.1 [rr]
```

The example indicates that the adapter supports both 802.1 forwarding mode and reflective relay mode, and reflective relay mode is active.

Using a network adapter in VEPA mode achieves further isolation. VEPA mode forces traffic from the Linux guests to be handled by the external switch. For example, Figure 87 on page 277 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 87: 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:

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

- To set the isolation policy to ISOLATION_DROP:

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

- To set the isolation policy to ISOLATION_FORWARD:

  ```bash
  # 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:

  ```bash
  # 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, SC24-6267 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:
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:
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 TCP segmentation offload, and taking over IP addresses.

Use the layer 2 attribute to set the mode. See “Setting the layer2 attribute” on page 265 about setting the mode. See “Layer 2 and layer 3” on page 250 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:
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 51 on page 281.

Table 51: Summary of router setup values

<table>
<thead>
<tr>
<th>Router specification</th>
<th>OSA-Express CHPID in QDIO mode</th>
<th>HiperSockets CHPID</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary_router</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>secondary_router</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>primary_connector</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>secondary_connector</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>multicast_router</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>no_router</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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 301.

- **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 301).

- **secondary_connector**
  to make your Linux instance a backup connection between a HiperSockets network and an external network (see “HiperSockets Network Concentrator” on page 301).

Example
In this example (see Figure 88 on page 282), 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:
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:

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

To make Linux P a primary router for IPv4:

```bash
# echo primary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0.0400/route4
# echo primary_router > /sys/bus/ccwgroup/drivers/qeth/0.0.0.0200/route4
```

**Linux S configuration:**

To create the qeth group devices:

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

To make Linux S a secondary router for IPv4:

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

In this example, qeth device 0.0.1510 is defined as a primary router for IPv6:

```bash
/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 301 for further examples.

**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. TCP segmentation offload is supported for OSA connections on layer 3 only. VLAN interfaces inherit offload settings from their base interface.
Procedure

Outbound (TX) checksumming and scatter gather are prerequisites for TCP segmentation offload (TSO). You must turn on scatter gather and outbound checksumming before configuring TSO. All three options can be turned on or off with a single ethtool command of the form:

```
# ethtool -K <interface_name> tx <value> sg <value> tso <value>
```

where `<value>` is either on or off.

For more information about TX checksumming, see “Turning outbound checksum calculations on and off” on page 275.

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 253.

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

An OSA-Express CHPID in QDIO mode can take over IP addresses from any IBM Z operating system. IP takeover for HiperSockets CHPIDs is restricted to taking over addresses from other Linux instances in the same Central Electronics Complex (CEC).

IP address takeover between multiple CHPIDs requires ARP for IPv4 and Neighbor Discovery for IPv6. OSA-Express handles ARP transparently, but not Neighbor Discovery.

There are three stages to taking over an IP address:

1. Stage 1: Ensure that your qeth group device is enabled for IP takeover
2. Stage 2: Activate the address to be taken over for IP takeover
3. 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:

  ```bash
  # qethconf ipa list
  ```

- To activate an IP address for IP takeover, add it to the list.
  Issue a command of the form:

  ```bash
  # 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:

  ```bash
  # 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 704 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.

```bash
# 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.

```bash
# 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.

```bash
# 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.

```bash
# 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:
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 284).
- 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 285).

About this task

Be aware of the information in “Confirming that an IP address has been set under layer 3” on page 272 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 253.

The qeth device driver maintains a list of IP addresses for which a qeth group device handles ARP and issues gratuitous ARP packets. For more information about proxy ARP, see http://www.cisco.com/c/en/us/support/docs/ip/dynamic-address-allocation-resolution/13718-5.html

Use the qethconf command to display this list or to change the list by adding and removing IP addresses (see “qethconf - Configure qeth devices” on page 704).

Be aware of the information in “Confirming that an IP address has been set under layer 3” on page 272 when you work with proxy ARP.

**Example**

Figure 89 on page 287 shows an environment where proxy ARP is used.

![Figure 89: Example of proxy ARP usage](image)

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.
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 253.

IBM Z uses VIPAs to protect against certain types of hardware connection failure. You can assign VIPAs that are independent from particular adapter. VIPAs can be built under Linux using dummy devices (for example, "dummy0" or "dummy1").

The qeth device driver maintains a list of VIPAs that the OSA-Express adapter accepts for each qeth group device. Use the qethconf utility to add or remove VIPAs (see “qethconf - Configure qeth devices” on page 704).

For an example of how to use VIPA, see “Scenario: VIPA – minimize outage due to adapter failure” on page 293.

Be aware of “Confirming that an IP address has been set under layer 3” on page 272 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 346).

**Procedure**

To set an identifier, issue a command of this form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/qeth/0.0.a007/hsuid
```

The identifier is case-sensitive and must adhere to these rules:

- It must be 1 - 8 characters.
- It must be unique across your environment.
• It must not match any z/VM user ID in your environment. The AF_IUCV addressing family support also supports z/VM IUCV connections.

Example
In this example, MYHOST01 is set as the identifier for a HiperSockets device with bus ID 0.0.a007.

```
# echo MYHOST01 > /sys/bus/ccwgroup/drivers/qeth/0.0.a007/hsuid
```

Working with qeth devices in layer 2 mode

Tasks that you can perform on qeth devices in layer 2 mode include setting up an OSA or HiperSockets bridge port and tuning packet handling for a HiperSockets device with VNIC characteristics.

VNIC characteristics and the bridge port role are mutually exclusive.

Use the layer2 attribute to set the mode. See “Setting the layer2 attribute” on page 265 about setting the mode. See “Layer 2 and layer 3” on page 250 for general information about the layer 2 and layer 3 disciplines.

Configuring a network device as a member of a Linux bridge

You can define an OSA or HiperSockets device to be a bridge port, which allows it to act as a member of a Linux software bridge. Use the bridge_role attribute of a network device in layer 2 to make it receive all traffic with unknown destination MAC addresses.

Alternatively, use VNIC characteristics to configure a layer 2 network device to receive all unknown traffic (see “Advanced packet-handling configuration” on page 291).

Before you begin

To use the bridging support, you need OSA or HiperSockets hardware that supports layer 2 SETBRIDGEPORT functionality.

You can have one active bridge port per Internal Queued Direct Communication (IQD) channel. You can have either only secondary bridge ports, or one primary and several secondary bridge ports.

Devices for which VNIC characteristics are configured cannot also be configured as bridge ports.

For more information about the bridge port concept, see “Layer 2 promiscuous mode” on page 256.

About this task

The following sysfs attributes control the bridge port functions. The attributes can be found in the /sys/bus/ccwgroup/drivers/qeth/<device_bus_id> directory.

bridge_role
Read-write attribute that controls the role of the port. Valid values are:

  primary
  Assigns the port the primary bridge port role.

  secondary
  Assigns the port a secondary bridge port role.

  none
  Revokes existing bridge port roles and indicates that no role is assigned.

Assigning a role directly to a port prevents use of the bridge_reflect_promisc attribute.

bridge_state
Read-only attribute that shows the state of the port. Valid values are:
active
The port is assigned a bridge port role and is switched into active state by the adapter. The device receives frames that are addressed to unknown MAC addresses.

standby
The port is assigned a bridge port role, but is not currently switched into active state by the adapter. The device does not receive frames that are destined to unknown MAC addresses.

inactive
The port is not assigned a bridge port role.

bridge_hostnotify
HiperSockets only: Read-write attribute that controls the sending of notifications for the port. When you enable notifications (even if notifications were already enabled), udev events are emitted for all currently connected communication peers in quick succession. After that, a udev event is emitted every time a communication peer is connected, or a previously connected peer is disconnected. Any user space program that monitors these events must repopulate its list of registered peers every time the status of the bridge port device changes to enable notifications.

Valid values are:

1
The port is set to send notifications.

0
Notifications are turned off.

Notifications about the change of the state of bridge ports, and (if enabled) about registration and deregistration of communication peers on the LAN are delivered as udev events. The events are described in the file Documentation/s390/qeth.txt in the Linux kernel source tree.

bridge_reflect_promisc
Read-write attribute that, when set, makes the bridge-port role of the port follow ("reflect") the promiscuity flag (IFF_PROMISC) of the corresponding Linux network interface. You can specify the following values:

none
Setting and resetting the promiscuous mode on the network interface has no effect on the bridge-port role of the underlying port.

primary
Setting or resetting the promiscuous mode on the network interface that is served by this device causes the driver to attempt assigning (or resetting) the primary role to the port. If a port with the primary role exists, assignment fails.

secondary
Setting or resetting the promiscuous mode on the network interface that is served by this device causes the driver to attempt assigning (or resetting) the secondary role to the port.

Setting bridge_reflect_promisc to anything but none causes the bridge_role attribute to become read-only. The role of a port changes as a result of setting or unsetting the promiscuity flag (IFF_PROMISC) of the corresponding network interface. You can check the currently assigned role by reading the bridge_role attribute.

Procedure
1. To configure a network device as a bridge, issue a command of this form:

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

   Setting the bridge_role attribute requires the bridge_reflect_promisc attribute to be none.
   Alternatively, to make the bridge-port role of the port follow the promiscuity flag (IFF_PROMISC) of the corresponding Linux network interface, issue a command of the following form:

   # echo 0 > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/bridge_reflect_promisc

   Setting the bridge_reflect_promisc attribute requires the bridge_role attribute to be none.
# echo <value> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/bridge_reflect_promisc

where valid values are:

• primary
• secondary
• none

2. Check the state of the bridge port by reading the bridge_state attribute. Issue a command of this form:

# cat > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/bridge_state

where valid values are:

• active
• standby
• inactive

**Example**

In this example, a network device with bus ID 0.0.a007 is defined as a primary bridge port.

```bash
# echo primary > /sys/bus/ccwgroup/drivers/qeth/0.0.a007/bridge_role
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.a007/bridge_state
active
```

**What to do next**

You can specify up to four secondary bridge ports together with one primary bridge port. If the primary bridge port fails, one of these bridge ports takes over. For each secondary bridge port, set bridge_role to secondary.

**Advanced packet-handling configuration**

Use VNIC characteristics to control how HiperSockets devices in layer 2 mode handle packets with unknown MAC addresses.

**Before you begin**

• VNIC characteristics are supported for HiperSockets devices only.
• VNIC characteristics are supported for layer 2 mode only.
• VNIC characteristics cannot be configured on devices that are configured as bridge ports.

**About this task**

You can configure and fine-tune a promiscuous mode for incoming packets. You can configure the device to receive all packets regardless of the MAC address, or you can reject incoming multicast packets, or broadcast packets, or both.

For Linux instances that host multiple guest operating systems with different MAC addresses, you can configure the device to learn and handle these MAC addresses. The device then provides functions similar to a switch or to a software bridge.

The VNIC characteristics also include settings that can protect the MAC address of the device from being taken over by another device. You can deny takeover, or you can explicitly permit takeover to configure redundancy.

The VNIC characteristics of a HiperSockets device are represented by sysfs attributes in /sys/devices/qeth/<device_bus_id>/vnicc:
flooding
With flooding enabled, the device receives packets to any unknown destination MAC addresses. Valid values are 0 for disabled and 1 for enabled. By default, flooding is disabled.

mcast_flooding
With multicast flooding enabled, the device receives packets to multicast MAC addresses. Valid values are 0 for disabled and 1 for enabled. By default, multicast flooding is disabled.

rx_bcast
With broadcast receiving enabled, the device receives packets with the broadcast destination MAC address. Valid values are 0 for disabled and 1 for enabled. By default, the device is enabled to receive broadcast packets.

learning
With learning enabled, the device assembles a list of MAC addresses of outgoing packets. An entry is added to the list if a MAC address is unknown and has not already been learned by another device. The device then receives incoming packets to any listed MAC addresses. Valid values are 0 for disabled and 1 for enabled. By default, learning is disabled.

A learned MAC address is dropped from the list of learned MAC addresses unless packets with this MAC address are received or sent within a specific timeout period. The default timeout period is 600 s. You can specify a different timeout period with the learning_timeout attribute.

takeover_setvmac
With this option enabled, the device's MAC address can be configured on a different device. Valid values are 0 for disabled and 1 for enabled. By default, this option is disabled and the MAC address cannot be configured on a different device.

takeover_learning
With takeover by learning enabled, the MAC address of this device can be learned on a different device and, thus, taken over by this other device. Valid values are 0 for disabled and 1 for enabled. By default, takeover by learning is disabled.

bridge_invisible
With bridge-port invisible enabled, packets are not transferred between the device and any other device that is configured as a bridge port. Valid values are 0 for disabled and 1 for enabled. By default, this option is disabled and, thus, traffic to and from bridge ports is permitted.

learning_timeout
With learning enabled, this attribute specifies a timeout period, in seconds. A MAC address is dropped from the list of learned MAC addresses if this timeout period expires without any packets with this MAC address being received or sent.

You can set this timeout period by writing a value in the range 60 - 86400 to the attribute. The default is 600. The timeout must be set before learning is enabled on the device.

Procedure
1. Optional: To read a VNIC characteristic setting from sysfs, issue a command of this form:

   # cat /sys/devices/qeth/<device_bus_id>/vnicc/<attribute>

   where <device_bus_id> is the device-bus ID of the HiperSockets device and <attribute> is one of the attributes that represent the VNIC characteristics.

   Example:

   # cat /sys/devices/qeth/0.0.a016/vnicc/learning

   0

   Tip: For an overview of all VNIC characteristics of the device, find the interface name of the device, then use the lsqeth command.

   Example:
# cat /sys/devices/qeth/0.0.a016/if_name
eth0
# lsqeth eth0 | grep vnicc
   vnicc/bridge_invisible : 0
   vnicc/flooding : 0
   vnicc/learning : 0
   vnicc/learning_timeout : 600
   vnicc/mcast_flooding : 1
   vnicc/rx_bcast : 1
   vnicc/takeover_learning : 0
   vnicc/takeover_setvmac : 0

2. To set a VNIC characteristic issue a command of this form:

```
# chzdev <device_bus_id> vnicc/<attribute>=<value>
```

where `<device_bus_id>` is the device-bus ID of the HiperSockets device, `<attribute>` is one of the attributes that represent the VNIC characteristics, and `<value>` is the value to be set.

This setting persists across re-boots. For more details, see Chapter 29, “Persistent device configuration,” on page 377. To apply this setting to the running system only, use the chzdev command with the -a option or use the corresponding sysfs attribute.

**Example:** In this example, learning is enabled for a device with bus-ID 0.0.a016.

```
# chzdev 0.0.a016 vnicc/learning=1
```

or, using sysfs:

```
# echo 1 > /sys/devices/qeth/0.0.a016/vnicc/learning
```

**Example**
This example shows a typical configuration for a bridge-like behavior of the device.

```
# lsqeth eth0 | grep vnicc
   vnicc/bridge_invisible : 0
   vnicc/flooding : 1
   vnicc/learning : 1
   vnicc/learning_timeout : 600
   vnicc/mcast_flooding : 1
   vnicc/rx_bcast : 1
   vnicc/takeover_learning : 1
   vnicc/takeover_setvmac : 1
```

**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 272 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” on page 294.
3. Set up routes to the virtual IP address, on clients or gateways. To do so, you can use either:
   - Static routing (shown in the example of Figure 90 on page 295).
   - Dynamic routing. For details of how to configure routes, you must see the documentation that is delivered with your routing daemon (for example, zebra or gated).

Adapter outage
If outage of an adapter occurs, you must switch adapters.

Procedure
- Under static routing:
  a) Delete the route that was set previously.
  b) Create an alternative route to the virtual IP address.
- Under dynamic routing, see the documentation that is delivered with your routing daemon for details.

Example of how to set up standard VIPA
This example shows you how to configure VIPA under static routing, and how to switch adapters when an adapter outage occurs.

About this task
Figure 90 on page 295 shows the network adapter configuration that is used in the example.
**Procedure**

1. Define the real interfaces.

   ```bash
   [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:

   ```bash
   [server]# modprobe dummy
   ```

3. Create a dummy interface with a virtual IP address 198.51.100.100 and a netmask 255.255.255.0:

   ```bash
   [server]# ip addr add 198.51.100.100/24 dev dummy0
   [server]# ip link set dev dummy0 up
   ```

4. Enable the network devices for this VIPA so that it accepts packets for this IP address.
   - IPv4 example:
     ```bash
     [server]# qethconf vipa add 198.51.100.100 eth0
     qethconf: Added 198.51.100.100 to /sys/class/net/eth0/device/vipa/add4.
     qethconf: Use "qethconf vipa list" to check for the result
     [server]# qethconf vipa add 198.51.100.100 eth1
     qethconf: Added 198.51.100.100 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:
     ```bash
     [server]# qethconf vipa add 2002::1235:5678 eth0
     qethconf: Added 2002::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::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:

   ```bash
   [server]# qethconf vipa list
   vipa add 198.51.100.100 eth0
   vipa add 198.51.100.100 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 198.51.100.100 via 10.1.0.2
```

**What to do next**

Now assume that an adapter outage occurs. You must then:

1. Delete the previously created route.

```
[router]# ip route del 198.51.100.100
```

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

```
[router]# ip route add 198.51.100.100 via 10.2.0.2
```

**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 297). 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 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 297 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 297 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 fulfill 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`. qeth device driver for OSA-Express (QDIO) and HiperSockets 297
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 91 on page 298 shows a configuration where two applications with VIPA 198.51.100.100 and 198.51.100.200 are to be set up for source VIPA with a local round robin policy.

![Diagram of IBM mainframe configuration](image)

**Figure 91: Example of using source VIPA**

The required entry in the Source VIPA configuration file is:

```
9.0.0.0/8 lrr 198.51.100.100 198.51.100.200
```
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 92 on page 299, 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.

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 93 on page 300 shows, a single router can provide the interfaces for all VLANs that appeared as separate LAN segments in the previous figure.
Figure 93: Switched VLAN network

Figure 94 on page 300 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 94: 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
```
Example: Creating two VLANs

VLANs are allocated in an existing interface that represents a physical Ethernet LAN.

The following example creates two VLANs, one with ID 3 and one with ID 5.

```
ip addr add 198.51.160.23/19 dev eth1
ip link set dev eth1 up
ip link add dev eth1.3 link eth1 type vlan id 3
ip link add dev eth1.5 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 198.51.1xx.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
```

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 Z Server environment
- From systems that are connected by a different HiperSockets Network Concentrator into a 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 287) 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 280).
  
  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_connection" or to "secondary_connection". 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

  \[ \texttt{sysctl -w net.ipv4.ip_forward=1} \]

  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 \texttt{killall ip_watcher.pl} to remove changes that are caused by running HiperSockets Network Concentrator.
Availability setups

If a connector system fails during operation, it can simply be restarted. If all the startup commands are run automatically, it will instantaneously be operational again after booting. Two common availability setups are mentioned here:

One connector partition and one monitoring system
As soon as the monitoring system cannot reach the connector for a specific timeout (for example, 5 seconds), it restarts the connector. The connector itself monitors the monitoring system. If it detects (with a longer timeout than the monitoring system, for example, 15 seconds) a monitor system failure, it restarts the monitoring system.

Two connector systems monitoring each other
In this setup, there is an active and a passive system. As soon as the passive system detects a failure of the active connector, it takes over operation. To take over operation, it must reset the other system to release all OSA resources for the multicast_router operation. The failed system can then be restarted manually or automatically, depending on the configuration. The passive backup HiperSockets interface can either switch into primary_connector mode during the failover, or it can be set up as secondary_connector. A secondary_connector takes over the connecting function, as soon as there is no active primary_connector. This setup has a faster failover time than the first one.

Hints

- The MTU of the OSA and HiperSockets link should be of the same size. Otherwise, multicast packets that do not fit in the link's MTU are discarded as there is no IP fragmentation for multicast bridging. Warnings are printed to /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 95 on page 304 shows a network environment where a Linux instance C acts as a network concentrator that connects other operating system instances on a HiperSockets LAN to an external LAN.
Setup for the network concentrator C:
The HiperSockets interface hsi0 (device bus-ID 0.0.a1c0) has IP address 10.20.30.51, and the netmask is 255.255.255.0. The default gateway is 10.20.30.1.

Issue:
```bash
# 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:
```bash
# echo multicast_router > /sys/bus/ccwgroup/drivers/qeth/0.0.a1c4/route4
```

To enable IP forwarding issue:
```bash
# 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:
```bash
# ip route del 10.20.30/24
```

To start the HiperSockets network concentrator, run the script `start_hsnc.sh`. Issue:
```bash
# 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 96 on page 305 shows the example of Figure 95 on page 304 with an additional mainframe. On the second mainframe a Linux instance D acts as a HiperSockets network concentrator.
The configuration of C, G, and the workstation remain the same as for Figure 95 on page 304.

**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 configured with the layer 3 discipline, 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 254.
Because a HiperSockets connection configured with the layer 3 discipline 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 a Linux instance 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 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:
  a) Ensure that the qeth device driver was compiled into the Linux kernel or that the qeth device driver was loaded as a module.
  b) 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 752. The qeth device driver automatically sets the `buffer_count` attribute to 128 for the analyzing device.
  c) Activate the device (no IP address is needed):

    ```
    # ip link set hsi0 up
    ```
  d) 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:**
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, SC24-6267.

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:

a) Check if the z/VM guest virtual machine already has the required authorizations. Enter a CP command of this form:

```bash
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.

b) If the output from step “1” on page 308 does not list the guest virtual machine, check if the guest is authorized for the virtual switch. Enter a CP command of this form:

```bash
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:

```bash
set vswitch <switchname> revoke <userid>
```
where `<switchname>` is the name of the virtual switch and `<userid>` identifies the z/VM guest virtual machine.

c) Authorize the Linux instance for the switch and for promiscuous mode. Enter a CP command of this form:

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

where `<switchname>` is the name of the virtual switch and `<userid>` identifies the z/VM guest virtual machine.

For details about the CP commands 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*, SC24-6268.
Chapter 18. 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 38 on page 246 and Table 39 on page 246.

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 701 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
- net-snmp package 5.1.x 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 the net-snmp package. The subagent uses the Agent eXtensibility (AgentX) protocol to communicate with the master agent.

net-snmp is an open source project that is owned by the Open Source Development Network, Inc. (OSDN). For more information on net-snmp visit net-snmp.sourceforge.net

When the master agent (snmpd) is started on a Linux system, it binds to a port (default 161) and awaits requests from SNMP management software. Subagents can connect to the master agent to support MIBs of special interest (for example, OSA-Express MIB). When the osasnmpd subagent is started, it retrieves the MIB objects of the OSA-Express features currently present on the Linux system. It then registers with the master agent the object IDs (OIDs) for which it can provide information.

An OID is a unique sequence of dot-separated numbers (for example, .1.3.6.1.4.1.2) that represents a particular information. OIDs form a hierarchical structure. The longer the OID, that is the more numbers it is made up of, the more specific is the information that is represented by the OID. For example, .1.3.6.1.4.1.2 represents all network information related to IBM, 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 312). You do not need to update the subagent. Place the updated MIB file in a directory that is searched by the master agent.
Figure 97: OSA-Express SNMP agent flow

Figure 97 on page 312 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
```

<table>
<thead>
<tr>
<th>USER</th>
<th>PID</th>
<th>USER</th>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>687</td>
<td>root</td>
<td>729</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>659</td>
<td>3</td>
</tr>
<tr>
<td>00:00:00 snmpd</td>
<td>00:00:00 osasnmpd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00:00:00 osasnmpd</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 can use 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.
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:

```c
->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 315).

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](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 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:

<table>
<thead>
<tr>
<th>#</th>
<th>sec.name</th>
<th>source</th>
<th>community</th>
</tr>
</thead>
<tbody>
<tr>
<td>com2sec</td>
<td>osasec</td>
<td>default</td>
<td>osacom</td>
</tr>
<tr>
<td>com2sec</td>
<td>pubsec</td>
<td>localhost</td>
<td>public</td>
</tr>
</tbody>
</table>

4. Find the group section.

Use the security name to define a group with different versions of the master agent for which you want to grant access rights. Include a line of this form for each master agent version:

```
group <group-name> <security-model> <security-name>
```

where:

- **<group-name>** is a group name of your choice.
- **<security-model>** is the security model of the SNMP version.
- **<security-name>** is the same as in step “3” on page 313.

Example:

<table>
<thead>
<tr>
<th>#</th>
<th>groupName</th>
<th>securityModel</th>
<th>securityName</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>osagroup</td>
<td>v1</td>
<td>osasec</td>
</tr>
<tr>
<td>group</td>
<td>osagroup</td>
<td>v2c</td>
<td>osasec</td>
</tr>
<tr>
<td>group</td>
<td>osagroup</td>
<td>usm</td>
<td>osasec</td>
</tr>
<tr>
<td>group</td>
<td>osasnmpd</td>
<td>v2c</td>
<td>pubsec</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>#</th>
<th>name</th>
<th>incl/excl</th>
<th>subtree</th>
<th>mask(optional)</th>
</tr>
</thead>
<tbody>
<tr>
<td>view</td>
<td>allview</td>
<td>included</td>
<td>.1</td>
<td></td>
</tr>
<tr>
<td>view</td>
<td>osaview</td>
<td>included</td>
<td>.1.3.6.1.4.1.2</td>
<td></td>
</tr>
<tr>
<td>view</td>
<td>ifmibview</td>
<td>included</td>
<td>.1.3.6.1.2.1.2</td>
<td></td>
</tr>
<tr>
<td>view</td>
<td>ifmibview</td>
<td>included</td>
<td>.1.3.6.1.2.1.1</td>
<td></td>
</tr>
</tbody>
</table>

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 314.

<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:

<table>
<thead>
<tr>
<th>#</th>
<th>group</th>
<th>context</th>
<th>sec.model</th>
<th>sec.level</th>
<th>prefix</th>
<th>read</th>
<th>write</th>
<th>notif</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>osagroup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>allview</td>
<td>osaview</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>osasnmwd</td>
<td>v2c</td>
<td>noauth</td>
<td>exact</td>
<td>ifmibview</td>
<td>none</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>

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.

8. Save and close `snmpd.conf`.

Example of an `snmpd.conf` file:

```
#       sec.name       source        community
com2sec osasec         default       osacom
com2sec pubsec         localhost     public
#       groupName      securityModel securityName
group   osagroup       v1            osasec
group   osagroup       v2c           osasec
group   osagroup       usm           osasec
group   osasnmwd       v2c           pubsec
#       name          incl/excl  subtree        mask(optional)
view    allview       included   .1
view    osaview       included   .1.3.6.1.4.1.2
view    ifmibview     included   .1.3.6.1.2.1.1
#       group          context sec.model sec.level prefix read       write    notif
access  osagroup       any       noauth    exact   allview    osaview   none
access  osasnmwd       v2c       noauth    exact   ifmibview  none     none
master  agentx
```

9. Open `snmp.conf` with your preferred text editor.

**Tip:** See `man snmp.conf` for possible locations of `snmp.conf`.

10. Include a line of this form to specify the directory to be searched for MIBs:

```
mibdirs +<mib-path>
```

**Example:**

```
mibdirs +/usr/share/snmp/mibs
```

11. Include a line of this form to make the OSA-Express MIB available to the master agent:

```
mibs +<mib-name>
```

where `<mib-name>` is the stem of the MIB file name you assigned in “Downloading the IBM OSA-Express MIB” on page 312.

**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:

```plaintext
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 313.

Example:

```plaintext
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” on page 316
- “Checking the log file” on page 316
- “Issuing queries” on page 317
- “Stopping osasnmpd” on page 318

**Starting the osasnmpd subagent**

Use the `osasnmpd` command to start the osasnmpd subagent.

**Procedure**

Start the osasnmpd subagent with the `osasnmpd` command:

```bash
# osasnmpd
```

The osasnmpd subagent starts a daemon that is called osasnmpd.

For command options see “osasnmpd – Start OSA-Express SNMP subagent” on page 701.

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.
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):

```bash
# 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:

```bash
# 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 313.

If you provided defaults for the SNMP version and the community (see step “12” on page 316), you can omit the -v and -c options:

```bash
# 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:

```bash
# 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:

```bash
# snmpwalk -OS localhost ibmOsaExpChannelNumber
IBM-OSA-MIB::ibmOSAExpPortType.6 = Hex-STRING: 70 80 81
IBM-OSA-MIB::ibmOSAExpPortType.7 = Hex-STRING: 70 80 81
IBM-OSA-MIB::ibmOSAExpPortType.8 = Hex-STRING: 70 80 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 701). 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      ADDR      AUTHPRIV  TTY      STAT   TIME  COMMAND
root       687     0.00:00:00 00:00:00 snmpd
root       729     0.00:00:00 00:00:00 osasnmpd
# kill -9 729
```

```
Chapter 19. 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 52 on page 319 shows the supported OSA-Express features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>z14 and z14 ZR1</th>
<th>z13 and z13s</th>
<th>zEC12 and zBC12</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSA-Express6S</td>
<td>1000Base-T Ethernet</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
<tr>
<td>OSA-Express5S</td>
<td>1000Base-T Ethernet</td>
<td>1000Base-T Ethernet</td>
<td>Not supported</td>
</tr>
<tr>
<td>OSA-Express4S</td>
<td>1000Base-T Ethernet</td>
<td>1000Base-T Ethernet</td>
<td>1000Base-T Ethernet</td>
</tr>
<tr>
<td>OSA-Express3</td>
<td>Not supported</td>
<td>Not supported</td>
<td>1000Base-T Ethernet</td>
</tr>
</tbody>
</table>

Table 53: The LCS device driver supported OSA features on earlier mainframes

<table>
<thead>
<tr>
<th>Feature</th>
<th>z196, z114, and System z10</th>
<th>System z9</th>
<th>eServer zSeries</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSA-Express4S</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
<tr>
<td>OSA-Express3</td>
<td>1000Base-T Ethernet</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
<tr>
<td>OSA-Express2</td>
<td>1000Base-T Ethernet</td>
<td>1000Base-T Ethernet</td>
<td>Not supported</td>
</tr>
<tr>
<td>OSA-Express</td>
<td>Not supported</td>
<td>• Fast Ethernet • 1000Base-T Ethernet</td>
<td>• Fast Ethernet • 1000Base-T Ethernet (z890, z990)</td>
</tr>
</tbody>
</table>

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 98: 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 other network device drivers. 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 252.

**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 99 on page 320).

```
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 99: 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” on page 321
- “Removing an LCS group device” on page 322
- “Specifying a timeout for LCS LAN commands” on page 322
- “Setting an LCS group device online or offline” on page 323
- “Activating and deactivating an interface” on page 324

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:
Example

Assuming that 0.0.d000 is the device bus-ID that corresponds to a read subchannel:

```bash
# 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:

```bash
# echo 1 > /sys/bus/ccwgroup/drivers/lcs/<device_bus_id>/ungroup
```

**Example**

This command removes device 0.0.d000:

```bash
# 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:

```bash
# 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.

```bash
# 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:

```bash
# 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:

```bash
# 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:

```bash
# 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:

```bash
# readlink /sys/class/net/eth0/device
../../../0.0.d000
```

To set the device offline issue:

```bash
# echo 0 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/online
```
Activating and deactivating an interface

Use the `ip` command or equivalent to activate or deactivate an interface.

**About this task**

Before you can activate an interface, you must set the group device online and find out the interface name that is assigned by the LCS device driver. See “Setting an LCS group device online or offline” on page 323.

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 20. 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 mainframe operating system instances.

Communications Server for Linux uses MPC connections to connect Linux on 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 100 on page 325 illustrates the I/O subchannel interface. The device bus-IDs that correspond to the two subchannels must be configured for control unit type 3088.

![Diagram showing 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 101 on page 326 shows a CTC interface that is connected to a network.

![Figure 101: Network connection](image)

**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 102 on page 327).
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” on page 327
- “Removing a CTCM group device” on page 328
- “Displaying the channel type” on page 329
- “Setting the protocol” on page 329
- “Setting a device online or offline” on page 330
- “Setting the maximum buffer size” on page 331 (CTC only)
- “Activating and deactivating a CTC interface” on page 331 (CTC only)
- “Recovering a lost CTC connection” on page 332 (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:
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 z/OS, for example, a z/VM TCP service machine. This value is the default.

1  
This protocol provides enhanced package checking for Linux peers.

3  
This protocol provides for compatibility with z/OS peers.

4  
This protocol provides for MPC connections to VTAM on traditional mainframe operating systems.

**Procedure**

Issue a command of this form:

```
# echo <value> > /sys/bus/ccwgroup/drivers/ctcm/<device_bus_id>/protocol
```

**Example**

In this example, the protocol is set for a CTCM group device 0.0.2000:

```
# echo 4 > /sys/bus/ccwgroup/drivers/ctcm/0.0.2000/protocol
```
Setting a device online or offline

Use the online device group attribute to set a CTCM device online or offline.

About this task

Setting a group device online associates it with an interface name. Setting the group device offline and back online with the same protocol preserves the association with the interface name. If you change the protocol before you set the group device back online, the interface name can change as described in “Interface names assigned by the CTCM device driver” on page 326.

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 \(<\text{minimum MTU} + \text{header size}>\) to \(<\text{maximum MTU} + \text{header size}>\). 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 \(<\text{buffer size} - \text{header size}>\).

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 \(<\text{value}>\) 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 330.

About this task

```
Figure 103: Syntax for setting an IP address for a CTC interface with the ip command
```

```
Figure 104: Syntax for activating a CTC interface with the ip command
```
Where:

<interface>
  is the interface name that was assigned when the CTCM group device was set online.

<ip_address>
  is the IP address that you want to assign to the interface.

<peer_ip_address>
  is the IP address of the remote side.

<max_transfer_unit>
  is the size of the largest IP packet that might be transmitted. Be sure to use the same MTU size on both sides of the connection. The MTU must be in the range of 576 byte to 65,536 byte (64 KB).

Figure 105: Syntax for deactivating a CTC interface with the ip command

Where:

<interface>
  is the interface name that was assigned when the CTCM group device was set online.

Procedure

• Use ip or an equivalent command to activate the interface:
• To deactivate an interface, issue a command of this form:

  # ip link set dev <interface> down

Examples

• This example activates a CTC interface 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 331.
3. Activate the interface on the crashed side and on the peer.

For details, see “Activating and deactivating a CTC interface” on page 331.
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 331.

**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” on page 333
- “Connecting Linux on z/VM to another guest of the same z/VM system” on page 334

**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 106 on page 333 illustrates a CTC setup with a peer in a different LPAR.

**Procedure**
1. Create a CTCM group device.
   Issue:
   ```bash
   # echo 0.0.f008,0.0.f009 > /sys/bus/ccwgroup/drivers/ctcm/group
   ```
2. Confirm that the device uses CTC FICON or CTC ESCON:
   ```bash
   # cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/type
   ```
   In this example, ESCON is used. You would proceed the same for FICON.
3. Select a protocol.
The choice depends on the peer.

<table>
<thead>
<tr>
<th>If the peer is ...</th>
<th>Choose ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>1</td>
</tr>
<tr>
<td>z/OS</td>
<td>3</td>
</tr>
<tr>
<td>Any other operating system</td>
<td>0</td>
</tr>
</tbody>
</table>

Assuming that the peer is Linux:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/protocol
```

4. Set the CTCM group device online and find out the assigned interface name:

```
# echo 1 > /sys/bus/ccwgroup/drivers/ctcm/0.0.f008/online
# 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 107 on page 334 illustrates a CTC setup with a peer in the same z/VM.

![Figure 107: CTC scenario with peer in the same z/VM](image)

**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:

```plaintext
special f004 ctca
special f005 ctca
```

Alternatively, you can use the CP commands:

```plaintext
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:

```plaintext
couple f004 to guestp f011
couple f005 to guestp f010
```

Be sure that you couple the read channel to the peers write channel and vice versa.

4. From your booted Linux instance, create a CTCM group device. Issue:

```bash
# echo 0.0.f004,0.0.f005 > /sys/bus/ccwgroup/drivers/ctcm/group
```

5. Confirm that the group device is a virtual CTCA device:

```bash
# cat /sys/bus/ccwgroup/drivers/ctcm/0.0.f004/type
CTC/A
```

6. Select a protocol.
   The choice depends on the peer.

<table>
<thead>
<tr>
<th>If the peer is ...</th>
<th>Choose ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>1</td>
</tr>
<tr>
<td>z/OS</td>
<td>3</td>
</tr>
<tr>
<td>Any other operating system</td>
<td>0</td>
</tr>
</tbody>
</table>

Assuming that the peer is Linux:

```bash
# 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:

```bash
# 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:

```bash
# 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 331).
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 331 if the peer is not Linux, see the operating system documentation of the peer.
Chapter 21. 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 108 on page 337.

![Figure 108: Direct IUCV connection](image)

If your IUCV connection is to a router, the peer can be remote and connected through an external network, see Figure 109 on page 338.
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, SC24-6331.

**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 339) 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 110 on page 338 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 110: 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.

```bash
# 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 maximum 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” on page 339
- “Changing the peer” on page 340
- “Setting the maximum buffer size” on page 341
- “Activating an interface” on page 342
- “Deactivating and removing an interface” on page 342

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:

```bash
# 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 address | add <ip_address> | dev <interface> | peer <peer_ip_address> |

Figure 111: ip syntax for setting an IP address for an IUCV connection

| ip link set | dev <interface> | up | mtu 9216 |

| mtu <max_transfer_unit> |

Figure 112: ip syntax for activating an IUCV interface

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 113 on page 343.

![Figure 113: IUCV connection scenario](image)

**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>
   ```
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:

<table>
<thead>
<tr>
<th>Home</th>
<th>1.2.3.4 LNK1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.2.3.5 LNK2</td>
</tr>
<tr>
<td>Device DEV1</td>
<td>IUCV 0 0 LNX1 A</td>
</tr>
<tr>
<td>Link LNK1</td>
<td>IUCV 0  DEV1</td>
</tr>
<tr>
<td>Device DEV2</td>
<td>IUCV 0 0 LNX2 A</td>
</tr>
<tr>
<td>Link LNK2</td>
<td>IUCV 0  DEV2</td>
</tr>
<tr>
<td>Start DEV1</td>
<td></td>
</tr>
<tr>
<td>Start DEV2</td>
<td></td>
</tr>
</tbody>
</table>

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 OBEXFILE 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 339.
2. Create an IUCV interface for connecting to the service machine:

   ```bash
   # 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:

   ```bash
   # ip addr add 1.2.3.4 dev iucv0 peer 1.2.3.10
   # ip link set dev iucv1 up mtu 32764
   ```
Chapter 22. AF_IUCV address family support

The AF_IUCV address family provides an addressing mode for communications between applications that run on Z.

This addressing mode can be used for connections through real HiperSockets and through the z/VM Inter-User Communication Vehicle (IUCV).

Support for AF_IUCV based connections through real HiperSockets requires Completion Queue Support. HiperSockets devices facilitate connections between applications across LPARs within an IBM Z. In particular, an application that runs on an instance of Linux on Z can communicate with:

• Itself
• Other applications that run on the same Linux instance
• An application on an instance of Linux on 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 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 114 on page 346 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 114: 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 288).

### Setting up your z/VM guest virtual machine for IUCV

You must specify suitable IUCV statements for your z/VM guest virtual machine.

For details and for general IUCV setup information for z/VM guest virtual machines, see z/VM: CP Programming Services, SC24-6272 and z/VM: CP Planning and Administration, SC24-6271.

### Granting IUCV authorizations

Use the IUCV statement to grant the necessary authorizations.

**IUCV ALLOW**
allows any other z/VM virtual machine to establish a communication path with this z/VM virtual machine. With this statement, no further authorization is required in the z/VM virtual machine that initiates the communication.

**IUCV ANY**
allows this z/VM guest virtual machine to establish a communication path with any other z/VM guest virtual machine.

**IUCV <user ID>**
allows this z/VM guest virtual machine to establish a communication path to the z/VM guest virtual machine with the z/VM user ID <user ID>.

You can specify multiple IUCV statements. To any of these IUCV statements you can append the MSGLIMIT <limit> parameter. <limit> specifies the maximum number of outstanding messages that are
allowed for each connection that is authorized by the statement. If no value is specified for MSGLIMIT, AF_IUCV requests 65 535, which is the maximum that is supported by IUCV.

### Setting a connection limit

Use the OPTION statement to limit the number of concurrent connections.

**OPTION MAXCONN** `<maxno>`

- `<maxno>` specifies the maximum number of IUCV connections that are allowed for this virtual machine. The default is 64. The maximum is 65 535.

### Example

These sample statements allow any z/VM guest virtual machine to connect to your z/VM guest virtual machine with a maximum of 10 000 outstanding messages for each incoming connection. Your z/VM guest virtual machine is permitted to connect to all other z/VM guest virtual machines. The total number of connections for your z/VM guest virtual machine cannot exceed 100.

```
IUCV ALLOW MSGLIMIT 10000
IUCV ANY
OPTION MAXCONN 100
```

### Loading the IUCV modules

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 23. SMC protocol support

The shared memory communication (SMC) protocol is an addition to TCP/IP and can be used transparently for shared memory communications.

The SMC protocol can be used for connections through:

• Shared Memory Communications through RDMA (SMC-R) with RoCE devices.
• Shared Memory Communications Direct (SMC-D) with ISM devices

If both variants are available for a connection, SMC-D is used.

For more information about the SMC protocol, see https://ibm.biz/BdYAWW and https://ibm.biz/BdYucc.

Prerequisites

SMC connections are initiated through TCP/IP. Hence, the communication partners must be able to reach each other through TCP/IP. Also, SMC is not routable and thus both communication partners must be in the same IP subnet.

An SMC connection requires both communication partners to support SMC. Unless both partners support SMC, the connection falls back to TCP/IP.

The SMC-R protocol requires:

• A system with a network adapter with RDMA-over-Converged-Ethernet (RoCE) capability, for example a Mellanox Connect HCA. See Chapter 24, “RDMA over Converged Ethernet,” on page 353.

The SMC-D protocol requires:

• A system with an Internal Shared Memory (ISM) device. For more information about ISM devices, see Chapter 25, “Internal shared memory device driver,” on page 357.
• The communication partners must be running on the same CPC.

To use SMC on Linux, a socket application must use the AF_SMC address family. For AF_SMC support in existing applications without code changes, the SMC-Tools package provides a preload library and the smc_run command. For more information about these tools and how to convert socket applications from AF_INET or AF_INET6 to AF_SMC, see “Setting up the SMC support” on page 350.

Features

The AF_SMC address family provides DMA communication through remote or internal shared memory. Benefits include:

• Transparency to existing TCP/IP applications with the preload library and smc_run.
• Low latency
• Lower CPU usage compared to native TCP/IP

Building a kernel with SMC support

Control the build options for SMC 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 115 on page 350 summarizes the kernel configuration menu options that you must select in the Linux configuration menu to include SMC support.
Networking support --->
...
Networking options --->
...
SMC socket protocol family (CONFIG_SMC)
SMC: socket monitoring interface (CONFIG_SMC_DIAG)

Figure 115: SMC kernel configuration menu option

**CONFIG_SMC**
This option provides SMC protocol support. It can be compiled into the kernel or as a separate module, smc.

**CONFIG_SMC_DIAG**
Builds a kernel module smc_diag if active. smc_diag is a prerequisite for the statistics tool smcss (see “smcss - Display information about the AF_SMC sockets and link groups” on page 716).

### Setting up the SMC support

SMC traffic requires a mapping of the OSA or HiperSockets network interfaces to the RoCE adapters or ISM devices.

Physical network (PNET) IDs provide this mapping. If a network interface and an RoCE or ISM device have the same PNET ID, they are connected to the same physical network and can be used together for SMC. Assign PNET IDs to OSA, HiperSockets, RoCE, and ISM devices through the IOCDS. For more information about IOCDS, see z/OS HCD User's Guide, SC34-2669.

Figure 116 on page 350 illustrates how the IOCDS assigns the PNETID NET1 to an ISM device and a network interface for an Ethernet device. In Linux, the matching PNETID associates the ISM device with an Ethernet device.

As a fallback solution, SMC-R supports a software PNET table that maps network interfaces to RoCE adapters. Use the smc_pnet command to create a physical network (PNET) table with this mapping. See “smc_pnet - Create network mapping table” on page 713. SMC-D does not support such a software table.

**Note:** z/OS does not support the Mellanox card as an Ethernet device, and therefore uses OSA adapters for the initial handshake for SMC-R connections. Linux has no such constraint.
**Network device settings**

On the network device that is associated with the RoCE device you want to use for SMC traffic, check the settings with the `ethtool` command and ensure that pause settings are turned on.

For example, if `enP2s13` is the network device that is associated with the wanted device port:

```
# ethtool -a enP2s13
Pause parameters for enP2s13:
  Autonegotiate: off
  RX: on
  TX: on
```

**Sysctl settings**

SMC requires contiguous memory. The minimum is 16 KB, and the maximum is 512 MB. The SMC implementation selects a value as follows:

- Some socket applications define the socket send- and receive buffer sizes with a `setsockopt` call, whose upper limits are defined in `net.core.wmem_max` and `net.core.rmem_max`.
- If `setsockopt SO_SNDBUF` is not used, the socket send buffer size is taken from the value of `net.ipv4.tcp_wmem`.
- If `setsockopt SO_RCVBUF` is not used, the socket receive buffer is taken from the value of `net.ipv4.tcp_rmem`, rounded to the next higher power of 2.

**Make an existing application use SMC**

Use the preload library to make the unmodified socket application use SMC. Existing TCP/IP applications can benefit from the SMC protocol without recompiling, if they are invoked with the SMC preload library `ld_pre_smc.so`. See the `smc-tools` package for the `smc_run` script (see “smc_run - Run a TCP socket program with the SMC protocol using a preloaded library” on page 715), which makes an existing TCP/IP socket program use SMC.

**Converting an application to use SMC**

Alternatively, if you need to, you can convert an application. To convert an application from TCP/IP to SMC sockets, change the `socket()` function call from `AF_INET` to `AF_SMC` with protocol "0" and from `AF_INET6` to `AF_SMC` with protocol "1". For example, change:

```c
sd = socket(AF_INET, SOCK_STREAM, 0);
```

to:

```c
sd = socket(AF_SMC, SOCK_STREAM, 0);
```

and

```c
sd = socket(AF_INET6, SOCK_STREAM, 0);
```

to:

```c
sd = socket(AF_SMC, SOCK_STREAM, 1);
```

Use the `sockets.h` header file from the `glibc-header` package. For more programming information, see the `af_smc(7)` man page.
Investigating PNET IDs

You can find the PNET IDs for PCIe devices and for CCW group devices in sysfs.

**PCle devices**

The PNET ID of PCI devices can be read, in EBCDIC format, as the value of the `util_string` attribute of the device in sysfs. You can use a command of the following form to read a PNET ID and convert it to ASCII:

```
# cat /sys/devices/pci<function_name>/<function_address>/util_string | iconv -f IBM-1047 -t ASCII
```

In the command, `/sys/devices/pci<function_name>/<function_address>` represents the PCI device in sysfs.

**Example:**

```
# cat /sys/devices/pci0000:00/0000:00:00.0/util_string | iconv -f IBM-1047 -t ASCII
```

The PNET ID of the example is `NET1`. If there is no command output or if the output is blank, no PNET ID is assigned to the device.

**CCW group devices**

The PNET ID of CCW group devices can be read, in EBCDIC format, as the value of the `util_string` of the corresponding channel path ID in sysfs. To find the channel path ID of a CCW group device, read its `chpid` attribute in sysfs.

**Example:**

```
# cat /sys/bus/ccwgroup/devices/0.0.b1f0/chpid
```

To find the PNET ID issue a command of this form:

```
# cat /sys/devices/css0/chp0.<chpid>/util_string | iconv -f IBM-1047 -t ASCII
```

where `<chpid>` is the channel path ID.

**Example:**

```
# cat /sys/devices/css0/chp0.4a/util_string | iconv -f IBM-1047 -t ASCII
```

The PNET ID of the example is `NET1`. If there is no command output or if the output is blank, no PNET ID is assigned to the device.

**Tips**

- The output of the `iconv` command does not have a trailing line break, so displayed PNET IDs are followed by a command prompt. Pipe the output to a suitable `sed` command, for example `sed 's/$/\n/'`, to display the PNET IDs on a separate line.
- Use the following command to display a list of all CCW devices and their PNET IDs:

```
# for device in `ls -1 /sys/bus/ccwgroup/devices`; do
    chpid=`cat /sys/bus/ccwgroup/devices/$device/chpid | tr [A-F] [a-f]`;  
    pnetid="`cat /sys/devices/css0/chp0.$chpid/util_string | iconv -f IBM-1047 -t ASCII | sed 's/^/ /'";  
    echo " device: $device chpid: $chpid pnetID: $pnetid";  
done
```
Chapter 24. RDMA over Converged Ethernet

Linux on 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 or Mellanox ConnectX-4 adapter. The adapters are two-port Ethernet adapters. On a mainframe, the mapping of ports to function keys depend on the adapter hardware:

- The two Mellanox ConnectX-3 EN adapter ports belong to the same function ID.
- The two Mellanox ConnectX-4 adapter ports belong to different function IDs.

The RoCE support requires PCI Express support, see Chapter 38, “PCI Express support,” on page 427.

Using a RoCE device for SMC-R

SMC-R requires RoCE devices that are associated with network devices of TCP networks through a PNET ID, for example through statements in the IOCDS.

The following figure illustrates how a RoCE device and a Ethernet device are associated by a matching PNET ID. A communication peer has a similarly associated pair of an RoCE device and Ethernet device. With this setup, the TCP connection can switch over to an SMC-R connection over the SMC protocol. The communication peer can but need not be on the same CPC.

![Figure 117: A matching PNET ID associates RoCE devices and Ethernet device](image)

For more information about PNET IDs, see “Setting up the SMC support” on page 350.

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 118 on page 354 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)
Mellanox Connect-IB HCA support ---> (CONFIG_MLX5_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)

Mellanox Technologies ConnectX-4 and Connect-IB core driver (CONFIG_MLX5_CORE)

Networking support ---> (CONFIG_NET)

Networking options

SMC socket protocol family (CONFIG_SMC)
  SMC: socket monitoring interface (CONFIG_SMC_DIAG)

The RDS Protocol
  RDS over Infiniband and iWARP (CONFIG_RDS_RDMA)
  RDS over TCP (CONFIG_RDS_TCP)
  RDS debugging messages (CONFIG_RDS_DEBUG)

Figure 118: RoCE kernel configuration menu options

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 428 and “Using PCIe hotplug” on page 428.

2. Use the appropriate Mellanox device driver:
   - To use TCP/IP, you need the mlx4_core and mlx4_en modules, or you need the mlx5_core module.
   - To also use RDMA with InfiniBand, you further need the mlx4_ib or mlx5_ib module. You can use SMC sockets or reliable datagram sockets (RDS).
     - For SMC, the SMC protocol support must be in place, see Chapter 23, “SMC protocol support,” on page 349.
     - For RDS, you need the rds module and the rds_rdma module, see Documentation/networking/rds.txt in the Linux source tree and the rds and rds-rdma man pages.

Load any modules that are not compiled into the kernel or already loaded, for example, with modprobe.

3. Activate the network interface.
   You need to know the network interface name, which you can find under:
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. Note that the numbering of network device ports start with 0, but the numbering of InfiniBand device ports start with 1. For example:

```bash
# cat /sys/class/infiniband/mlx4_0/ports/
1/ 2/
```

**What to do next**
For further information about Mellanox, see:


---

## 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:

```bash
echo 1 > /sys/module/mlx4_core/parameters/debug_level
```
Chapter 25. Internal shared memory device driver

The internal shared memory (ISM) device driver provides virtual PCI devices for shared memory communications direct (SMC-D).

ISM devices are defined in the IOCDS. Each ISM definition includes a physical network ID (PNET ID) to associate the ISM device with Ethernet devices.

The following figure illustrates how an ISM device and a HiperSockets device are associated by a matching PNET ID. A communication peer on the same CPC has a similarly associated pair of an ISM device and HiperSockets device. With this setup, the TCP connection can switch over to an SMC-D connection over the SMC protocol.

![Figure 119: A matching PNET ID associates ISM devices and Ethernet devices](image)

For information about how to find the PNET ID of PCI devices from your Linux instance, see “Investigating PNET IDs” on page 352.

For more information on SMC and SMC-D, see Chapter 23, “SMC protocol support,” on page 349.

Building a kernel with the ISM device driver

You need to select the kernel configuration option CONFIG_ISM to be able to use ISM 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.

Figure 120 on page 358 shows the CONFIG_ISM option in the kernel configuration menu.
The ISM device driver can be compiled into the kernel or as a separate module, `ism`. The `CONFIG_ISM` option depends on the following common code options:

- `CONFIG_SMC`, see “Building a kernel with SMC support” on page 349.
- `CONFIG_PCI`, see “Building a kernel with the PCIe support” on page 427.

### Loading the ISM device driver

If the ISM device driver is compiled as a separate module, you must load it before you can use ISM devices.

Load the `ism` module with the `modprobe` command. The `ism` module has no module parameters.

```
# modprobe ism
```

### Listing ISM devices

Because ISM devices are PCI devices, you can list them with the `lspci` command.

**Example**

```
# lspci -v
0001:00:00.0 Non-VGA unclassified device: IBM Internal Shared Memory (ISM) virtual PCI device
   Physical Slot: 000002e1
   Flags: bus master, fast devsel, latency 0, IRQ 8
   Memory at 8001000000000000 (64-bit, prefetchable) [size=256T]
   Memory at 8002000000000000 (64-bit, prefetchable) [size=256]
   Capabilities: [40] MSI: Enable+ Count=1/32 Maskable- 64bit+
   Kernel driver in use: ism
   Kernel modules: ism
```
Part 5. System resources

These device drivers and features help you to manage the resources of your real or virtual hardware.

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

Restrictions
For prerequisites and restrictions see
Chapter 26. Managing CPUs

You can read CPU capability, activate standby CPUs, and examine the CPU topology.

Use the `lscpu` and `chcpu` commands to manage CPUs. These commands are part of the util-linux package. For details, see the man pages. Alternatively, you can manage CPUs through the attributes of their entries in sysfs.

Some attributes that govern CPUs are available in sysfs under:

```
/sys/devices/system/cpu/cpu<N>
```

where `<N>` is the number of the logical CPU. Both the sysfs interface and the `lscpu` and `chcpu` commands manage CPUs through their logical representation in Linux.

You can obtain a mapping of logical CPU numbers to physical CPU addresses by issuing the `lscpu` command with the `-e` option.

**Example:**

```
# lscpu -e
CPU NODE DRAWER BOOK SOCKET CORE L1d:L1i:L2d:L2i ONLINE CONFIGURED POLARIZATION ADDRESS
0   1    0      0    0      0    0:0:0:0         yes    yes        horizontal   0
1   1    0      0    0      0    1:1:1:1         yes    yes        horizontal   1
2   1    0      0    0      1    2:2:2:2         yes    yes        horizontal   2
3   1    0      0    0      1    3:3:3:3         yes    yes        horizontal   3
4   1    0      0    0      2    4:4:4:4         yes    yes        horizontal   4
5   1    0      0    0      2    5:5:5:5         yes    yes        horizontal   5
6   1    0      0    0      3    6:6:6:6         yes    yes        horizontal   6
7   1    0      0    0      3    7:7:7:7         yes    yes        horizontal   7
8   0    1      1    1      4    8:8:8:8         yes    yes        horizontal   8
...
```

The logical CPU numbers are shown in the CPU column and the physical address in the ADDRESS column of the output table.

Alternatively, you can find the physical address of a CPU in the sysfs `address` attribute of a logical CPU.

**Example:**

```
# cat /sys/devices/system/cpu/cpu0/address
0
```

**Simultaneous multithreading**

Linux in LPAR mode can use the simultaneous multithreading technology on mainframes.

IBM z13 introduced the simultaneous multithreading technology to the mainframe. In Linux terminology, simultaneous multithreading is also known as SMT or Hyper-Threading.

With multithreading enabled, a single core on the hardware is mapped to multiple logical CPUs on Linux. Thus, multiple threads can issue instructions to a core simultaneously during each cycle.

To find out whether multithreading is enabled for a particular Linux instance, compare the number of cores with the number of threads that are available in the LPAR. You can use the `hyptop` command to obtain this information.

Simultaneous multithreading is designed to enhance performance. Whether this goal is achieved strongly depends on the available resources, the workload, and the applications that run on a particular Linux instance. Depending on these conditions, it might be advantageous to not make full use of multithreading or to disable it completely. Use the `hyptop` command to obtain utilization data for threads while Linux runs with multithreading enabled.
You can use the smt= and nosmt kernel parameters to control multithreading. By default, Linux in LPAR mode uses multithreading if it is provided by the hardware.

**CPU capability change**

When the CPUs of a mainframe heat or cool, the Linux kernel generates a uevent for all affected online CPUs.

You can read the CPU capability from the Capability and, if present, Secondary Capability fields in `/proc/sysinfo`.

The capability 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.

**Changing the configuration state of CPUs**

A CPU on an LPAR can be in a configured, standby, or reserved state. You can change the state of standby CPUs to configured state and vice versa.

**Before you begin**

- You can change the configuration state of CPUs for Linux in LPAR mode only. For Linux on z/VM, CPUs are always in a configured state.
- Daemon processes like cpuplugd can change the state of any CPU at any time. Such changes can interfere with manual changes.

**About this task**

When Linux is booted, only CPUs that are in a configured state are brought online and used. The kernel does not detect CPUs in reserved state.

**Procedure**

Issue a command of this form to change the configuration state of a CPU:

```
# chcpu -c|-g <N>
```

where

<

is the number of the logical CPU.

- `c` changes the configuration state of a CPU from standby to configured.
- `g` changes the configuration state of a CPU from configured to standby. Only offline CPUs can be changed to the standby state.

Alternatively, you can write 1 to the configure sysfs attribute of a CPU to set its configuration state to configured, or 0 to change its configuration state to standby.

**Examples:**

- The following chcpu command changes the state of the logical CPU with number 2 from standby to configured:

```
# chcpu -c 2
```
The following command achieves the same results by writing 1 to the configure sysfs attribute of the CPU.

```bash
# echo 1 > /sys/devices/system/cpu/cpu2/configure
```

- The following chcpu command changes the state of the logical CPU with number 2 from configured to standby:

```bash
# chcpu -g 2
```

The following command achieves the same results by writing 0 to the configure sysfs attribute of the CPU.

```bash
# echo 0 > /sys/devices/system/cpu/cpu2/configure
```

### Setting CPUs online or offline

Use the chcpu command or the online sysfs attribute of a logical CPU to set a CPU online or offline.

**Before you begin**

- Daemon processes like cpuplugd can change the state of any CPU at any time. Such changes can interfere with manual changes.

**Procedure**

1. Optional: Rescan the CPUs to ensure that Linux has a current list of configured CPUs.
   
   To initiate a rescan, issue the chcpu command with the -r option.

   ```bash
   # chcpu -r
   ```
   
   Alternatively, you can write 1 to /sys/devices/system/cpu/rescan.

   You might need a rescan for Linux on z/VM after one or more CPUs have been added to the z/VM guest virtual machine by the z/VM hypervisor. Linux in LPAR mode automatically detects newly available CPUs.

2. Change the online state of a CPU by issuing a command of this form:

   ```bash
   # chcpu -e|-d <N>
   ```

   where

   `<N>`
   
   is the number of the logical CPU.

   - **-e**
     
     sets an offline CPU online. Only CPUs that are in the configuration state configured can be set online. For Linux on z/VM, all CPUs are in the configured state.

   - **-d**
     
     sets an online CPU offline.

   Alternatively, you can write 1 to the online sysfs attribute of a CPU to set it online, or 0 to set it offline.

**Examples:**
• The following chcpu commands force a CPU rescan, and then set the logical CPU with number 2 online.

```bash
# chcpu -r
# chcpu -e 2
```

The following commands achieve the same results by writing 1 to the online sysfs attribute of the CPU.

```bash
# echo 1 > /sys/devices/system/cpu/rescan
# echo 1 > /sys/devices/system/cpu/cpu2/online
```

• The following chcpu command sets the logical CPU with number 2 offline.

```bash
# chcpu -d 2
```

The following command achieves the same results by writing 0 to the online sysfs attribute of the CPU.

```bash
# echo 0 > /sys/devices/system/cpu/cpu2/online
```

**Examining the CPU topology**

Depending on your hardware support, sysfs provides information about the CPU topology of an LPAR.

**Before you begin**

Meaningful CPU topology information is available only to Linux in LPAR mode.

**About this task**

The Linux scheduler uses this topology information to optimize decisions about which process to schedule to which CPU. Depending on the workload, this optimization might increase cache hits and, therefore, overall performance.

By default, CPU topology support is enabled and default assumptions are used if no topology information is available. You can override these defaults if they are not suitable for your workload, see “Overriding topology default assumptions” on page 365.

The following sysfs attributes provide information about the CPU topology:

```bash
/sys/devices/system/cpu/cpu<N>/topology/thread_siblings
/sys/devices/system/cpu/cpu<N>/topology/core_siblings
/sys/devices/system/cpu/cpu<N>/topology/book_siblings
/sys/devices/system/cpu/cpu<N>/topology/drawer_siblings
```

where `<N>` specifies a particular logical CPU number. These attributes contain masks that specify sets of CPUs.

Because the mainframe hardware is evolving over time, the terms *drawer*, *book*, *core*, and *thread* do not necessarily correspond to fixed hardware entities. What matters for the Linux scheduler is the levels of relatedness that these terms signify, not the physical embodiment of the levels. In this context, more closely related means sharing more resources, like caches.

The thread_siblings, core_siblings, book_siblings, and drawer_siblings attribute each contain a mask that specifies the CPU and its peers at a particular level of relatedness.

1. The thread_siblings attribute covers the CPU and its closely related peers.
2. The core_siblings attribute covers all CPUs of the thread_siblings attribute and peers related at the core level.
3. The book_siblings attribute covers all CPUs of the core_siblings attribute and peers related at the book level.

4. The drawer_siblings attribute covers all CPUs of the book_siblings attribute and peers related at the drawer level.

If a machine reconfiguration causes the CPU topology to change, change uevents are created for each online CPU.

If the kernel also supports standby CPU activation and deactivation (see “Changing the configuration state of CPUs” on page 362), the masks also contains the CPUs that are in a configured, but offline state. Updating the masks after a reconfiguration might take some time.

**Overriding topology default assumptions**

Use the topology= kernel parameter to override default assumptions about the CPU topology.

By default, CPU topology support is enabled in the Linux kernel. If no topology information is available, a topology is assumed where the CPUs share least resources, that is, each CPU is assumed to be on a different drawer.

Use the topology=off kernel parameter to override these defaults if they are not suitable for your workload.

**Format**

```
| topology = off |
| topology = on |
```

*Figure 121: topology= syntax*

**off**

Disables CPU topology support.

**on**

If no topology information is available, assumes that all CPUs are core siblings. If topology information is available, this setting is ignored.

**Example**

```
topology=off
```

**Dynamic changes**

On a running Linux instance, you can read the current topology setting from `/proc/sys/s390/topology`. You can also write to `/proc/sys/s390/topology` to change the setting.

The value 1 corresponds to topology=on, and 0 corresponds to topology=off.

**Example**

```
# echo 0 > /proc/sys/s390/topology
```
CPU polarization

You can optimize the operation of a vertical SMP environment by adjusting the SMP factor based on the workload demands.

Before you begin
CPU polarization is relevant only to Linux in LPAR mode.

About this task
Horizontal CPU polarization means that the PR/SM™ hypervisor dispatches each virtual CPU of an LPAR for the same amount of time.

With vertical CPU polarization, the PR/SM hypervisor dispatches certain CPUs for a longer time than others. For example, if an LPAR 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. This CPU would run most of the time while the other two CPUs would get nearly no 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: Switching to vertical CPU polarization usually results in a system with different types of vertical CPUs. Running a system with different types of vertical CPUs can result in significant performance regressions. If possible, use only one type of vertical CPUs. Set all other CPUs offline and deconfigure them.

Procedure
To change the polarization, issue a command of this form:

```
# chcpu -p horizontal|vertical
```

Alternatively, you can write a 0 for horizontal polarization (the default) or a 1 for vertical polarization to `/sys/devices/system/cpu/dispatching`.

Example: The following chcpu command sets the polarization to vertical.

```
# chcpu -p vertical
```

You can achieve the same results by issuing the following command:

```
# echo 1 > /sys/devices/system/cpu/dispatching
```

What to do next
You can issue the lscpu command with the -e option to find out the polarization of your CPUs. For more detailed information for a particular CPU, read the polarization attribute of the CPU in sysfs.

```
# cat /sys/devices/system/cpu/cpu<N>/polarization
```

The polarization can have one of the following values:

- `horizontal` - each of the guests' virtual CPUs is dispatched for the same amount of time.
- `vertical:high` - full CPU time is allocated.
- `vertical:medium` - medium CPU time is allocated.
- `vertical:low` - very little CPU time is allocated.
- `unknown` - temporary value following a polarization change until the change is completed and the kernel has established the new polarization of each CPU.
Chapter 27. NUMA emulation

The NUMA emulation on Linux on Z distributes the available memory to logical NUMA nodes without using topology information about the physical memory.

Linux maintains separate memory management structures for each node. Especially on large systems, this separation can improve the overall system performance, or latency, or both.

What you should know about NUMA emulation

The NUMA emulation distributes memory and CPU resources among NUMA nodes.

Memory distribution and stripe size

The NUMA emulation splits the usable system memory into stripes of a fixed size.

These memory stripes are then distributed, in round-robin mode, among the NUMA nodes. You can configure the number of NUMA nodes and the stripe size through kernel parameters (see “Configuring NUMA emulation” on page 368).

The difference between nodes in assigned memory cannot exceed the stripe size, so configuring small stripes leads to a balanced distribution. However, the stripes must not be too small, otherwise failing memory allocations prevent the kernel from booting. The minimum stripe size depends on the maximum number of CPUs (CONFIG_NR_CPUS) for which the kernel is compiled. For example, 2 CPUs require a minimum size of about 4 MB and 256 CPUs require about 512 MB.

Another approach to achieving a balanced memory distribution is to configure large stripes, such that exactly one stripe is assigned to each NUMA node.

CPU assignment to NUMA nodes

The Linux scheduler requires a stable mapping of CPUs to NUMA nodes. Therefore, cores are pinned to NUMA nodes when one of their CPUs is set online for the first time.

As a consequence, a CPU that is set offline is always assigned to its previous NUMA node when it is set back online. With multithreading enabled, a CPU is equivalent to a thread (see “Simultaneous multithreading” on page 361).

Pinned cores are distributed evenly across the NUMA nodes. You can distort this initial balance by setting a disproportionate number of CPUs from a particular NUMA node offline. New CPUs are assigned according to the number of pinned cores, not according to the number of online CPUs.

For example, assume a node A that has two cores and with one of four CPUs (threads) online. Further, assume a node B that has one core but two CPUs online. Because node B has fewer cores than node A, a newly configured CPU that is set online is assigned to node B, and the corresponding core is pinned to node B.

Note: Do not use NUMA emulation with cpuplugd. The cpuplugd daemon can distort the balance of CPU assignment to NUMA nodes. Issue the following command to find out if cpuplugd is running:

```bash
# service cpuplugd status
```

See also “cpuplugd - Control CPUs and memory” on page 606.
Building a kernel with NUMA support

Support for emulated NUMA depends on several kernel configuration options.

**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 following figure shows how to find the z/Architecture specific kernel configuration menu options for NUMA emulation.

```
Processor type and features --->
...---
NUMA support (common code option CONFIG_NUMA)
  [4] Maximum NUMA nodes (as a power of 2) (common code option CONFIG_NODES_SHIFT)
  Select NUMA modes (depends on NUMA) --->
  NUMA emulation (CONFIG_NUMA_EMU)
  NUMA emulation memory chunk size (CONFIG_EMU_SIZE)
```

Figure 122: Kernel configuration menu options for NUMA emulation

---

**CONFIG_NUMA_EMU**
Enables the NUMA emulation mode.

**CONFIG_EMU_SIZE**
Sets the default size for the stripes into which the memory is split for distribution to the emulated NUMA nodes.

You also need the following common code options:
- CONFIG_NUMA
- CONFIG_NODES_SHIFT
- CONFIG_SCHED_TOPOLOGY

Optionally, you can also select CONFIG_NUMA_BALANCING, but do not select CONFIG_NUMA_BALANCING_DEFAULT_ENABLED.

---

Configuring NUMA emulation

You configure NUMA emulation through kernel parameters.

```
numa=emu
emu_nodes = 1
emu_nodes = <number>
emu_size = <stripe_size>
numa_balancing = disable
numa_balancing = enable
numa_debug
sched_debug
```

Notes:

1. The default stripe size is set through the CONFIG_EMU_SIZE kernel configuration parameter when the kernel is compiled.
2. Do not enable NUMA balancing.

Figure 123: NUMA emulation kernel parameter syntax
where:

**numa=emu**

Sets the NUMA emulation mode and enables NUMA for the Linux instance.

**emu_nodes=<number>**

Specifies the number of NUMA nodes to be emulated. The default is 1. Emulating only one NUMA node, in effect, disables NUMA.

**emu_size=<stripe_size>**

Specifies the memory stripe size in byte. You can use the k, M, G, and T suffixes. If this parameter is omitted NUMA uses a default value that is set when the kernel is compiled, see “Building a kernel with NUMA support” on page 368.

The memory stripe size must be a multiple of the memory block size (see “Finding out the memory block size” on page 373).

For other considerations about setting the stripe size see “Memory distribution and stripe size” on page 367.

**numa_balancing**

Do not enable NUMA balancing, see also “Building a kernel with NUMA support” on page 368.

**numa_debug**

Enables kernel debug messages for the NUMA emulation on IBM Z.

**sched_debug**

Enables scheduler kernel debug messages.

**Example**

```
numa=emu  emu_nodes=4  emu_size=1G
```
Chapter 28. 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.

Hotplug memory management overhead

Linux requires 64 bytes of memory to manage a 4-KB page of hotplug memory.

Use the following formula to calculate the total amount of initial memory that is consumed to manage your hotplug memory:

\[
\text{<hotplug memory> / 64}
\]

Example: 4.5 TB of hotplug memory consume 4.5 TB / 64 = 72 GB.

For large amounts of hotplug memory, you might have to increase the initial memory that is available to your Linux instance. Otherwise, booting Linux might fail with a kernel panic and a message that there is not enough free memory.

How memory is represented in sysfs

Both the core memory of a Linux instance and the available hotplug memory are represented by directories in sysfs.

The memory with which Linux is started is the core memory. On the running Linux system, additional memory can be added as hotplug memory. The Linux kernel requires core memory to allocate its own data structures.

In sysfs, both the core memory of a Linux instance and the available hotplug memory are represented in form of memory blocks of equal size. Each block is represented as a directory of the form /sys/devices/system/memory/memory<n>, where <n> is an integer. You can find out the block size by reading the /sys/devices/system/memory/block_size_bytes attribute.

In the naming scheme, the memory blocks with the lowest address ranges are assigned the lowest integer numbers. The core memory always begins with memory0. The hotplug memory blocks follow the core memory blocks.

You can calculate where the hotplug memory begins. To find the number of core memory blocks, divide the base memory by the block size.

Example:

• With a core memory of 512 MB and a block size of 128 MB, the core memory is represented by four blocks, memory0 through memory3. Therefore, first hotplug memory block on this Linux instance is memory4.

• Another Linux instance with a core memory of 1024 MB and access to the same hotplug memory, represents this first hotplug memory block as memory8.

The hotplug memory is available to all operating system instances within the z/VM system or LPAR to which it was defined. The state sysfs attribute of a memory block indicates whether the block is in use.
by your own Linux system. The state attribute does not indicate whether a block is in use by another operating system instance. Attempts to add memory blocks that are already in use fail.

**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.

**Memory zones**

The Linux kernel divides memory into memory zones. On a mainframe, three zones are used: DMA, Normal, and Movable.

- Memory in the DMA zone is below 2 GB, and some I/O operations require that memory buffers are located in this zone.
- Memory in the Normal zone is above 2 GB, and it can be used for all memory allocations that do not require zone DMA.
- Memory in the Movable zone cannot be used for arbitrary kernel allocations, but only for memory buffers that can easily be moved by the kernel, such as user memory allocations and page cache memory. Memory in the Movable zone can more easily be taken offline than memory in other zones.

The zones that are available to a memory block are listed in the `valid_zones` sysfs attribute. For more information, see “Adding memory” on page 374.

**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 124 on page 372 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 124: 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.

Always align the z/VM guest storage with the Linux memory block size. Otherwise, memory blocks might be missing or impossible to set offline in Linux.

For information about defining standby memory for z/VM guests see the "DEFINE STORAGE" section in z/VM: CP Commands and Utilities Reference, SC24-6268.

Performing memory management tasks

Typical memory management tasks include finding out the memory block size, adding memory, and removing memory.

• “Finding out the memory block size” on page 373
• “Listing the available memory blocks” on page 374
• “Adding memory” on page 374
• “Removing memory” on page 375

Finding out the memory block size

On an IBM Z mainframe, memory is provided to Linux as memory blocks of equal size.

Procedure

• Use the lsmem command to find out the size of your memory blocks.

Example:

```
# lsmem

Address range                  Size (MB)  State  Removable  Device
================================================================================
0x0000000000000000-0xffffffff  256  online  no         0
0x0000000000000000-0xffffffff  512  online  yes        1-2
0x0000000000000000-0xffffffff  256  online  no         3
0x0000000000000000-0xffffffff  768  online  yes        4-6
0x0000000000000000-0xffffffff  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:

```bash
# 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:

```bash
# lsmem -a
```

<table>
<thead>
<tr>
<th>Address range</th>
<th>Size (MB)</th>
<th>State</th>
<th>Removable</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000000000000000-0x000000000fffffff</td>
<td>256</td>
<td>online</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>0x0000000010000000-0x000000001fffffff</td>
<td>256</td>
<td>online</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>0x0000000020000000-0x000000002fffffff</td>
<td>256</td>
<td>online</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>0x0000000030000000-0x000000003fffffff</td>
<td>256</td>
<td>online</td>
<td>yes</td>
<td>3</td>
</tr>
<tr>
<td>0x0000000040000000-0x000000004fffffff</td>
<td>256</td>
<td>online</td>
<td>yes</td>
<td>4</td>
</tr>
<tr>
<td>0x0000000050000000-0x000000005fffffff</td>
<td>256</td>
<td>offline</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>0x0000000060000000-0x000000006fffffff</td>
<td>256</td>
<td>offline</td>
<td>-</td>
<td>6</td>
</tr>
<tr>
<td>0x0000000070000000-0x000000007fffffff</td>
<td>256</td>
<td>offline</td>
<td>-</td>
<td>7</td>
</tr>
</tbody>
</table>

- Memory device size : 256 MB
- Memory block size   : 256 MB
- Total online memory : 1280 MB
- Total offline memory: 786 MB

- Alternatively, you can list the available memory blocks by listing the contents of `/sys/devices/system/memory`. Read the state attributes of each memory block to find out whether it is online or offline.

#### Example: The following command results in an overview for all available memory blocks.

```bash
# 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. You can chose a memory zone for certain memory blocks.

#### 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 8, “Suspending and resuming Linux,” on page 99 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.
- Alternatively, you can write to the `state` sysfs attribute of an unused memory block.
  Issue a command of the form:

```
# echo online_value > /sys/devices/system/memory/memory<n>/state
```

where `online_value` is one of:

- **online**
  sets the memory block online to the default zone. The default zone is the first zone listed in the `valid_zones` sysfs attribute.

- **online_movable**
  sets the memory block online to the Movable zone. Setting the block online fails if the Movable zone is not listed in the `valid_zones` sysfs attribute.

- **online_kernel**
  sets the memory block online to the first non-Movable zone listed in the `valid_zones` directory. Setting the block online fails if the Movable zone is the only zone listed in the `valid_zones` sysfs attribute.

<n> is an integer that identifies the memory unit.

Results

Adding the memory block fails if the memory block is already in use. The `state` attribute changes to `online` when the memory block has been added successfully.

Removing memory

You can remove memory from your Linux instance by setting memory blocks offline.

About this task

Avoid removing core memory. The Linux kernel requires core memory to allocate its own data structures.

Procedure

- Use the `chmem` command with the `-d` parameter to set memory offline.
  You can specify the amount of memory you want to remove with the command without specifying particular memory blocks. The tool finds eligible memory blocks for you and sets the most suitable blocks offline.
- 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 29. Persistent device configuration

Use the chzdev command to persistently configure your devices and device drivers and the lszdev command to view your configuration.

Mainframe-specific devices, such as DASDs, FCP devices, and network devices require special configuration steps before they can be used. Tools are available that configure devices, for example chccwdev and znetconf, but this type of configuration is not preserved across reboots.

The chzdev command facilitates persistent configuration. The command performs all configuration steps that are required to make devices operational, for example, as a block device, a character device, or a network interface.

The following device types are supported:

- FICON-attached direct access storage devices (DASDs)
- SCSI-over-Fibre Channel (FCP) devices and SCSI devices
- OSA-Express and HiperSockets network devices
- Channel-to-channel (CTC) and CTC-MPC network devices
- LAN-Channel-Station (LCS) network devices
- Channel command word (CCW) devices that are not covered by any other device type, for example the 3215 console, 3270 terminal devices, z/VM reader and puncher devices, and CCW tape devices.

The chzdev and lszdev commands are included in the s390-tools package. chzdev configures the devices and device drivers in two ways:

- In the currently running configuration, called the active configuration
- In configuration files such as udev rules, called the persistent configuration

The lszdev command displays configuration information about devices and device drivers. For details, see “lszdev - Display IBM Z device configurations” on page 682.

For details about the chzdev command, see “chzdev - Configure IBM Z devices” on page 589.

Note: Distributions might use tools that are not aligned with lszdev and chzdev. Using such distribution tools alternating with chzdev can result in conflicting configuration settings.

Device ID

The chzdev and lszdev commands use device IDs to identify devices. For CCW devices and CCW group devices, this device ID is the device bus-ID.

The device bus-ID is of the format 0.<subchannel_set_ID>.<devno>, for example, 0.0.8000.

Tip: For device bus-IDs with a leading "0.0", you can shorten the specification to just the device number (devno). For example, you can shorten 0.0.0b10 to 0b10.

Other device categories can have different IDs. For example, SCSI devices have a triplet device ID consisting of a device number, a WWPN, and a LUN.

Configuring device drivers

You can use the chzdev command to modify device driver attributes, for example module parameters such as DASD's ee_pages. You select a device driver, rather than a device, by using the --type option. Device drivers can be selected by type or sub-type, for example DASDs are of type dasd, but have the sub-types dasd-fba and dasd-eckd.
chzdev syntax overview

Where the different command sections have these meanings:

**Device or device type selection**
Select devices by device ID, device state, or function. Select device types by specifying a device type and the --type option. For details about selecting devices or device types, see “Selecting devices and device drivers” on page 378.

**Actions**
Perform an action against the selected devices. For details about these actions, see:
- “Enabling and disabling a device” on page 381
- “Changing device or device driver settings” on page 386
- “Importing and exporting configuration data” on page 388

**Options**
Choose how to apply the command, for example as a test run, as applying to the persistent configuration only, or as running in quiet mode. Options include --dry-run, --verbose, --quiet, and --yes. For the complete list of options, see “chzdev - Configure IBM Z devices” on page 589

lszdev syntax overview

Where the different command sections have these meanings:

**Device or device type selection**
Select devices to display by device ID, device state, or function. For details about selecting devices, see “Selecting devices and device drivers” on page 378. If no selection is made, all existing and configured devices are displayed.

**Options**
Choose the configuration information, and how to display it.
- To display a list with information about all devices, specify only lszdev without options. You can restrict output to a single device, a device type, or a range of devices. You can control what information is included by specifying output columns.
- To display details about a single device, specify the device and the --info option.

For details on and examples of displaying information, see “Viewing the configuration” on page 383
For the complete list of options, see “lszdev - Display IBM Z device configurations” on page 682

Selecting devices and device drivers

You can select devices by type and ID, by state, by path, or by network interface name.

**About this task**
You can select either devices or device drivers.

To select a device driver, specify the --type option and a <type> without a device ID. For details, see the description of <type> and the --type option.
Select devices by device ID, by attribute, interface, node, or path. Then restrict your choice to the devices with a particular configuration state. For details see the descriptions for the --configured, --existing, --online, and --offline options. To select all devices, specify --all.

**Device driver selection:**

`<type>`

restricts the scope of an action to the specified device type. Specify a device type together with the --type option to manage the configuration of the device driver itself.

`-t` or `--type <device_type>`

selects a device type as target for a configuration or query action.

**Device selection:**

`<type>`

restricts the scope of an action to the specified device type. Specify a device type and optionally a device ID to only work on devices with matching type and ID.

**Note:**

As a precaution, use the most specific device type when configuring a device by ID. Otherwise the same device ID might accidentally match other devices of a different sub-type. To get a list of supported device types, use the --list-types option.

`<device>`

selects a single device or a range of devices by device ID. Separate multiple IDs or ranges with a comma (,). To select a range of devices, specify the ID of the first and the last device in the range separated by a hyphen (-).
--all
   selects all existing and configured devices.

--by-attrib <attrib=value> | <attrib!=value>
   selects devices with a specific attribute, <attrib> that has a value of <value>. When specified as
   <attrib>=<value>, selects all devices that do not provide an attribute named <attrib> with a value of
   <value>.

   Tip: You can use the --list-attributes option to display a list of available attributes and the --help-
   attribute to get more detailed information about a specific attribute.

--by-interface <interface>
   selects devices by network interface, for example, eth0. <interface> must be the name of an existing
   networking interface.

--by-node <device_node>
   selects devices by device node, for example, /dev/sda. <device_node> must be the path to the
device node for a block device or character device.

   Note: If <device_node> is the device node for a logical device (such as a device mapper device),
   lszdev tries to resolve the corresponding physical device nodes. The lsb1k tool must be available
   for this resolution to work.

--by-path <path>
   selects devices by file-system path, for example, /usr. The <path> parameter can be the mount point
   of a mounted file system, or a path on that file system.

   Note: If the file system that provides <path> is stored on multiple physical devices (such as supported
   by btrfs), lszdev tries to resolve the corresponding physical device nodes. The lsb1k tool must be
   available and the file system must provide a valid UUID for this resolution to work.

Restrict selection to configuration state:
--configured
   narrows the selection to those devices for which a persistent configuration exists.

--existing
   narrows the selection to all devices that are present in the active configuration.

--configured --existing
   specifying both --configured and --existing narrows the selection to devices that are present in both
   configurations, persistent and active.

--online
   narrows the selection to devices that are enabled in the active configuration.

--offline
   narrows the selection to devices that are disabled in the active configuration.

Examples
In the following examples the enable and disable actions are used when illustrating the different methods
of selecting devices.

• To enable device 0.0.8000 by type and ID, issue:

```
# chzdev dasd-fba 0.0.8000 --enable
FBA DASD 0.0.8000 configured
```

• To select a device by ID only, issue the following command to disable device 0.0.8000. In this case
chzdev automatically determines the associated type:

```
# chzdev 0.0.8000 --disable
FBA DASD 0.0.8000 deconfigured
```

Note: This addressing mode might result in multiple devices being selected. For example, if an FBA
DASD is part of the persistent configuration, but you have disabled it and defined a new ECKD DASD
0.0.8000 that is active, and you enable 0.0.8000, both the FBA and the ECKD device will be enabled. The same is true if you specify only "dasd" and not "dasd-fba".

**Tip:** To ensure that you enable the correct device, include the most specific type, for example dasd-eckd instead of just dasd.

- To select multiple devices by range, specify the beginning of the range, a hyphen, and the end of the range:

```bash
# chzdev dasd-eckd 0.0.3000-0.0.4000 --enable
```

```text
ECKD DASD 0.0.3718 configured
ECKD DASD 0.0.3719 configured
ECKD DASD 0.0.371a configured
ECKD DASD 0.0.37b8 configured
ECKD DASD 0.0.37b9 configured
ECKD DASD 0.0.37be configured
ECKD DASD 0.0.37bf configured
```

- To narrow the selection by configuration state, for example "online", use the --online option. This example configures all devices that are enabled in the active configuration persistently:

```bash
# chzdev --persistent --online --enable
```

```text
Configuring devices in the persistent configuration only
ECKD DASD 0.0.3718 configured
ECKD DASD 0.0.3719 configured
ECKD DASD 0.0.371a configured
ECKD DASD 0.0.37b8 configured
ECKD DASD 0.0.37b9 configured
ECKD DASD 0.0.37be configured
ECKD DASD 0.0.37bf configured
FBA DASD 0.0.8000 configured
FCP device 0.0.1940 configured
Note: NPIV mode disabled - LUNs must be configured manually
zFCP LUN 0.0.1940:0x500507630508c1ae:0x402140ac00000000 configured
QETH device 0.0.f5f0:0.0.f5f1:0.0.f5f2 configured
Generic CCW device 0.0.0009 configured
```

- To select devices by specifying a file system path that is located on the target device, use the --by-path option:

```bash
# chzdev --persistent --by-path /mnt --enable
```

```text
Configuring devices in the persistent configuration only
ECKD DASD 0.0.3718 configured
```

- To select a device by specifying a device node that refers to a block or character device:

```bash
# chzdev --persistent --by-node /dev/dasda1 --enable
```

```text
Configuring devices in the persistent configuration only
ECKD DASD 0.0.3718 configured
```

### Enabling and disabling a device

You can use the chzdev command to enable or disable a device, both in the active and in the persistent configuration.

**About this task**

Any steps necessary for the device to function are automatically taken. If a cio_ignore device exclusion list is active, chzdev removes the specified device ID from the list.

**Note:** If a SCSI device is enabled and its FCP device taken off the cio_ignore exclusion list, all devices defined to the FCP device might come online automatically, see "Configuring SCSI devices" on page 201.
where:

**-d or --disable**

Disables the selected devices.

**Active configuration**

Disables the selected devices by reverting the configuration steps necessary to enable them.

**Persistent configuration**

Removes configuration files and settings associated with the selected devices.

**-e or --enable**

Enables the selected devices. Any steps necessary for the devices to function are taken, for example: create a CCW group device, remove a device from the CIO exclusion list, or set a CCW device online.

**Active configuration**

Performs all setup steps required for a device to become operational, for example, as a block device or as a network interface.

**Persistent configuration**

Creates configuration files and settings associated with the selected devices.

**-a or --active**

Applies changes to the active configuration only. The persistent configuration is not changed unless you also specify --persistent.

**Note:** Changes to the active configuration are effective immediately. They are lost on reboot, when a device driver is unloaded, or when a device becomes unavailable.

**-p or --persistent**

Applies changes to the persistent configuration only. The persistent configuration takes effect when the system boots, when a device driver is loaded, or when a device becomes available.

**Examples**

- To enable an ECKD DASD with bus ID 0.0.3718 in both the active and the persistent configuration, specify the device ID and its type to the chzdev command, together with the --enable option:

  ```bash
  # chzdev dasd-eckd 0.0.3718 --enable
  ECKD DASD 0.0.3718 configured
  ```

  This command sets ECKD DASD 0.0.3718 online and creates udev rules to automatically bring it online at the next boot.

- To restrict the scope of the configuration actions to either the active or persistent configuration, specify --active or --persistent. The following command creates udev rules to automatically bring device 0.0.3718 online at the next boot:

  ```bash
  # chzdev dasd-eckd 0.0.3718 --enable --persistent
  Configuring devices in the persistent configuration only
  ECKD DASD 0.0.3718 configured
  ```

- To disable a device, specify its device type and ID together with the --disable option:

  ```bash
  # chzdev dasd-eckd 0.0.3718 --disable
  ECKD DASD 0.0.3718 deconfigured
  ```
• To enable an FCP device and create a persistent configuration, issue:

```bash
# chzdev --enable zfcp-lun 0.0.198d:0x50050763070bc5e3:0x4006404600000000
```

• To enable a FCP device without creating a persistent configuration, issue:

```bash
# chzdev --enable --active zfcp-lun 0.0.198d:0x50050763070bc5e3:0x4006404600000000
```

• To enable a QETH device and create a persistent configuration, issue:

```bash
# chzdev --enable qeth 0.0.a000:0.0.a001:0.0.a002
```

• To enable a QETH device without creating a persistent configuration, issue:

```bash
# chzdev --enable --active qeth 0.0.a000:0.0.a001:0.0.a002
```

### Viewing the configuration

Use the `lszdev` command to list information about device drivers, devices, device types, and attributes.

#### About this task

The `lszdev` command provides options for extracting information from the active and persistent system configuration. You can list and get help on available device attributes and list available device types.

```bash
lszdev
```

where:

- **-i or --info**
  
  displays detailed information about the configuration of the selected device or device type.

- **-c or --columns <columns>**
  
  specifies a comma-separated list of columns to display.
  
  Example:

  ```bash
  # lszdev --columns TYPE,ID
  ```

  Tip: To get a list of supported column names, use the `--list-columns` option.

- **-n or --no-headings**
  
  suppresses column headings for list output.

- **--pairs**
  
  produces output in `<key="value">` format. Use this option to generate output in a format more suitable for processing by other programs. In this format, column values are prefixed with the name of the
corresponding column. Values are enclosed in double quotation marks. The lszdev command automatically escapes quotation marks and slashes that are part of the value string.

--base <path> | <key=value>
changes file system paths that are used to access files. If <path> is specified without an equal sign (=), it is used as base path for accessing files in the active and persistent configuration. If the specified parameter is in <key=value> format, only those paths that begin with <key> are modified. For these paths, the initial <key> portion is replaced with <value>.

Example:

```
# lszdev --persistent --base /etc=/mnt/etc
```

Examples

- To list devices with their status, use the lszdev command, issue:

```
# lszdev
TYPE        ID                         ON  PERS  NAMES
dasd-eckd   0.0.0190                   no  no
dasd-eckd   0.0.3718                   yes yes   dasda dasda1
dasd-eckd   0.0.3719                   no  no
dasd-eckd   0.0.371a                   yes yes   dasdb dasdb1 dasdb2 dasdb3
dasd-fba    0.0.8000                   no  no
zfcp-host   0.0.1940                   no  no
zfcp-host   0.0.1941                   no  no
qeth        0.0.a000:0.0.a001:0.0.a002 no  no
qeth        0.0.a003:0.0.a004:0.0.a005 no  no
qeth        0.0.f500:0.0.f501:0.0.f502 no  no
qeth        0.0.f503:0.0.f504:0.0.f505 no  no
qeth        0.0.f5f0:0.0.f5f1:0.0.f5f2 yes yes   enccw0.0.f5f0
ctc         0.0.c000:0.0.c001          no  no
ctc         0.0.c002:0.0.c003          no  no
generic-ccw 0.0.0009                   yes no
generic-ccw 0.0.000c                   no  no
generic-ccw 0.0.000d                   no  no
generic-ccw 0.0.000e                   no  no
```

- To restrict output to a single device type, issue:

```
# lszdev qeth
TYPE        ID                          ON  PERS  NAMES
qeth        0.0.a000:0.0.a001:0.0.a002  no  no
qeth        0.0.a003:0.0.a004:0.0.a005  no  no
qeth        0.0.f500:0.0.f501:0.0.f502  no  no
qeth        0.0.f503:0.0.f504:0.0.f505  no  no
qeth        0.0.f5f0:0.0.f5f1:0.0.f5f2  yes yes   enccw0.0.f5f0
```

- To list output in machine-readable format, use the --pairs option, issue:

```
# lszdev qeth --pairs
TYPE="qeth" ID="0.0.a000:0.0.a001:0.0.a002" ON="no" PERS="no" NAMES=""
TYPE="qeth" ID="0.0.a003:0.0.a004:0.0.a005" ON="no" PERS="no" NAMES=""
TYPE="qeth" ID="0.0.f500:0.0.f501:0.0.f502" ON="no" PERS="no" NAMES=""
TYPE="qeth" ID="0.0.f503:0.0.f504:0.0.f505" ON="no" PERS="no" NAMES=""
TYPE="qeth" ID="0.0.f5f0:0.0.f5f1:0.0.f5f2" ON="yes" PERS="yes" NAMES="enccw0.0.f5f0"
```

- To list all columns that you can display, use the --list-columns option:
To display specific columns only, use the --columns option with the names of the columns as a comma-separated list:

```
# lszdev qeth --columns ID,NAMES,ATTR:layer2
ID       NAMES            ATTR:layer2
0.0.a000:0.0.a001:0.0.a002                  0
0.0.a003:0.0.a004:0.0.a005                  0
0.0.f500:0.0.f501:0.0.f502                  1
0.0.f503:0.0.f504:0.0.f505                  0
0.0.f5f0:0.0.f5f1:0.0.f5f2    enccw0.0.f5f0 1
```

To get a list of supported device types, use the --list-types option:

```
# lszdev --list-types
TYPE        DESCRIPTION
dasd        FICON-attached Direct Access Storage Devices (DASDs)
dasd-eckd   Enhanced Count Key Data (ECKD) DASDs
dasd-fba    Fixed Block Architecture (FBA) DASDs
zfcp        SCSI-over-Fibre Channel (FCP) devices and SCSI devices
zfcp-host   FCP devices
zfcp-lun    zfcp-attached SCSI devices
qeth        OSA-Express and HiperSockets network devices
ttc         Channel-To-Channel (CTC) and CTC-MPC network devices
lcs         LAN-Channel-Station (LCS) network devices
generic-ccw Generic Channel-Command-Word (CCW) devices
```

Some device types are related. For example, specifying the "dasd" device type will select both "dasd-eckd" and "dasd-fba" devices.

To list configuration details for a device driver, use the --type and a device-type name:

```
# lszdev --info --by-interface enccw0.0.f5f0
DEVICE qeth 0.0.f5f0:0.0.f5f1:0.0.f5f2
Names     : enccw0.0.f5f0
Modules    : qeth
Online     : yes
Exists     : yes
Persistent : yes

ATTRIBUTE            ACTIVE        PERSISTENT
buffer_count          "64"           -
hw_trap               "disarm"       -
isolation             "none"         -
layer2                "1"            "1"
online                "1"            "1"
performance_stats     "0"            -
portname              ""             -
portno                "0"            -
priority_queueing     "always queue 0" -
```

To list configuration details for a device driver, use the --type and a device-type name:
Changing device or device driver settings

Use the chzdev command to change both device and device driver settings.

About this task

Some devices provide named attributes. You can change the value of the attributes by using the chzdev command with a `<attribute=value>` assignment.

You can also use the chzdev command to modify device driver attributes, for example module parameters such as the DASD module parameter `eer_pages`. You select a device driver, rather than a device, by specifying the `--type` option. Device drivers can be selected by type or sub-type, for example DASDs are of type `dasd`, but have the sub-types `dasd-fba` and `dasd-eckd`.

where:

`<attribute=value>`
specifies a device attribute and its value. To specify multiple attributes, separate attribute-value pairs with a blank.

You can use the `--list-attributes` option to display a list of available attributes and the `--help-attribute` to get more detailed information about a specific attribute.

Tip: To specify an attribute that is not known to chzdev, use the `--force` option.

- `--remove <attrib>` removes the setting for attribute `<attrib>`.

  **Active configuration**
  For attributes that maintain a list of values, clears all values for that list.

  **Persistent configuration**
  Removes any setting for the specified attribute. When the device or device driver is configured again, the attribute is set to its default value.

  Some attributes cannot be removed.

- `--remove-all` removes the settings for all attributes of the selected device or device driver.
Active configuration
For attributes that maintain a list of values, clears all values for that list.

Persistent configuration
Removes all attribute settings that can be removed. When the device or device driver is configured again, the attribute is set to its default value.

Some attributes cannot be removed.

- **f** or --force
  overrides safety checks and confirmation questions, including:
  - More than 256 devices selected
  - Configuring unknown attributes
  - Combining apparently inconsistent settings

Examples for changing device settings

- To set the use_diag attribute for device 0.0.8000, issue:

  ```
  # chzdev dasd-fba 0.0.8000 use_diag=1 --persistent --verbose
  Configuring devices in the persistent configuration only
  FBA DASD 0.0.8000 configured
  Changes: use_diag=1
  Block devices: /dev/dasda /dev/dasda1
  ```

- To list available attributes, specify the device type together with the --list-attributes option:

  ```
  # chzdev dasd-fba --list-attributes
  NAME                        DESCRIPTION
  online                       Activate a device
  cmb_enable                   Enable the Channel measurement facility
  failfast                     Modify error recovery in no-path scenario
  readonly                     Inhibit write access to DASD
  erplog                       Enable logging of Error Recovery Processing
  use_diag                     Activate z/VM hypervisor assisted I/O processing
  eer_enabled                  Enable Extended Error Reporting
  expires                      Modify I/O operation timeout
  retries                      Modify I/O operation retry counter
  timeout                      Modify I/O request timeout
  reservation_policy           Modify lost device reservation behavior
  last_known_reservation_state Display and reset driver device reservation view
  safe_offline                 Deactivate DASD after processing outstanding I/Os
  ```

- To display more details about an attribute, use the --help-attribute option:

  ```
  # chzdev dasd-fba --help-attribute use_diag
  ATTRIBUTE use_diag
  DESCRIPTION
  Control I/O access mode for a DASD:
  0: I/O is performed using standard channel programs
  1: I/O is performed using the z/VM DIAGNOSE 'X'250' interface

  Note that the DIAGNOSE X'250' access mode only works when running Linux as z/VM guest, and only for devices formatted with consistent block sizes such as ECKD DASDs with LDL or CMS format, or FBA devices.

  DEFAULT VALUE
  The default value is '0'.
  ACCEPTED VALUES
  - Numbers 0 to 1 in decimal notation
  ```

- To remove a setting from a persistent configuration, use the --remove and --persistent options:

  ```
  # chzdev dasd 8000 --persistent --remove use_diag --verbose
  Configuring devices in the persistent configuration only
  FBA DASD 0.0.8000 configured
  Changes: -use_diag
  Block devices: /dev/dasda /dev/dasda1
  ```
Example for changing device driver settings

To change device driver settings, use a `<attribute=value>` assignment together with the device type:

```
# chzdev dasd --type nopav=1
dasd device type configured
```

Importing and exporting configuration data

Import configuration data from, or export data to, a text file by using the `--import` and `--export` options of the chzdev command. You can also use an existing configuration as a template for a new configuration, for example, for a new Linux instance.

About this task

The export function can be used, for example, to transfer a device configuration to another system, or to store it for later use as a backup or for a rollback.

Where:

- `--export <filename>`
  - writes configuration data to a text file called `<filename>`. If a single hyphen (`-`) is specified instead of a file name, data is written to the standard output stream. The output format of this option can be used with the `--import` option. To reduce the scope of exported configuration data, you can select specific devices, a device type, or define whether to export only data for the active or persistent configuration.

- `--import <filename>`
  - reads configuration data from `<filename>` and applies it. If a single hyphen (`-`) is specified instead of a file name, data is read from the standard input stream. The input format must be the same as the format produced by the `--export` option.

By default, all configuration data that is read is also applied. To reduce the scope of imported configuration data, you can select specific devices, a device type, or define whether to import only data for the active or persistent configuration.

- `--base <path> | <key=value>`
  - changes file system paths that are used to access files. If `<path>` is specified without an equal sign (`=`), it is used as base path for accessing files in the active and persistent configuration. If the specified parameter is in `<key=value>` format, only those paths that begin with `<key>` are modified. For these paths, the initial `<key>` portion is replaced with `<value>`.

Example: `lszdev --persistent --base /etc=/mnt/etc`

Examples

- Textual configuration data is either written to a file or, when you specify the special file name "-", to standard output:
# chzdev --by-interface enccw0.0.a000 --persistent --export -
Exporting configuration data to standard output
# Generated by chzdev on linux.example.com
[persistent qeth 0.0.a000:0.0.a001:0.0.a002]
layers=0
online=1

• To export this configuration data to a file called config.txt, issue:

  # chzdev --by-interface enccw0.0.a000 --persistent --export config.txt

• To apply data in this format to a system’s configuration, use the --import option:

  # chzdev --import config.txt
  Importing configuration data from config.txt
  QETH device 0.0.a000:0.0.a001:0.0.a002 configured

• To create a persistent configuration for an ECKD DASD 0.0.1000 and write it to a new Linux system that is mounted under /mnt/etc, issue:

  # chzdev dasd-eckd 1000 -e -p --base /etc=/mnt/etc

• To display a configuration that was written to /mnt/etc, issue:

  # lszdev --persistent --base /etc=/mnt/etc

• To copy the persistent configuration to /mnt/etc, issue:

  # chzdev --all --persistent --export - | chzdev --import - --base /etc=/mnt/etc
  Exporting configuration data to standard output
  Importing configuration data from Standard input
  ECKD DASD 0.0.1234 configured
  FCP device 0.0.190d configured
  FCP device 0.0.194d configured
  CTC device 0.0.c000:0.0.c001 configured

### Configuring the root device

Changing the configuration of the device that provides the root file system might require additional configuration steps. Which steps are taken depends on whether the Linux instance uses a RAM disk during boot.

**Before you begin**

**Tool builders:** If your Linux uses dracut and a RAM disk, build the lszdev and chzdev commands with HAVE_ZDEV_DRACUT=1. Otherwise, build with HAVE_ZDEV_DRACUT=0.

The actions chzdev performs depend on whether the lszdev and chzdev commands are built with HAVE_ZDEV_DRACUT= set to 0 or 1. "0" implies that there is no RAM disk, "1" implies that the Linux instance uses dracut and a RAM disk. The chzdev command takes the following actions:

**No RAM disk**

When built with setting HAVE_ZDEV_DRACUT=0, the chzdev command attempts to update the kernel command line found in file /etc/zipl.conf and run zipl to update the IPL records.

**dracut-based RAM disk**

When built with setting HAVE_ZDEV_DRACUT=1, a new dracut module is installed, which copies the persistent root device configuration to newly built initial RAM disks. Also, when the root device configuration is updated, the dracut command is run to rebuild the RAM disk, followed by zipl to update the IPL records.
Example

The following example shows a Linux instance with an initial RAM disk. The chzdev command has been compiled accordingly. The example changes the device that provides the root file system:

```bash
# chzdev --by-path / --enable --persistent
Configuring devices in the persistent configuration only
zFCP LUN 0.0.1940:0x500507630508c1ae:0x402140ac00000000 configured
Note: Some of the changes affect devices providing the root file system:
    - zFCP LUN 0.0.1940:0x500507630508c1ae:0x402140ac00000000
Additional steps such as rebuilding the RAM-disk might be required.
Update persistent root device configuration now? (yes/no) yes
Building initial RAM-disk
Installing IPL record
Using config file '/etc/zipl.conf'
Building bootmap in '/boot'
Building menu 'zipl-automatic-menu'
Adding #1: IPL section ‘3.10.0’ (default)
Preparing boot device: sda.
Done.
```

Note how additional steps are taken to update the IPL records.
Chapter 30. 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 KB pages; for IBM Z the hardware page size is 1 MB.

To check whether 1 MB large pages are supported in your environment, issue the command:

```
# grep edat /proc/cpuinfo
```

An output line that lists edat as a feature indicates 1MB large page support.

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.

As of zEC12, you can also configure 2 GB large pages if Linux is running on an LPAR. See “Pre-allocating 2 GB large pages” on page 392.

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 127 on page 391 shows the kernel configuration menu options that are relevant to large page support.

```
Memory setup --->
... Transparent large page support (CONFIG_TRANSPARENT_HUGEPAGE)
... File systems --->
... Pseudo filesystems --->
... HugeTLB file system support (CONFIG_HUGETLBFS)
```

Figure 127: 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 parameters to the kernel parameter line.

```
default_hugepagesz = 1M
hugepagesz = <size>

default_hugepagesz = <size>
hugepages = <number>
```

**Figure 128: Large page support kernel parameter syntax**

where:

- `default_hugepagesz=<size>` specifies the default page size in byte. You can use suffixes K, M, and G to specify KB, MB, and GB. The default value is 1 MB.
- `hugepages=<number>` is the number of large pages to be allocated at boot time.
- `hugepagesz=<size>` specifies the page size in byte. You can use suffixes K, M, and G to specify KB, MB, and GB.

**Note:** If you specify more pages than available, Linux reserves as many as possible. As a likely result, too few general pages remain for the boot process, and your system stops with an out-of-memory error.

Pre-allocating 2 GB large pages

If Linux is running in an LPAR, you can use 2 GB large pages.

Before you can use 2 GB large pages, you must pre-allocate them to the kernel page pool. To pre-allocate 2 GB pages, precede the `hugepages=` parameter with the page size selection parameter, `hugepagesz=2G`.

**Tip:** Memory quickly becomes fragmented after booting, and new 2 GB large pages cannot be allocated. Use kernel boot parameters to allocate 2 GB large pages to avoid the memory fragmentation problem.

To pre-allocate a number of pages of 2 GB size and also set the default size to 2 GB:

```
default_hugepagesz=2G hugepagesz=2G hugepages=<number>
```

Setting up multiple large page pools

You can allocate multiple large page pools and use them simultaneously. To allocate multiple large page pools, specify the `hugepagesz=` parameter several times, each time followed by a corresponding `hugepages=` parameter.

For example, to specify two pools, one with 1 MB pages and one with 2 GB pages, specify:

```
hugepagesz=1M hugepages=8 hugepagesz=2G hugepages=2
```

This creates a sysfs attribute for each pool, `/sys/kernel/mm/hugepages/hugepages-<size>kB/nr_hugepages`, where `<size>` is the page size in KB. For the example given, the following attributes are created:

```
/sys/kernel/mm/hugepages/hugepages-1024kB/nr_hugepages
/sys/kernel/mm/hugepages/hugepages-2097152kB/nr_hugepages
```

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:

```bash
# 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:

```bash
# 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.

Your database product might support large page memory. See your database documentation to find out if and how it can be configured to use large page memory.

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:

```bash
# 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 default-sized large pages, issue:

  ```bash
  # cat /proc/sys/vm/nr_hugepages
  ```

- To change the number of default-sized large pages dynamically during runtime, write to procfs:

  ```bash
  # echo 20 > /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 default-sized large pages currently available and the default large page size, issue:

  ```bash
  # cat /proc/meminfo
  ...
  HugePages_Total: 20
  HugePages_Free: 14
  HugePages_Rsvd: 0
  HugePages_Surp: 0
  ...
  Hugepagesize: 1024 kB
  ...
  ```

- To adjust characteristics of a large-page pool, when more than one pool exists, use the sysfs attributes of the pool.
  These can be found under

  ```bash
  /sys/kernel/mm/hugepages/hugepages-<size>/nr_hugepages
  ```

  Where `<size>` is the page size in KB.

**Example**

To allocate 2 GB large pages:

1. Specify 2 GB large pages and pre-allocate them to the page pool at boot time. Use the following kernel boot parameters:

   ```bash
   default_hugepagesz=2G hugepagesize=2G hugepages=4
   ```

2. After booting, read `/proc/meminfo` to see information about the amount of large pages currently available and the large page size:

   ```bash
   cat /proc/meminfo
   ...
   HugePages_Total: 4
   HugePages_Free: 4
   HugePages_Rsvd: 0
   HugePages_Surp: 0
   Hugepagesize: 2097152 kB
   ...
   ```
Chapter 31. 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 129 on page 396 illustrates the file system tree that is created for LPAR.
Figure 129: The hypervisor file system for LPAR

**update**
Write-only file to trigger an update of all attributes.

**cpus/**
Directory for all physical cores.

**cpus/**<core_ID>
Directory for one physical core. <core_ID> is the logical (decimal) core number.

  **type**
  Type of the physical core, such as CP or IFL.

  **mgmtime**
  Physical-LPAR-management time in microseconds (LPAR overhead).

**hyp/**
Directory for hypervisor information.

**hyp/**type
Type of hypervisor (LPAR hypervisor).

**systems/**
Directory for all LPARs.

**systems/**<lpar name>/
Directory for one LPAR.

**systems/**<lpar name>/cpus/<core_ID>/
Directory for the virtual cores for one LPAR. The <core_ID> is the logical (decimal) core number.

  **type**
  Type of the logical core, such as CP or IFL.
**mgmtime**

LPAR-management time. Accumulated number of microseconds during which a physical core was assigned to the logical core and the core time was consumed by the hypervisor and was not provided to the LPAR (LPAR overhead).

**cputime**

Accumulated number of microseconds during which a physical core was assigned to the logical core and the core time was consumed by the LPAR.

**onlinetime**

Accumulated number of microseconds during which the logical core has been online.

**Note:** For LPARs with multithreading enabled, the entities in the cpus directories represent hardware cores, not threads.

**Note:** For older machines, the `onlinetime` attribute might be missing. Generally, it is advantageous for applications to tolerate missing attributes or new attributes that are added to the file system. To check the content of the files, you can use tools such as `cat` or `less`.

### z/VM directories and attributes

There are hypfs directories and attributes with hypervisor information for Linux on z/VM.

**update**

Write-only file to trigger an update of all attributes.

**cpus/**

Directory for all physical CPUs.

**cpus/count**

Total current CPUs.

**hyp/**

Directory for hypervisor information.

**hyp/type**

Type of hypervisor (z/VM hypervisor).

**systems/**

Directory for all z/VM guest virtual machines.

**systems/<guest name>/**

Directory for one 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:

```bash
# 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:

```bash
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:

```bash
mount: unknown filesystem type 's390_hypfs'
```

To get data for all z/VM guests, privilege class B is required for the guest where hypfs is mounted. For non-class B guests, data is provided only for the local guest.
To get data for all LPARs, select the **Global performance data control** check box in the HMC or SE security menu of the LPAR activation profile. Otherwise, data is provided only for the local LPAR.

## Working with the S/390 hypervisor file system

Typical tasks that you must perform when working with the S/390 hypervisor file system include defining access permissions and updating hypfs information.

- “Defining access permissions” on page 399
- “Updating hypfs information” on page 399

### 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 32. 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 130 on page 401).

<table>
<thead>
<tr>
<th>I/O subsystem ---</th>
<th>Support for CHSC subchannels (CONFIG_CHSC_SCH)</th>
</tr>
</thead>
</table>

Figure 130: 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” on page 402 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 54 on page 402 lists the ioctls that are defined.

<table>
<thead>
<tr>
<th>Name</th>
<th>Structure passed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHSC_START</td>
<td>struct chsc_async_area</td>
<td>Use to start an asynchronous chsc request that changes the I/O configuration.</td>
</tr>
<tr>
<td>CHSC_INFO_CHANNEL_PATH</td>
<td>struct chsc_chp_cd</td>
<td>Use to obtain information about a specific channel path.</td>
</tr>
<tr>
<td>CHSC_INFO_CU</td>
<td>struct chsc_cu_cd</td>
<td>Use to obtain information about a specific control unit.</td>
</tr>
<tr>
<td>Name</td>
<td>Structure passed</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CHSC_INFO_SCH_CU</td>
<td>struct chsc_sch_cud</td>
<td>Use CHSC_INFO_SCH_CU to obtain information about control units on a subchannel.</td>
</tr>
<tr>
<td>CHSC_INFO_CI</td>
<td>struct chsc_conf_info</td>
<td>CHSC_INFO_CI can be used to obtain the configuration information as needed by the dynamic I/O configuration chscs.</td>
</tr>
<tr>
<td>CHSC_INFO_CCL</td>
<td>struct chsc_comp_list</td>
<td>CHSC_INFO_CCL can be used to obtain information about various configuration components.</td>
</tr>
<tr>
<td>CHSC_INFO_CPD</td>
<td>struct chsc_cpd_info</td>
<td>Use CHSC_INFO_CPD to obtain a description of a specified channel path.</td>
</tr>
<tr>
<td>CHSC_INFO_DCAL</td>
<td>struct chsc_dcal</td>
<td>Use CHSC_INFO_DCAL to obtain domain attributes of the configuration.</td>
</tr>
<tr>
<td>CHSC_START_SYNC</td>
<td>struct chsc_sync_area</td>
<td>Use CHSC_START_SYNC to start a synchronous chsc request.</td>
</tr>
<tr>
<td>CHSC_ON_CLOSE_SET</td>
<td>struct chsc_async_area</td>
<td>Install the asynchronous &quot;on close&quot; chsc request. This request is executed when the device node is closed.</td>
</tr>
<tr>
<td>CHSC_ON_CLOSE_REMOVE</td>
<td>none</td>
<td>Remove the asynchronous &quot;on close&quot; chsc request.</td>
</tr>
</tbody>
</table>
Chapter 33. Call logical processor device driver

Through the call logical processor (CLP) device driver, you can issue a restricted set of CLP instructions to retrieve information about available PCI functions. Applications can use this device driver to obtain information about PCI functions.

Building a kernel with the CLP device driver

Kernels that are built with PCIe support include the CLP 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 common code CONFIG_PCI kernel configuration option to build a kernel with PCIe support (see also “Building a kernel with the PCIe support” on page 427). The CLP device driver is always built into the kernel image.

Assuring that a device node exists

The CLP device driver provides a misc character device, /dev/clp. If your distribution does not create the device node for you, create a node.

Example

Check for an existing node:

```
# find /dev -name clp
```

If no node is found, use major number 10 and a dynamic minor number to create one. Issue the following command to find the minor number:

```
# grep clp /proc/misc
58 clp
```

The minor number of the example is 58. Issue an mknod command to create the node:

```
# mknod /dev/clp c 10 58
```

Applications can issue an ioctl on this device to obtain information about the available CPI functions (see “External programming interfaces ” on page 405).

External programming interfaces

Applications can use the CLP device driver through an ioctl.

Programmers: This information is intended for programmers of applications that need information about PCI functions on IBM Z.

Issue the CLP_SYNC ioctl on the misc character device /dev/clp to obtain information about PCI functions.
Chapter 34. 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 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 information about STP, see


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 409).
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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>on</code></td>
<td>sets both ports online.</td>
</tr>
<tr>
<td><code>port0</code></td>
<td>sets port0 online and port1 offline.</td>
</tr>
<tr>
<td><code>port1</code></td>
<td>sets port1 online and port0 offline.</td>
</tr>
<tr>
<td><code>off</code></td>
<td>sets both ports offline. With both ports offline, ETR-based clock synchronization is not enabled. This is the default.</td>
</tr>
</tbody>
</table>

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.

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:

```bash
# echo 0 > /sys/devices/system/stp/online
```
Chapter 35. Identifying the IBM Z hardware

In installations with several IBM Z mainframes, you might need to identify the particular hardware system on which a Linux instance is running.

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:**

```bash
# cat /sys/firmware/ocf/cpc_name
Z05
# cat /sys/firmware/ocf/hmc_network
SNA08
```
Chapter 36. HMC media device driver

You use the HMC media device driver to access files on removable media at a system that runs the Hardware Management Console (HMC).

Before you begin: You must log in to the HMC on the system with the removable media and assign the media to the LPAR.

As of System z10, the HMC media device driver supports the following removable media:

• A DVD in the DVD drive of the HMC system
• A CD in the DVD drive of the HMC system
• USB-attached storage that is plugged into the HMC system

The most commonly used removable media at the HMC is a DVD.

The HMC media device driver uses the /dev/hmcdrv device node to support these capabilities:

• List the media contents with the lshmc command (see “lshmc - List media contents in the HMC media drive” on page 665).
• Mount the media contents as a file system with the hmcdrvfs command (see “hmcdrvfs - Mount a FUSE file system for remote access to media in the HMC media drive” on page 641).

Installers on suitably prepared installation DVDs can use these capabilities to install Linux in an LPAR.

Building a kernel with the HMC media device driver

Control the build options for the HMC media 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 133 on page 413 shows where to find the kernel configuration menu option that is required for the HMC media device driver.

You must select the kernel configuration option CONFIG_HMC_DRV and the common code option CONFIG_FUSE_FS to build a kernel that supports access to the files on removable media at the HMC system.

The HMC media device driver can be compiled into the kernel or as a separate module, hmcdrv.
Setting up the HMC media device driver

You can set the cache size for the HMC media device driver.

**Kernel parameters**

If the HMC media device driver has been compiled into the kernel, you set the cache size by adding the `hmcdrv.cachesize=` parameter to the kernel parameter line.

```
Figure 134: hmcdrv kernel parameter syntax
```

where `<size>` is the cache size in bytes. The specification must be a multiple of 2048. You can use the suffixes K for kilobytes, M for megabytes, or G for gigabytes. Specify 0 to not cache any media content. By default, the cache size is 534288 bytes (0.5 MB).

Loading the `hmcdrv` module creates a device node at `/dev/hmcdrv`.

**Example**
The following specifications are equivalent:

```
  hmcdrv.cachesize=153600
  hmcdrv.cachesize=150K
```

**Module parameters**

If the HMC media device driver has been built as a separate module, `hmcdrv`, you set the cache size through the `cachesize=` module parameter.

Before you can work with the HMC media device driver and with the dependent `lshmc` and `hmcdrvfs` commands, you must load the `hmcdrv` kernel module.

```
Figure 135: hmcdrv module parameter syntax
```

where `<size>` is the cache size in bytes. The specification must be a multiple of 2048. You can use the suffixes K for kilobytes, M for megabytes, or G for gigabytes. Specify 0 to not cache any media content. By default, the cache size is 534288 bytes (0.5 MB).

Loading the `hmcdrv` module creates a device node at `/dev/hmcdrv`.

**Example**
The following specifications are equivalent:

```
  # modprobe hmcdrv cachesize=153600
  # modprobe hmcdrv cachesize=150K
```
Working with the HMC media

You can list files on media that are inserted into the HMC system and you can mount the media content on the Linux file system.

- “Assigning the removable media of the HMC to an LPAR” on page 415
- “Listing files on the removable media at the HMC” on page 415
- “Mounting the content of the removable media at the HMC” on page 416

Assigning the removable media of the HMC to an LPAR

Use the HMC to assign the removable media to the LPAR where your Linux instance runs.

Before you begin

- You need access to the HMC, and you must be authorized to use the Access Removable Media task for the LPAR to which you want to assign the media.
- For Linux on z/VM, the z/VM guest virtual machine must have at least privilege class B.
- For Linux in LPAR mode, the LPAR activation profile must allow issuing SCLP requests.

About this task

You can list files on the removable media at the HMC without having to first mount the contents on the Linux file system.

Procedure

1. Insert the removable media into the HMC system.
2. Use the Access Removable Media task on your HMC to assign the removable media to the LPAR where your Linux instance runs.
   
   For Linux on z/VM, this is the LPAR where the z/VM hypervisor runs that provides the guest virtual machine to your Linux instance.
   
   For details, see the HMC documentation for the HMC at your installation.

Results

You can now access the removable media from your Linux instance.

Listing files on the removable media at the HMC

Use the lshmc command to list files on the removable media at the HMC.

Before you begin

Your Linux instance must have access to the removable media at the HMC (see “Assigning the removable media of the HMC to an LPAR” on page 415).

About this task

You can list files on the removable media at the HMC without having to first mount the contents on the Linux file system.

Procedure

Issue a command of this form:

```bash
# lshmc <filepath>
```
where `<filepath>` is an optional specification for a particular path and file. Path specifications are interpreted as relative to the root directory of the removable media. You can use the asterisk (*) and question mark (?) as wildcards. If you omit `<filepath>`, all files in the root directory of the media are listed.

**Example:** The following command lists all `.html` files in the `www` subdirectory of the media.

```
# lshmc www/*/html
```

For more information about the `lshmc` command, see "`lshmc - List media contents in the HMC media drive`" on page 665.

### Mounting the content of the removable media at the HMC

Use the `hmcdrvfs` command to mount the content of the removable media at the HMC.

**Before you begin**

Your Linux instance must have access to the removable media of the HMC (see “Assigning the removable media of the HMC to an LPAR” on page 415).

**About this task**

You can mount the content of the removable media at the HMC in read-only mode on the Linux file system.

**Procedure**

1. Optional: Confirm that you are accessing the intended content by issuing the `lshmc` command.
2. Mount the media content by issuing a command of this form:

   ```
   # hmcdrvfs <mountpoint>
   ```

   where `<mountpoint>` is the mount point on the Linux file system.

   **Example:** The following command mounts the media content at `/mnt/hmc`:

   ```
   # hmcdrvfs /mnt/hmc
   ```

**Results**

You can now access the files on the media in read-only mode through the Linux file system.

**What to do next**

When you no longer need access to the media content, unmount the media with the `fusermount` command.
Chapter 37. Data compression with GenWQE and zEDC Express

Generic Work Queue Engine (GenWQE) supports hardware-accelerated data compression and decompression through zEDC Express, a PCIe-attached Field Programmable Gate Array (FPGA) acceleration adapter.

zEDC Express is available for zEC12 and later IBM mainframes.

zEDC hardware-acceleration is available for both Linux and z/OS. For more information about zEDC on z/OS and about setting up zEDC Express, see Reduce Storage Occupancy and Increase Operations Efficiency with IBM zEnterprise Data Compression, SG24-8259. You can obtain this publication from the IBM Redbooks® website at www.redbooks.ibm.com/abstracts/sg248259.html.

Features

GenWQE supports hardware-accelerated data compression and decompression using common standards.

• GenWQE implements the zlib API.
• GenWQE adheres to the following RFCs:
  – RFC 1950 (zlib)
  – RFC 1951 (deflate)
  – RFC 1952 (gzip)

These standards ensure compatibility among different zlib implementations.

– Data that is compressed with GenWQE can be decompressed through a zlib software library.
– Data that is compressed through a software zlib software library can be decompressed with GenWQE.
• GenWQE supports the following PCIe FPGA acceleration hardware:
  – zEDC Express

What you should know about GenWQE

Learn about the GenWQE components, how to enable GenWQE accelerated zlib for user applications, and device representation in Linux.

The GenWQE accelerated zlib

The GenWQE accelerated zlib can replace a zlib software library.

For data compression and decompression tasks, Linux distributions include software libraries. The zlib library, which provides the zlib API, is one of the most commonly used libraries for data compression and decompression. For information about zlib, see www.zlib.net.

Because the GenWQE accelerated zlib offers the zlib API, applications can use it instead of the default zlib software library. The GenWQE hardware-accelerated zlib is designed to enhance performance by offloading tasks to a hardware accelerator.
Applications
As an experimental setup, you can deploy the user space components of the GenWQE hardware-accelerated zlib. This zlib implementation is then available to applications that request data compression functions through the zlib API.

The following applications are designed to use GenWQE if it is available:

- genwqe_gzip and genwqe_gunzip, which are GenWQE versions of gzip and gunzip (see “Examples for using GenWQE” on page 422).
  
  These tools can be used for most purposes, but they do not implement all of the more unusual options of their common code counterparts. See the man pages to find out which options are supported.

- genwqe_echo, a tool to confirm the availability of accelerator hardware through the GenWQE accelerated zlib (see “Confirming that the accelerator hardware can be reached” on page 424).

IBM Java version 7.1 or later includes components of the GenWQE hardware-accelerated zlib. Through these components, it can directly address the GenWQE device nodes. With the required environment variables in place, it uses hardware-acceleration if it is available (see “GenWQE hardware-acceleration for IBM Java” on page 423).

Hardware-accelerated zlib
The hardware-accelerated zlib is a zlib implementation that acts as a wrapper for two included libraries:

libzHW
a hardware library that prepares requests for processing by the hardware accelerator. The hardware library is intended to handle the bulk of the requests.

This library also manages data buffers for optimized hardware compression.

libz
a software implementation of the zlib interface. Because it provides the same interface as its wrapper library, it can handle any requests unmodified.
The hardware-accelerated zlib arbitrates between the two included libraries. It uses the software library as a backup if no hardware accelerator is available. It also evaluates the expected performance gain against the extra processing for channeling requests to the accelerator. For small or fragmented data, software processing might be advantageous, especially for decompression. The evaluation takes available resources, such as buffer space, into account.

**Card library**
The card library, `libcard`, mediates between the hardware-accelerated zlib library and the GenWQE device driver. It provides recovery features and can move jobs between available accelerators.

**Device driver**
The GenWQE device driver is the kernel part of GenWQE. It serializes requests to an accelerator in form of device driver control blocks (DDCBs), and it enables multi-process and multi-thread usage.

**GenWQE device nodes**
GenWQE user space components use device nodes to exchange data with the GenWQE device driver. Loading the GenWQE device driver module creates a device node of the form `/dev/genwqe<n>_card` for each available virtual acceleration card. `<n>` is an index number that identifies an individual virtual card.

Node `/dev/genwqe0_card` is assigned to the first card that is detected, `/dev/genwqe1_card` to the second card, and so on. Do not directly use these device nodes.

The nodes must grant read and write permissions to all users, for example:

```bash
# ls -l /dev/genwqe*
crwrwrw 1 root root 249, 0 Jun 30 10:01 /dev/genwqe0_card
crwrwrw 1 root root 248, 0 Jun 30 10:01 /dev/genwqe1_card
```

If the permissions are not `crwrwrw`, create a file `/etc/udev/rules.d/52-genwqedevices.rules` with this rule as its content:

```bash
KERNEL="genwqe\*", MODE="0666"
```

The new rule takes effect next time the GenWQE device driver is loaded.

**Tip**: Use the `chmod` command to temporarily set the permissions.

**Virtual accelerators**
Each physical accelerator card can provide up to 15 virtual cards. In PCIe terminology, these virtual cards are called virtual functions.

GenWQE accelerator cards, as detected by Linux on Z, are virtual cards. Which and how many cards are available to a particular Linux instance depends on the mainframe configuration and, if applicable, the hypervisor configuration.

As for most mainframe devices, availability can be enhanced by assigning virtual accelerator cards from different physical cards.

A degree of load distribution can be achieved by unevenly distributing accelerator cards among different Linux instances.

**Tradeoff between best compression and speed**
A minimum size of compressed data and fast compression are conflicting goals.

For hardware-accelerated compression with GenWQE, the compression ratio is roughly equivalent to `gzip --fast`.

Data that has been compressed with GenWQE hardware-acceleration might have a different size from data that has been compressed in software. The data compression standards are not violated by this
difference. Despite possible differences in size of the compressed data, data that is compressed with GenWQE hardware-acceleration can be decompressed in software and vice versa.

### Building a kernel with the GenWQE device driver

Control the build options for the GenWQE 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 following figure shows where to find the kernel configuration menu option for the GenWQE device driver:

![Figure 137: GenWQE kernel configuration menu option](image)

The CONFIG_GENWQE option is required if you want to use zlib hardware acceleration through GenWQE. The GenWQE device driver can be compiled as a separate module, genwqe_card, or built-in.

You also need the common code option CONFIG_PCI.

### Setting up hardware acceleration

You might have to install the workspace components of the hardware-accelerated zlib and load the GenWQE device driver.

#### Obtaining the code

Compile the components of the GenWQE hardware-accelerated zlib from the source code from the GitHub website.

Obtain the source code from github.com/ibm-genwqe/genwqe-user. See the wiki to be sure that you use a stable version.

After downloading the code, run the make file to generate the object code for the GenWQE user space components, including the GenWQE tools.

#### Loading the device driver module

If the GenWQE device driver has been compiled as a separate module, you must load it before you can use GenWQE data compression functions.

Load the genwqe_card module with the modprobe command. There are no module parameters.

```
# modprobe genwqe_card
```
Environment variables

You can set environment variables to control the GenWQE hardware-accelerated zlib.

The GenWQE hardware-accelerated zlib uses defaults that correspond to the following environment variable settings:

```
ZLIB_ACCELERATOR=GENWQE
ZLIB_CARD=-1
ZLIB_TRACE=0x0
ZLIB_DEFLATE_IMPL=0x41
ZLIB_INFLATE_IMPL=0x41
```

You can override these defaults by setting the following environment variables:

**ZLIB_ACCELERATOR**
Sets the accelerator type. For zEDC Express, the type is GENWQE.

**ZLIB_CARD**
-1, uses all accelerators that are available to the Linux instance. Failed requests are retried on alternative accelerators.

You can specify the ID of a particular virtual accelerator card to be used. The ID is the index number used in the device nodes to make the nodes unique. All other cards are ignored, and no retry on alternative cards is performed if the specified card fails. Specify an ID only if you want to test a particular card.

0 uses the first card that is found by the device driver. As for specifying an individual card, all other cards are ignored.

**ZLIB_TRACE**
Sets tracing bits:

0x1 General trace
0x2 Hardware trace
0x4 Software trace
0x8 Trace summary at the end of a process

Tracing requires extra processing and incurs a performance penalty. The least performance impact is to be expected from the trace summary. By default, tracing is off.

**ZLIB_DEFLATE_IMPL**
0x01 and 0x41 enable hardware compression, where 0x41 adds an optimization setting. 0x00 forces software compression and is intended for experimentation, for example, for gathering performance data with and without hardware acceleration.

**ZLIB_INFLATE_IMPL**
0x01 and 0x41 enable hardware decompression, where 0x41 adds an optimization setting. 0x00 forces software decompression and is intended for experimentation, for example, for gathering performance data with and without hardware acceleration.

You can find more details about these environment variables in the GenWQE wiki on GitHub at github.com/ibm-genwqe/genwqe-user/wiki/Environment Variables.
Examples for using GenWQE

You can use the GenWQE hardware-accelerated zlib through GenWQE tools.

Activating the GenWQE hardware-accelerated zlib for an application

Whether and how you can make an application use the GenWQE hardware-accelerated zlib depends on how the application links to `libz.so`.

Examine the application for links to `libz.so`, for example with the `ldd` tool.

- If the application does not link to `libz.so` or if it statically links to `libz.so`, it would require recompilation, and possibly code changes, to make acceleration through GenWQE possible.
- If an application dynamically links to `libz.so`, you might be able to redirect the library calls from the default implementation to the GenWQE hardware-accelerated zlib.

Some applications require zlib features that are not available from the GenWQE hardware-accelerated zlib. Such applications fail if a global redirect is put in place. The following technique redirects calls for the scope of a particular application.

Specify the LD_PRELOAD environment variable to load the GenWQE hardware-accelerated zlib. Set the variable with the start command for your application.

Example:

```bash
# LD_PRELOAD=/lib/s390x-linux-gnu/genwqe/libz.so.1 <application_start_cmd>
```

Compressing data with genwqe_gzip

GenWQE provides two tools, `genwqe_gzip` and `genwqe_gunzip` that can be used in place of the common code `gzip` and `gunzip` tools. The GenWQE versions of the tools use hardware acceleration if it is available.

Procedure

Run the `genwqe_gzip` command with the `-AGENWQE` parameter to compress a file.

```bash
# genwqe_gzip -AGENWQE <file>
```

The `-AGENWQE` parameter ensures that the correct, PCIe-attached, accelerator card is used. Also use this option when decompressing data with the `genwqe_gunzip` command. See the man pages for other options.

Running tar with GenWQE hardware-acceleration

GenWQE provides two tools, `genwqe_gzip` and `genwqe_gunzip`, that can be used in place of the common code `gzip` and `gunzip` tools. The GenWQE versions of the tools use hardware acceleration.

About this task

If called with the `z` option, the tar utility uses the first `gzip` tool in the search path, which is usually the common code version. By inserting the path to the GenWQE version of the `gzip` tool at the beginning of the `PATH` variable, you can make the tar utility use hardware acceleration.

The effect is most marked for a single large text file. The example that follows compresses a directory with the Linux source code.

Procedure

1. Run the `tar` command as usual to use software compression. To obtain performance data, specify the `tar` command as an argument to the `time` command.
2. Run the tar command with an adjusted PATH variable to use GenWQE hardware acceleration. Again, use the time command to obtain performance data.

```bash
# time PATH=/usr/lib/genwqe:$PATH \
    tar cfz linux-src.hw.tar.gz linux-src
real 0m1.323s
user 0m0.242s
sys 0m1.023s
```

In the example, the accelerated operation is significantly faster.

The hardware-compressed data is slightly larger than the software-compressed version of the same data.

**Note:** In this scenario, the path to the GenWQE gzip tool is `/usr/lib/genwqe`. This path depends on distribution-specific requirements and might be different for your distribution.

---

## GenWQE hardware-acceleration for IBM Java

IBM Java version 7.1 or later can use the GenWQE hardware-accelerated zlib.

To activate the GenWQE hardware-accelerated zlib for IBM Java, you must set environment parameters. See the documentation for your Java version to find out which settings are required.

**Note:** Any values that you set for the environment variables override the default settings for the GenWQE user space components (see “Environment variables” on page 421).

---

## Exploring the GenWQE setup

You might want to ensure that your GenWQE setup works as intended.

- “Listing your GenWQE accelerator cards” on page 423
- “Checking the GenWQE device driver setup” on page 424
- “Confirming that the accelerator hardware can be reached” on page 424

---

### Listing your GenWQE accelerator cards

Use the `lspci` command to list the available GenWQE accelerator cards.

**Procedure**

1. Issue the `lspci` command and look for GenWQE.
   
   **Example:**
   
   ```bash
   # lspci | grep GenWQE
   0002:00:00.0 Processing accelerators: IBM GenWQE Accelerator Adapter
   ```

2. Issue the `lspci` command with the verbose option to display details about a particular card.
   
   **Example:**
   
   ```bash
   ```

---
Checking the GenWQE device driver setup

Perform these tasks if GenWQE does not work as expected.

Procedure

1. Confirm that the device driver is loaded.

   ```bash
   # lsmod | grep genwqe
genwqe_card 88997 0
crc_itu_t 1910 1 genwqe_card
   ```

   If the `genwqe_card` module is not listed in the command output, load it with `modprobe` (see “Loading the device driver module” on page 420).

   **Hint:** Use the following command to find out whether the device driver is compiled into the kernel:

   ```bash
   # cat /lib/modules/$(uname -r)/modules.builtin | grep genwqe
   ```

2. Confirm that GenWQE device nodes have been created and that the nodes have the required permissions (see “GenWQE device nodes” on page 419).

3. Verify that all required environment variables are set to suitable values and that no variables override correct default settings.

   ```bash
   # env | grep ZLIB
   ```

What to do next

You can find debug information in the Linux source tree at `/sys/kernel/debug/genwqe` and at `/sys/class/genwqe`

Confirming that the accelerator hardware can be reached

The `genwqe_echo` command is similar to a `ping` command.

**Before you begin**

The `genwqe_echo` command is included with the GenWQE components that you can obtain from github.

**Procedure**

Issue a command of this form to confirm that you can reach the accelerator hardware.

```bash
# genwqe_echo -AGENWQE -C <n> -c <m>
```

In the command, `<n>` is the index number of the card and `<m>` is a positive integer that specifies how many requests are sent to the card. The `-AGENWQE` parameter ensures that the correct, PCIe-attached, accelerator card is used.

**Example:** The following command sends 4 requests to the card with device node `/dev/genwqe1_card`:
See the `genwqe_echo` man page for other command options you can use to obtain information about the accelerator.

**External programming interfaces**

The GenWQE hardware-accelerated zlib implements a large subset of the original software zlib.

For information about programming against the GenWQE hardware-accelerated zlib, see the section about implemented zlib functions in *Accelerated Data Compressing using the GenWQE Linux Driver and Corsa FPGA PCIe card*.

To obtain this document, go to the developerWorks website at [www.ibm.com/developerworks/community/files/app](http://www.ibm.com/developerworks/community/files/app) and search for "genwqe".
Chapter 38. PCI Express support

The Peripheral Component Interconnect Express (PCIe) device driver supports various PCI devices, including but not limited to devices that implement the SMC network protocol.

For more information about RoCE, see Chapter 24, “RDMA over Converged Ethernet,” on page 353. For more information about ISM, see Chapter 25, “Internal shared memory device driver,” on page 357.

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

Linux supports UIDs as persistent identifiers for PCI functions. Provide UIDs for required PCI functions in the hardware configuration (IOCDS). The LPAR needs to be enabled for UID checking. UIDs are unique hexadecimal values in the range 1 - FFFF. For example, with a UID of 0x318, the function address would be: 0318:00:00.0.

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 138 on page 427 summarizes the kernel configuration menu options that are relevant to the PCIe support:

```
I/O subsystem --->
    ... PCI support --->
        Maximum number of PCI functions (1-4096) (CONFIG_PCI_NR_FUNCTIONS)
        Support for PCI Hotplug ---> (common code option CONFIG_HOTPLUG_PCI)
        System z PCI Hotplug Support (CONFIG_HOTPLUG_PCI_S390)
```

*Figure 138: PCIe kernel configuration menu options*

**CONFIG_PCI**

Enable PCI support.

**Hint:** Configuration-option defaults and inter-dependencies likely result in the CONFIG_USB_SUPPORT being selected automatically when selecting CONFIG_PCI. To avoid unwanted support for USB and related items, ensure that CONFIG_USB_SUPPORT is disabled at Device drivers > USB Support in the configuration menu.

**CONFIG_PCI_NR_FUNCTIONS**

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

**CONFIG_HOTPLUG_PCI_S390**

Hotplug support enables simple online and offline configuration through the `power` sysfs attribute. If selected, the PCIe support is compiled into the kernel.
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 IBM Z and might have adverse effects on your system.

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

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 `lspci -v` command to find the slot.

2. Write the value that you want to the `power` attribute:
   
   - Write 1 to `power` to connect the PCIe device to the LPAR in which your Linux instance is running. Linux automatically scans the device, registers it, and brings it online. For example:
   
   ```
   echo 1 > /sys/bus/pci/slots/00000011/power
   ```
   
   - Write 0 to `power` to stop using the PCIe device. The device state changes to stand-by. The PCIe device is set offline automatically. For example:
   
   ```
   echo 0 > /sys/bus/pci/slots/00000011/power
   ```
   
   A PCIe device in standby is also in the standby state to all other LPARs in the candidates list. A standby PCIe device appears as a slot, but without a PCIe device.
Recovering a PCIe device

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

**About this task**

A 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<function_name>/<function_address>`
   
   where `<function_name>` and `<function_address>` identify the PCIe device.
   
   For example: `/sys/devices/pci0000:00/0000:00:00.0`.

2. Write 1 to the `recover` attribute of the PCIe device.
   
   For example:
   
   ```bash
   # echo 1 > /sys/devices/pci0000:00/0000:00:00.0/recover
   ```
   
   After a successful recovery, the PCI device is de-registered and reprobed.

Reporting defective PCIe devices

Use the `zpcictl` command to report defective PCIe devices to the support element (SE). Such devices might require physical repair actions.

**Before you begin**

- You need to know the function address of the defective PCIe device.

**Procedure**

Issue a command of this form to report a device with function address `0000:00:00.0` to the SE:

```bash
# zpcictl --report-error <device>
```

where `<device>` is the device’s function address.

**Example:**

```bash
# zpcictl --report-error 0000:00:00.0
```

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<function_name>/<function_address>`

where `<function_name>` and `<function_address>` identify the PCIe device. This sysfs directory contains a number of attributes with information about the PCIe device.
### Table 55: Read-only attributes with PCIe device information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>function_handle</td>
<td>Eight-character, hexadecimal PCI-function (device) handle.</td>
</tr>
<tr>
<td>function_id</td>
<td>Eight-character, hexadecimal PCI-function (device) ID. The ID identifies</td>
</tr>
<tr>
<td></td>
<td>the PCIe device within the processor configuration.</td>
</tr>
<tr>
<td>pchid</td>
<td>Four-character, hexadecimal, physical channel ID. Specifies the slot of the</td>
</tr>
<tr>
<td></td>
<td>PCIe adapter in the I/O drawer. Thus identifies the adapter that provides</td>
</tr>
<tr>
<td></td>
<td>the device.</td>
</tr>
<tr>
<td>pfgid</td>
<td>Two-character, hexadecimal, physical function group ID.</td>
</tr>
<tr>
<td>pfip/segment0</td>
<td>Two-character, hexadecimal, PCI-function internal path. Provides an</td>
</tr>
<tr>
<td>segment1</td>
<td>abstract indication of the path that is used to access the PCI function.</td>
</tr>
<tr>
<td>segment2</td>
<td>This can be used to compare the paths used by two or more PCI functions, to</td>
</tr>
<tr>
<td>segment3</td>
<td>give an indication of the degree of isolation between them.</td>
</tr>
<tr>
<td>uid</td>
<td>Up to eight-character, hexadecimal, user-defined identifier.</td>
</tr>
<tr>
<td>vfn</td>
<td>Four-character, hexadecimal, virtual function number. If an adapter,</td>
</tr>
<tr>
<td></td>
<td>identified by its PCHID, supports more than one PCI function, the VFN</td>
</tr>
<tr>
<td></td>
<td>uniquely identifies the instance of that function within the adapter.</td>
</tr>
<tr>
<td>util_string</td>
<td>Type-specific information about the device. For RoCE devices and ISM</td>
</tr>
<tr>
<td></td>
<td>devices, it contains the PNET ID if a PNET ID has been assigned in the I/O</td>
</tr>
<tr>
<td></td>
<td>configuration (“Investigating PNET IDs” on page 352).</td>
</tr>
</tbody>
</table>

**Procedure**

Issue a command of this form to read an attribute:

```bash
# cat /sys/devices/pci<function_name>/<function_address>/<attribute>
```

where `<attribute>` is one of the attributes of Table 55 on page 430.

**Reading statistics for a PCIe device**

Use the statistics attribute file to see measurement data for a PCIe device.

**About this task**

All PCIe devices collect measurement data by default. You can read the data in a sysfs attribute file in the debug file system, by default mounted at `/sys/kernel/debug`.

You can turn data collection on and off. To switch off measurement data collecting for the current session, write "0" to the statistics attribute. To enable data collection again, write "1" to the statistics attribute.

**Example**

To read measurement data for a (RoCE) function named 0000:00:00.0 use:

```bash
# cat /sys/kernel/debug/pci/0000:00:00.0/statistics
```
The statistics attribute file might look similar to this example:

```
FMB @ 0000000078cd8000
Update interval: 4000 ms
Samples: 14373
Last update TOD: cefa44fa50006378
    Load operations:       1002780
    Store operations:      1950622
    Store block operations: 0
    Refresh operations:    0
    Received bytes:       0
    Received packets:     0
    Transmitted bytes:    0
    Transmitted packets:  0
    Allocated pages:      9104
    Mapped pages:         16633
    Unmapped pages:       2337
```
Part 6. z/VM virtual server integration

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
Chapter 39. z/VM concepts

The z/VM performance monitoring and cooperative memory management concepts help you to understand how the different components interact with Linux.

Performance monitoring for z/VM guest virtual machines

You can monitor the performance of z/VM guest virtual machines and their guest operating systems with performance monitoring tools on z/VM or on Linux.

These tools can be your own, IBM tools such as the Performance Toolkit for VM, or third-party tools. The guests being monitored require agents that write monitor data.

Monitoring on z/VM

z/VM monitoring tools must read performance data. For monitoring Linux instances, this data is APPLDATA monitor records.

Linux instances must write these records for the tool to read, as shown in Figure 139 on page 435.

Figure 139: 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_fssstatd. 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 z/VM supports writing and collecting performance data as follows:

- The Linux kernel can write z/VM monitor data for Linux instances, see Chapter 40, “Writing kernel APPLDATA records,” on page 439.
- Linux applications that are running on z/VM guests can write z/VM monitor data, see Chapter 41, “Writing z/VM monitor records,” on page 447.
- You can collect monitor file system size information, see “mon_fssstatd – Monitor z/VM guest file system size” on page 689.
• You can collect system information about up to 100 concurrently running processes, see “mon procd – Monitor Linux on z/VM” on page 694.

Monitoring on Linux

A Linux instance can read the monitor data by using the monreader device driver.

Figure 140 on page 436 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 140: Performance monitoring using monitor DCSS data

In summary, Linux on z/VM supports reading performance data in the form of read access to z/VM monitor data for Linux instances. See Chapter 42, “Reading z/VM monitor records,” on page 451 for more details.

Further information

Several z/VM publications include information about monitoring.

• See z/VM: Getting Started with Linux on System z, SC24-6287, the chapter on monitoring performance for information about using the CP Monitor and the Performance Toolkit for VM.

• See z/VM: Saved Segments Planning and Administration, SC24-6322 for general information about DCSSs (z/VM keeps monitor records in a DCSS).

• See z/VM: Performance, SC24-6301 for information about creating a monitor DCSS.

• See z/VM: CP Commands and Utilities Reference, SC24-6268 for information about the CP commands that are used in the context of DCSSs and for controlling the z/VM monitor system service.

• For the layout of the monitor records, visit www.ibm.com/vm/pubs/ctlblk.html and see Chapter 40, “Writing kernel APPLDATA records,” on page 439.

• 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 48, “Cooperative memory management,” on page 489.
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 141 on page 437.

**Figure 141: 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 142 on page 437).

**Figure 142: 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 143 on page 438). 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 143: 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*, SC24-6301.

Third-party tools can provide a Linux daemon that receives commands for the memory allocation through TCP/IP. The daemon, 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 *Linux on z Systems Troubleshooting*, SC34-2612.
Chapter 40. 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 441).

For an overview of performance monitoring support, see “Performance monitoring for z/VM guest virtual machines” on page 435.

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.

<table>
<thead>
<tr>
<th>Virtualization --&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux - VM Monitor Stream, base infrastructure</td>
</tr>
<tr>
<td>Monitor memory management statistics</td>
</tr>
<tr>
<td>Monitor OS statistics</td>
</tr>
<tr>
<td>Monitor overall network statistics</td>
</tr>
</tbody>
</table>

Figure 144: 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.

```
modprobe appldata_mem
modprobe appldata_os
modprobe appldata_net_sum
```

*Figure 145: APPLDATA record support module parameter syntax*

where `appldata_mem`, `appldata_os`, and `appldata_net_sum` are the modules for gathering memory-related data, operating system-related data, and network-related data.

See the `modprobe` man page for command details.

**Generating APPLDATA monitor records**

You can set the timer interval and enable or disable data collection.

You control the monitor stream support through the procfs. APPLDATA monitor records are produced if both a particular data-gathering module and the monitoring support in general are enabled.

**Enabling or disabling the support**

Use the procfs `timer` attribute to enable or disable the monitoring support.

**Procedure**

To read the current setting, issue:

```
# cat /proc/sys/appldata/timer
```

To enable the monitoring support, issue:

```
# echo 1 > /proc/sys/appldata/timer
```

To disable the monitoring support, issue:

```
# echo 0 > /proc/sys/appldata/timer
```

**Activating or deactivating individual data-gathering modules**

Each data-gathering module has a procfs entry that contains a value 1 if the module is active and 0 if the module is inactive.

**About this task**

The following procfs entries control the data-gathering modules:

```
/proc/sys/appldata/mem for the memory data-gathering module
/proc/sys/appldata/os for the CPU data-gathering module
/proc/sys/appldata/net_sum for the net data-gathering module
```

To check whether a module is active, look at the content of the corresponding procfs entry.

**Procedure**

Issue a command of this form:
# echo <flag> > /proc/sys/appldata/<data_type>

where <data_type> is one of mem, os, or net_sum.

**Note:** An active data-gathering module produces APPLDATA monitor records only if the monitoring support is enabled (see “Enabling or disabling the support” on page 440).

**Example**

To find out whether memory data-gathering is active, issue:

```
# cat /proc/sys/appldata/mem
0
```

In the example, memory data-gathering is off. To activate memory data-gathering, issue:

```
# echo 1 > /proc/sys/appldata/mem
```

To deactivate the memory data-gathering module, issue:

```
# echo 0 > /proc/sys/appldata/mem
```

**Setting the sampling interval**

You can set the time that lapses between consecutive data samples.

**About this task**

The time that you set is measured by the virtual CPU timer. Because the virtual timer slows down as the guest idles, the sampling interval in real time can be considerably longer than the value you set.

The value in `/proc/sys/appldata/interval` is the sample interval in milliseconds. The default sample interval is 10 000 ms.

**Procedure**

To read the current value, issue:

```
# cat /proc/sys/appldata/interval
```

To set the sample interval to a different value, write the new value (in milliseconds) to `/proc/sys/appldata/interval`. Issue a command of this form:

```
# echo <interval> > /proc/sys/appldata/interval
```

where <interval> is the new sample interval in milliseconds. The specification must be in the range 1 - 2147483647, where 2,147,483,647 = 2³¹ - 1.

**Example**

To set the sampling interval to 20 s (20000 ms), issue:

```
# echo 20000 > /proc/sys/appldata/interval
```

**APPLDATA monitor record layout**

Each of the data-gathering modules writes a different type of record.

- Memory data (see Table 56 on page 442)
z/VM can identify the records by their unique product ID. The product ID is an EBCDIC string of this form: "LINUXKRNL<record ID>260100". The <record ID> is treated as a byte value, not a string.

The records contain data of the following types:

- **u32**
  - unsigned 4-byte integer.
- **u64**
  - unsigned 8-byte integer.

<table>
<thead>
<tr>
<th>Offset (Decimal)</th>
<th>Offset (Hex)</th>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0</td>
<td>u64</td>
<td>timestamp</td>
<td>TOD time stamp that is generated on the Linux side after record update</td>
</tr>
<tr>
<td>8</td>
<td>0x8</td>
<td>u32</td>
<td>sync_count_1</td>
<td>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.</td>
</tr>
<tr>
<td>12</td>
<td>0xC</td>
<td>u32</td>
<td>sync_count_2</td>
<td>See sync_count_1.</td>
</tr>
<tr>
<td>16</td>
<td>0x10</td>
<td>u64</td>
<td>pggpin</td>
<td>Data that was read from disk (in KB)</td>
</tr>
<tr>
<td>24</td>
<td>0x18</td>
<td>u64</td>
<td>pggpout</td>
<td>Data that was written to disk (in KB)</td>
</tr>
<tr>
<td>32</td>
<td>0x20</td>
<td>u64</td>
<td>pswpin</td>
<td>Pages that were swapped in</td>
</tr>
<tr>
<td>40</td>
<td>0x28</td>
<td>u64</td>
<td>pswpout</td>
<td>Pages that were swapped out</td>
</tr>
<tr>
<td>48</td>
<td>0x30</td>
<td>u64</td>
<td>sharedram</td>
<td>Shared RAM in KB, set to 0</td>
</tr>
<tr>
<td>56</td>
<td>0x38</td>
<td>u64</td>
<td>totalram</td>
<td>Total usable main memory size in KB</td>
</tr>
<tr>
<td>64</td>
<td>0x40</td>
<td>u64</td>
<td>freeram</td>
<td>Available memory size in KB</td>
</tr>
<tr>
<td>72</td>
<td>0x48</td>
<td>u64</td>
<td>totalhigh</td>
<td>Total high memory size in KB</td>
</tr>
<tr>
<td>80</td>
<td>0x50</td>
<td>u64</td>
<td>freehigh</td>
<td>Available high memory size in KB</td>
</tr>
<tr>
<td>88</td>
<td>0x58</td>
<td>u64</td>
<td>bufferram</td>
<td>Memory that was reserved for raw disk blocks, corresponding to &quot;Buffers&quot; from /proc/meminfo, in KB</td>
</tr>
<tr>
<td>96</td>
<td>0x60</td>
<td>u64</td>
<td>cached</td>
<td>Size of used cache, including &quot;Cached&quot; and &quot;SwapCached&quot; from /proc/meminfo, in KB</td>
</tr>
<tr>
<td>104</td>
<td>0x68</td>
<td>u64</td>
<td>totalswap</td>
<td>Total swap space size in KB</td>
</tr>
<tr>
<td>112</td>
<td>0x70</td>
<td>u64</td>
<td>freeswap</td>
<td>Free swap space in KB</td>
</tr>
<tr>
<td>120</td>
<td>0x78</td>
<td>u64</td>
<td>pgalloc</td>
<td>Page allocations</td>
</tr>
<tr>
<td>128</td>
<td>0x80</td>
<td>u64</td>
<td>pfault</td>
<td>Page faults (major+minor)</td>
</tr>
<tr>
<td>136</td>
<td>0x88</td>
<td>u64</td>
<td>pfmajfault</td>
<td>Page faults (major only)</td>
</tr>
</tbody>
</table>
Table 57: APPLDATA_OS_DATA record (Record ID 0x02).

<table>
<thead>
<tr>
<th>Offset (Decimal)</th>
<th>Offset (Hex)</th>
<th>Type (size)</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0</td>
<td>u64</td>
<td>timestamp</td>
<td>TOD time stamp that is generated on the Linux side after record update</td>
</tr>
<tr>
<td>8</td>
<td>0x8</td>
<td>u32</td>
<td>sync_count_1</td>
<td>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.</td>
</tr>
<tr>
<td>12</td>
<td>0xC</td>
<td>u32</td>
<td>sync_count_2</td>
<td>See sync_count_1.</td>
</tr>
<tr>
<td>16</td>
<td>0x10</td>
<td>u32</td>
<td>nr_cpus</td>
<td>Number of virtual CPUs.</td>
</tr>
<tr>
<td>20</td>
<td>0x14</td>
<td>u32</td>
<td>per_cpu_size</td>
<td>Size of the per_cpu_data for each CPU (= 36).</td>
</tr>
<tr>
<td>24</td>
<td>0x18</td>
<td>u32</td>
<td>cpu_offset</td>
<td>Offset of the first per_cpu_data (= 52).</td>
</tr>
<tr>
<td>28</td>
<td>0x1C</td>
<td>u32</td>
<td>nr_running</td>
<td>Number of runnable threads.</td>
</tr>
<tr>
<td>32</td>
<td>0x20</td>
<td>u32</td>
<td>nr_threads</td>
<td>Number of threads.</td>
</tr>
<tr>
<td>36</td>
<td>0x24</td>
<td>3 × u32</td>
<td>avenrun[3]</td>
<td>Average number of running processes during the last 1 (first value), 5 (second value) and 15 (third value) minutes. These values are &quot;fake fix-point&quot;. Each value is composed of a 10-bit integer and an 11-bit fractional part. See note “1” on page 444 at the end of this table.</td>
</tr>
<tr>
<td>48</td>
<td>0x30</td>
<td>u32</td>
<td>nr_iowait</td>
<td>Number of blocked threads (waiting for I/O).</td>
</tr>
<tr>
<td>52</td>
<td>0x34</td>
<td>See note “2” on page 444, per_cpu_data</td>
<td>Time spent in user, kernel, idle, nice, etc for every CPU. See note “3” on page 444 at the end of this table.</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>0x34</td>
<td>u32</td>
<td>per_cpu_user</td>
<td>Timer ticks that were spent in user mode.</td>
</tr>
<tr>
<td>56</td>
<td>0x38</td>
<td>u32</td>
<td>per_cpu_nice</td>
<td>Timer ticks that were spent with modified priority.</td>
</tr>
<tr>
<td>60</td>
<td>0x3C</td>
<td>u32</td>
<td>per_cpu_system</td>
<td>Timer ticks that were spent in kernel mode.</td>
</tr>
<tr>
<td>64</td>
<td>0x40</td>
<td>u32</td>
<td>per_cpu_idle</td>
<td>Timer ticks that were spent in idle mode.</td>
</tr>
<tr>
<td>68</td>
<td>0x44</td>
<td>u32</td>
<td>per_cpu_irq</td>
<td>Timer ticks that were spent in interrupts.</td>
</tr>
<tr>
<td>72</td>
<td>0x48</td>
<td>u32</td>
<td>per_cpu_softirq</td>
<td>Timer ticks that were spent in softirqs.</td>
</tr>
<tr>
<td>76</td>
<td>0x4C</td>
<td>u32</td>
<td>per_cpu_iowait</td>
<td>Timer ticks that were spent while waiting for I/O.</td>
</tr>
<tr>
<td>80</td>
<td>0x50</td>
<td>u32</td>
<td>per_cpu_steal</td>
<td>Timer ticks &quot;stolen&quot; by the hypervisor.</td>
</tr>
<tr>
<td>84</td>
<td>0x54</td>
<td>u32</td>
<td>cpu_id</td>
<td>The number of this CPU.</td>
</tr>
</tbody>
</table>
Table 57: APPLDATA_OS_DATA record (Record ID 0x02). (continued)

<table>
<thead>
<tr>
<th>Offset (Decimal)</th>
<th>Offset (Hex)</th>
<th>Type (size)</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Note:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1. The following C-Macros are used inside Linux to transform these into values with two decimal places:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#define LOAD_INT(x) ((x) &gt;&gt; 11)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#define LOAD_FRAC(x) LOAD_INT(((x) &amp; ((1 &lt;&lt; 11) - 1)) * 100)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. nr_cpus * per_cpu_size</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3. per_cpu_user through cpu_id are repeated for each CPU</td>
</tr>
</tbody>
</table>

Table 58: APPLDATA_NET_SUM_DATA record (Record ID 0x03)

<table>
<thead>
<tr>
<th>Offset (Decimal)</th>
<th>Offset (Hex)</th>
<th>Type (size)</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0</td>
<td>u64</td>
<td>timestamp</td>
<td>TOD time stamp that is generated on the Linux side after record update</td>
</tr>
<tr>
<td>8</td>
<td>0x8</td>
<td>u32</td>
<td>sync_count_1</td>
<td>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.</td>
</tr>
<tr>
<td>12</td>
<td>0xC</td>
<td>u32</td>
<td>sync_count_2</td>
<td>See sync_count_1.</td>
</tr>
<tr>
<td>16</td>
<td>0x10</td>
<td>u32</td>
<td>nr_interfaces</td>
<td>Number of interfaces being monitored</td>
</tr>
<tr>
<td>20</td>
<td>0x14</td>
<td>u32</td>
<td>padding</td>
<td>Unused. The next value is 64-bit aligned, so these 4 bytes would be padded out by compiler</td>
</tr>
<tr>
<td>24</td>
<td>0x18</td>
<td>u64</td>
<td>rx_packets</td>
<td>Total packets that were received</td>
</tr>
<tr>
<td>32</td>
<td>0x20</td>
<td>u64</td>
<td>tx_packets</td>
<td>Total packets that were transmitted</td>
</tr>
<tr>
<td>40</td>
<td>0x28</td>
<td>u64</td>
<td>rx_bytes</td>
<td>Total bytes that were received</td>
</tr>
<tr>
<td>48</td>
<td>0x30</td>
<td>u64</td>
<td>tx_bytes</td>
<td>Total bytes that were transmitted</td>
</tr>
<tr>
<td>56</td>
<td>0x38</td>
<td>u64</td>
<td>rx_errors</td>
<td>Number of bad packets that were received</td>
</tr>
<tr>
<td>64</td>
<td>0x40</td>
<td>u64</td>
<td>tx_errors</td>
<td>Number of packet transmit problems</td>
</tr>
<tr>
<td>72</td>
<td>0x48</td>
<td>u64</td>
<td>rx_dropped</td>
<td>Number of incoming packets that were dropped because of insufficient space in Linux buffers</td>
</tr>
<tr>
<td>80</td>
<td>0x50</td>
<td>u64</td>
<td>tx_dropped</td>
<td>Number of outgoing packets that were dropped because of insufficient space in Linux buffers</td>
</tr>
<tr>
<td>88</td>
<td>0x58</td>
<td>u64</td>
<td>collisions</td>
<td>Number of collisions while transmitting</td>
</tr>
</tbody>
</table>

**Programming interfaces**

The monitor stream support base module exports two functions.

- appldata_register_ops() to register data-gathering modules
- appldata_unregister_ops() to undo the registration of data-gathering modules
Both functions receive a pointer to a struct appldata_ops as parameter. Additional data-gathering modules that want to plug into the base module must provide this data structure. You can find the definition of the structure and the functions in `arch/s390/appldata/appldata.h` in the Linux source tree.

See “APPLDATA monitor record layout” on page 441 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 41. 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 435.

The monitor stream application device driver interacts with the z/VM monitor APPLDATA facilities for performance monitoring. A better understanding of these z/VM facilities might help when you are using this device driver. See z/VM: Performance, SC24-6301 for information about monitor APPLDATA.

The monitor stream application device driver provides the following functions:

- An interface to the z/VM monitor stream.
- A means of writing z/VM monitor APPLDATA records.

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.

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.
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.

```
modprobe monwriter max_bufs=255
max_bufs = <numbufs>
```

*Figure 147: 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*, SC24-6268 for information about the CP MONITOR command.
Working with the z/VM *MONITOR record writer

The monitor stream application device driver uses the z/VM CP instruction DIAG X'DC' to write to the z/VM monitor stream. Monitor data must be preceded by a data structure, monwrite_hdr.

See z/VM: CP Programming Services, SC24-6272 for more information about the DIAG X'DC' instruction and the different monitor record types (sample, config, event).

The application writes monitor data by passing a monwrite_hdr structure that is followed by monitor data. The only exception is the STOP function, which requires no monitor data. The monwrite_hdr structure, as described in monwriter.h, is filled in by the application. The structure includes the DIAG X'DC' function to be performed, the product identifier, the header length, and the data length.

All records that are written to the z/VM monitor stream begin with a product identifier. This device driver uses the product ID. The product ID is a 16-byte structure of the form pppppppppffnnvrrmm, where:

pppppppp

is a fixed ASCII string, for example, LNXAPPL.

ff

is the application number (hexadecimal number). This number can be chosen by the application. 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:

```c
/* the header the app uses in its write() data */
struct monwrite_hdr {
    unsigned char mon_function;
    unsigned short applid;
    unsigned char record_num;
    unsigned short version;
    unsigned short release;
    unsigned short mod_level;
    unsigned short datalen;
    unsigned char hdrlen;
} __attribute__((packed));
```

The following function code values are defined:

```c
/* mon_function values */
#define MONWRITE_START_INTERVAL 0x00 /* start interval recording */
#define MONWRITE_STOP_INTERVAL  0x01 /* stop interval or config recording */
#define MONWRITE_GEN_EVENT      0x02 /* generate event record */
#define MONWRITE_START_CONFIG   0x03 /* start configuration recording */
```

Writing data 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 42. 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 435.

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 40, “Writing kernel APPLDATA records,” on page 439).

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, SC24-6322 for general information about DCSSs.
- See z/VM: Performance, SC24-6301 for information about creating a monitor DCSS.
- See z/VM: CP Commands and Utilities Reference, SC24-6268 for information about the CP commands that are used in the context of DCSSs and for controlling the z/VM monitor system service.
- For the layout of the monitor records, go to www.ibm.com/vm/pubs/ctlblk.html and click the link to the monitor record format for your z/VM version. Also, see Chapter 40, “Writing kernel APPLDATA records,” on page 439.
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.

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 451 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:
Assuring that the DCSS is addressable for your Linux instance

The DCSS address range must not overlap with the storage of your 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:

```
quickscore 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 470.

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.

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.

```
modprobe monreader

mondcss = MONDCSS
```

Figure 150: monitor stream support module parameter syntax

where `<dcss>` is the name of the DCSS that z/VM uses for the monitor records.

Example
To load the monitor read support module and specify MYDCSS as the DCSS issue:

```
modprobe monreader mondcss=mydcss
```

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” on page 455
- “Reading monitor records” on page 455

Opening and closing the character device

Only one user can open the character device at any one time. Once you have opened the device, you must close it to make it accessible to other users.

About this task

The open function can fail (return a negative value) with one of the following values for errno:

EBUSY
The device has already been opened by another user.

EIO
No IUCV connection to the z/VM MONITOR system service could be established. An error message with an IPUSER SEVER code is printed into syslog. See z/VM: Performance, SC24-6301 for details about the codes.

Once the device is opened, incoming messages are accepted and account for the message limit. If you keep the device open indefinitely, expect to eventually reach the message limit (with error code EOVERFLOW).

Reading monitor records

You can either read in non-blocking mode with polling, or you can read in blocking mode without polling.

About this task

Reading from the device provides a 12-byte monitor control element (MCE), followed by a set of one or more contiguous monitor records (similar to the output of the CMS utility MONWRITE without the 4 K control blocks). The MCE contains information about:

- The type of the following record set (sample/event data)
- The monitor domains contained within it
- The start and end address of the record set in the monitor DCSS

The start and end address can be used to determine the size of the record set. The end address is the address of the last byte of data. The start address is needed to handle "end-of-frame" records correctly (domain 1, record 13), that is, it can be used to determine the record start offset relative to a 4 K page (frame) boundary.

See "Appendix A: *MONITOR" in z/VM: Performance, SC24-6301 for a description of the monitor control element layout. The layout of the monitor records can be found on

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 453).

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 43. 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, SC24-6272.

Features

With the z/VM recording device driver, you can read from several CP services and collect records.

In particular, the z/VM recording device driver supports:

- Reading records from the CP error logging service, *LOGREC.
- Reading records from the CP accounting service, *ACCOUNT.
- Reading records from the CP diagnostic service, *SYMPTOM.
- Automatic and explicit record collection (see “Starting and stopping record collection” on page 460).

What you should know about the z/VM recording device driver

You can read records from different recording services, one record at a time.

The z/VM recording device driver is a character device driver that is grouped under the IUCV category of device drivers (see “Device categories” on page 7). There is one device for each recording service. The 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 59 on page 457 shows the mapping of names and minor numbers.

Table 59: Device names and minor numbers

<table>
<thead>
<tr>
<th>z/VM recording service</th>
<th>Standard device name</th>
<th>Minor number</th>
</tr>
</thead>
<tbody>
<tr>
<td>*LOGREC</td>
<td>logrec</td>
<td>0</td>
</tr>
<tr>
<td>*ACCOUNT</td>
<td>account</td>
<td>1</td>
</tr>
<tr>
<td>*SYMPTOM</td>
<td>symptom</td>
<td>2</td>
</tr>
</tbody>
</table>

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:

```bash
# 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:

```bash
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 457).
- `<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 59 on page 457) and assuming that the assigned major number is 254, you could create the device nodes like this:

```bash
# 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.

![Record structure diagram](image)

*Figure 151: Record structure*

Figure 151 on page 458 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](http://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_VMOGRDR 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_VMOGRDR)

Figure 152: 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” on page 460
• “Purging existing records” on page 461
• “Querying the z/VM recording status” on page 461
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 461).

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 461.
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:

```bash
# 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:

```bash
# 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:

  ```bash
  # 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:

  ```bash
  # 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.

```bash
# 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 029   LINUX31 INACTIVE
```
where the lines represent:

- The service
- The recording status
- The number of queued records
- The number of records that result in a message to the operator
- The guest that is or was connected to that service and the status of that connection

A detailed description of the QUERY RECORDING command can be found in the z/VM: CP Commands and Utilities Reference, SC24-6268.

### Opening and closing devices

You can open, read, and release the device. You cannot open the device multiple times. Each time the device is opened it must be released before it can be opened again.

#### About this task

You can use a device's autorecord attribute (see “Starting and stopping record collection” on page 460) to enable automatic record collection while a device is open.

You can use a device's autopurge attribute (see “Purging existing records” on page 461) 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:

   ```command
cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
   ```

   The results depend on the system, and look similar to the following example:

<table>
<thead>
<tr>
<th>RECORDING</th>
<th>COUNT</th>
<th>LMT</th>
<th>USERID</th>
<th>COMMUNICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EREP ON</td>
<td>00000000</td>
<td>002</td>
<td>EREP</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>ACCOUNT ON</td>
<td>00001812</td>
<td>020</td>
<td>DISKACNT</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>SYMPTOM ON</td>
<td>00000000</td>
<td>002</td>
<td>OPERSYMP</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>ACCOUNT OFF</td>
<td>00000000</td>
<td>020</td>
<td>LINUX31</td>
<td>INACTIVE</td>
</tr>
</tbody>
</table>

2. Open `/dev/account` with an appropriate application.

   This action connects the guest to the *ACCOUNT service and starts recording. The entry for *ACCOUNT on guest LINUX31 changes to ACTIVE and ON:

   ```command
cat /sys/bus/iucv/drivers/vmlogrdr/recording_status
   ```

<table>
<thead>
<tr>
<th>RECORDING</th>
<th>COUNT</th>
<th>LMT</th>
<th>USERID</th>
<th>COMMUNICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EREP ON</td>
<td>00000000</td>
<td>002</td>
<td>EREP</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>ACCOUNT ON</td>
<td>00001812</td>
<td>020</td>
<td>DISKACNT</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>SYMPTOM ON</td>
<td>00000000</td>
<td>002</td>
<td>OPERSYMP</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>ACCOUNT ON</td>
<td>00000000</td>
<td>020</td>
<td>LINUX31</td>
<td>ACTIVE</td>
</tr>
</tbody>
</table>

3. Switch autopurge and autorecord off:

   ```command
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 OFF 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 00000000 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 00000000 002 EREP ACTIVE
ACCOUNT ON 00001821 020 DISKACNT INACTIVE
SYMPTOM ON 00000000 002 OPERSYMP ACTIVE
ACCOUNT OFF 00000000 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 00000000 002 EREP ACTIVE
ACCOUNT ON 00001821 020 DISKACNT INACTIVE
SYMPTOM ON 00000000 002 OPERSYMP ACTIVE
ACCOUNT OFF 00000000 020 LINUX31 INACTIVE
```

Recording did not start, in the message logs you might find a message:

```
vmlogrdr: recording response: HCPCRC0887I 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
```
<table>
<thead>
<tr>
<th>Command</th>
<th>Count</th>
<th>Limit</th>
<th>User ID</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>EREP ON</td>
<td>0000000000</td>
<td>002</td>
<td>EREP</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>ACCOUNT ON</td>
<td>00001821</td>
<td>020</td>
<td>DISKACNT</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>SYMPTOM ON</td>
<td>0000000000</td>
<td>002</td>
<td>OPERSYM</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>ACCOUNT ON</td>
<td>0000000000</td>
<td>020</td>
<td>LINUX31</td>
<td>INACTIVE</td>
</tr>
</tbody>
</table>
Chapter 44. 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 739).

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/vmrd{}0.0.<device_number>`
- Punch: `/dev/vmpn{}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 153: 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 739).

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 45. 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.

Through DCSSs, you can implement shared read-write RAM disks for Linux instances that run as guests of the same z/VM system. Such RAM disks can provide shared file systems.

For information about DCSS, see z/VM: Saved Segments Planning and Administration, SC24-6322.

Removed information: References to execute-in-place technology have been removed because changed dependencies make this technology experimental.

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” on page 467).

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, SC24-6271 for more information about the
  NAMESAVE statement.

• The DCSS was defined with the LOADNSHR operand.
  See z/VM: CP Commands and Utilities Reference, SC24-6268 for information about the LOADNSHR
  operand.

  See “DCSS options” on page 468 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 475 for an example.

See z/VM: CP Commands and Utilities Reference, SC24-6268 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.

![Figure 154: DCSS kernel configuration menu option](image)

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>
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>
  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
... 0011 MONDCSS CPDCSS N/A 09000 097FF SC R 00003 N/A N/A
... 
```

If all DCSSs that you intend to access are located above the guest storage, you do not need to take any action.

Procedure
If any DCSS that you intend to access with your guest machine overlaps with the guest storage, redefine the guest storage. Define two or more discontiguous storage extents such that the storage gap with the lowest address range covers the address ranges of all your DCSSs.

Note:
- You cannot place a DCSS into a storage gap other than the storage gap with the lowest address range.
A z/VM guest that was defined with one or more storage gaps cannot access a DCSS above the guest storage.

From a CMS session, use the DEF STORE command to define your guest storage as discontiguous storage extents. Ensure that the storage gap between the extents covers all your DCSSs' address ranges. Issue a command of this form:

```
DEF STOR CONFIG 0.<storage_gap_begin> <storage_gap_end>..<storage above gap>
```

where:

- `<storage_gap_begin>` is the lower limit of the storage gap. This limit must be at or below the lowest address of the DCSS with the lowest address range.
  - Because the lower address ranges are needed for memory management functions, make the lower limit at least 128 MB. The lower limit for the DCSS increases with the total memory size. Although 128 MB is not an exact value, it is an approximation that is sufficient for most cases.
- `<storage_gap_end>` is the upper limit of the storage gap. The upper limit must be above the upper limit of the DCSS with the highest address range.
- `<storage above gap>` is the amount of storage above the storage gap. The total guest storage is `<storage_gap_begin>` + `<storage above gap>`.

All values can be suffixed with M to provide the values in megabyte. See z/VM: CP Commands and Utilities Reference, SC24-6268 for more information about the DEF STORE command.

**Example**

To make a DCSS that starts at 144 MB and ends at 152 MB accessible to a z/VM guest with 512 MB guest storage:

```
DEF STORE CONFIG 0.140M 160M.372M
```

This specification is one example of how a suitable storage gap can be defined. In this example, the storage gap covers 140 - 160 MB and, thus, the entire DCSS range. The total guest storage is 140 MB + 372 MB = 512 MB.

**Working with DCSS devices**

Typical tasks for working with DCSS devices include mapping DCSS representations in z/VM and Linux, adding and removing DCSSs, and accessing and updating DCSS contents.

- “Adding a DCSS device” on page 472
- “Listing the DCSSs that map to a particular device” on page 472
- “Finding the minor number for a DCSS device” on page 473
- “Setting the access mode” on page 473
- “Saving updates to a DCSS or set of DCSSs” on page 474
- “Workaround for saving DCSSs with optional properties” on page 475
- “Removing a DCSS device” on page 475
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 470).
- 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".
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 467).

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 474).

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 467).

For either access mode the changes are volatile until they are saved (see “Saving updates to a DCSS or set of DCSSs” on page 474).

**Procedure**

Issue a command of this form:

```bash
# 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”:

```bash
# cat /sys/devices/dcssblk/MYDCSS/shared
```

1 means that the current access mode is shared. To set the access mode to exclusive-writable, issue:

```bash
# 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 468), see “Workaround for saving DCSSs with optional properties” on page 475.
- If you use the watchdog device driver, turn off the watchdog before saving updates to DCSSs. Saving updates to DCSSs can result in a watchdog timeout if the watchdog is active.
- Do not place save requests before you have accessed the DCSS device.

**Procedure**

Issue a command of this form:

```bash
# 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”: 
Workaround for saving DCSSs with optional properties

If you need a DCSS that is defined with special options, you must use a workaround to save the DCSSs.

Before you begin

Important: This section applies to DCSSs with special options only. The workaround in this section is error-prone and requires utmost care. Erroneous parameter values for the described CP commands can render a DCSS unusable. Use this workaround only if you really need a DCSS with special options.

Procedure

Perform the following steps to save a DCSS with optional properties:

1. Unmount the DCSS.

   Example: Enter this command to unmount a DCSS with the device node /dev/dcssblk0:

   ```
   # umount /dev/dcssblk0
   ```

2. Use the CP DEFSEG command to newly define the DCSS with the required properties.

   Example: Enter this command to newly define a DCSS, mydcss, with the range 80000-9ffff, segment type sr, and the loadnshr operand:

   ```
   # vmcp defseg mydcss 80000-9ffff sr loadnshr
   ```

   Note: If your DCSS device maps to multiple DCSSs as defined to z/VM, you must perform this step for each DCSS. Be sure to specify the command correctly with the correct address ranges and segment types. Incorrect specifications can render the DCSS unusable.

3. Use the CP SAVESEG command to save the DCSS.

   Example: Enter this command to save a DCSS mydcss:

   ```
   # vmcp saveseg mydcss
   ```

   Note: If your DCSS device maps to multiple DCSSs as defined to z/VM, you must perform this step for each DCSS. Omitting this step for individual DCSSs can render the DCSS device unusable.

Reference

See z/VM: CP Commands and Utilities Reference, SC24-6268 for details about the DEFSEG and SAVESEG CP commands.

Removing a DCSS device

Use the remove sysfs attribute to remove a DCSS from Linux.

Before you begin

A DCSS device can 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 on page 476. 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:
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 46. 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 737)

Differences between vmcp and a 3270 or 3215 console

Most CP commands behave identically with vmcp and on a 3270 or 3215 console. However, some commands show a different behavior:

• Diagnose X'08' (see z/VM: CP Programming Services, SC24-6272) requires you to specify a response buffer with the command. Because the response size is not known in advance, the default response buffer of vmcp might be too small and the response truncated.

• On a 3270 or 3215 console, the CP command is executed on virtual CPU 0. The vmcp device driver uses the CPU that is scheduled by the Linux kernel. For CP commands that depend on the CPU number (like trace), specify the CPU, for example: cpu 3 trace count.

• Some CP commands do not return specific error or status messages through diagnose X'08'. These messages are 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, SC24-6268.

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

Control the build options for the z/VM CP interface 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.
Device Drivers --->
  ... Character devices --->
    ... S/390 character device drivers (depends on S390) --->
    ... Support for the z/VM CP interface (CONFIG_VMCP)
    Memory in MiB reserved for z/VM CP interface (CONFIG_VMCP_CMA_SIZE)

Figure 157: Kernel configuration menu options for the z/VM CP interface

**CONFIG_VMCP**

is required to build a kernel with user space access to CP commands.

**CONFIG_VMCP_CMA_SIZE**

specifies the default amount of contiguous memory, in MiB, to be reserved for large responses from z/VM CP. CONFIG_VMCP_CMA_SIZE is preset to 4 MiB.

When booting Linux, the `vmcp_cma=` kernel parameter can override the value that is set with CONFIG_VMCP_CMA_SIZE (see “Reserving contiguous memory” on page 480).

---

**Reserving contiguous memory**

z/VM CP might return large responses that require sufficient contiguous memory.

The default amount of contiguous memory that is reserved for the z/VM CP interface is set through the CONFIG_VMCP_CMA_SIZE kernel build option (see “Building a kernel with the z/VM CP interface” on page 479). You can override this default with the `vmcp_cma=` kernel parameter.

```
vmcp_cma = <size>
```

*Figure 158: z/VM CP interface kernel parameter syntax*

where `<size>` is the amount, in byte, of contiguous memory to be reserved. Valid values are integers, optionally with a suffix M for MiB or G for GiB.

**Example:** The following specification reserves 6 MiB of contiguous memory:

```
vmcp_cma=6M
```

---

**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.
**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>
<thead>
<tr>
<th>Name</th>
<th>Code definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMCP_GETCODE</td>
<td>_IOR (0x10, 1, int)</td>
<td>Queries the return code of z/VM.</td>
</tr>
<tr>
<td>VMCP_SETBUF</td>
<td>_IOW(0x10, 2, int)</td>
<td>Sets the buffer size (the device driver has a default of 4 KB; vmcp calls this ioctl to set it to 8 KB instead).</td>
</tr>
<tr>
<td>VMCP_GETSIZE</td>
<td>_IOR(0x10, 3, int)</td>
<td>Queries the size of the response.</td>
</tr>
</tbody>
</table>
Chapter 47. 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 159 on page 483).

Figure 159: 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 160 on page 484 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 160: 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 4, “Kernel and module parameters,” on page 23 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.sender](<user_ID>)

*Figure 161: smsgiucv_app kernel parameter syntax*

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 487).
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.

```
modprobe smsgiucv_app
sender = <user_ID>
```

*Figure 162: 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 487).

Working with CP special messages

You might have to send, access, or respond to CP special messages.

- “Sending CP special messages” on page 485
- “Accessing CP special messages through uevent environment variables” on page 485
- “Writing udev rules for handling CP special messages” on page 486

Sending CP special messages

Issue a CP SMSG command from a CP or CMS session or from Linux to send a CP special message.

**Procedure**

To send a CP special message to LXGUEST1 from Linux, enter a command of the following form:

```
# vmcp SMSG LXGUEST1 APP "<message text>"
```

To send a CP special message to LXGUEST1, enter the following command from a CP or CMS session:

```
#CP SMSG LXGUEST1 APP <message text>
```

The special messages cause uevents to be generated. See “Writing udev rules for handling CP special messages” on page 486 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.

**SMGS_ID**

Specifies the message prefix. The `SMGS_ID` environment variable is always set to APP, which is the prefix that is assigned to the smsgiucv_app device driver.

**SMGS_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.

**SMSS_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
SMSS_SENDER=MAINT
SMSS_ID=APP
SMSS_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 485).

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:

```bash
# 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 -t -N LINUX.DMESG -u $env{SMSG_SENDER}''
```
Chapter 48. 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 436.

You can also use the cpuplugd command to define rules for cmm behavior, see “cpuplugd - Control CPUs and memory” on page 606.

For information about setting up the external resource manager, see the chapter on VMRM in z/VM: Performance, SC24-6301.

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 163 on page 489 shows option CONFIG_CMM in the configuration menu. It also shows the options that you need in addition if you are using VMRM.

---

**Figure 163: 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.

```
cmm.sender = VMRMSVM

```

Figure 164: Cooperative memory management kernel 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.

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.

```
modprobe cmm sender=VMRMSVM

```

Figure 165: 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:

```bash
# 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:

```bash
# cat /proc/sys/vm/cmm_timed_pages
```
Part 7. Security

These device drivers and features support security aspects of Linux on 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
Chapter 49. Generic cryptographic device driver

The generic cryptographic device driver (zcrypt) supports cryptographic coprocessor and accelerator hardware. Cryptographic coprocessors provide secure key cryptographic operations for the IBM Common Cryptographic Architecture (CCA) 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 CEX6, CEX5, and CEX4 cryptographic adapters can also be configured as EP11 coprocessors.

The following types of cryptographic adapters are supported:

- Crypto Express6S (EP11) Coprocessor (CEX6P)
- Crypto Express6S (CCA) Coprocessor (CEX6C)
- Crypto Express6S Accelerator (CEX6A)
- Crypto Express5S Accelerator (CEX5A)
- Crypto Express5S (CCA) Coprocessor (CEX5C)
- Crypto Express5S (EP11) Coprocessor (CEX5P)
- Crypto Express4S Accelerator (CEX4A)
- Crypto Express4S (CCA) Coprocessor (CEX4C)
- Crypto Express4S (EP11) Coprocessor (CEX4P)
- Crypto Express3 Accelerator (CEX3A)
- Crypto Express3 Coprocessor (CEX3C)
- Crypto Express2 Accelerator (CEX2A)
- Crypto Express2 Coprocessor (CEX2C)

For information about setting up your cryptographic environment on Linux under z/VM, see z/VM: Secure Configuration Guide, SG24-6323 and Security for Linux on System z, SG24-7728.

Supported facilities

The cryptographic device driver supports several cryptographic accelerators as well as CCA and EP11 coprocessors.

Cryptographic accelerators support clear key cryptographic algorithms. In particular, they provide fast RSA encryption and decryption for any key size in the range 57 - 4096 bit (for CEX2A the range is 57 - 2048 bit).

Cryptographic coprocessors act as a hardware security module (HSM) and provide secure key cryptographic operations for the IBM Common Cryptographic Architecture (CCA) and the Enterprise PKCS#11 feature (EP11).
Cryptographic CCA coprocessors also provide clear key RSA operations for any key size in the range 57 - 4096 bit, and a random number generator for /dev/hwrng. The EP11 coprocessor supports only secure key operations.


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.

Hardware and software prerequisites

Support for the Crypto Express6S, Crypto Express5S, Crypto Express4S, Crypto Express3, and Crypto Express2 features depends on the IBM Z hardware model.

Table 61 on page 496 lists the support for the cryptographic adapters.

<table>
<thead>
<tr>
<th>Cryptographic adapters</th>
<th>Mainframe support</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEX6A, CEX6C, and CEX6P</td>
<td>z14 and z14 ZR1</td>
</tr>
<tr>
<td>CEX5A, CEX5C, and CEX5P</td>
<td>z14, z13 and z13s</td>
</tr>
<tr>
<td>CEX4A, CEX4C, and CEX4P</td>
<td>zEC12 and zBC12</td>
</tr>
<tr>
<td>CEX3A and CEX3C</td>
<td>• zEC12 and zBC12</td>
</tr>
<tr>
<td></td>
<td>• z196 and z114</td>
</tr>
<tr>
<td></td>
<td>• System z10 (as of October 2009)</td>
</tr>
<tr>
<td>CEX2A and CEX2C</td>
<td>System z10 and System z9</td>
</tr>
</tbody>
</table>

Table 62 on page 496 lists the required software by function.

<table>
<thead>
<tr>
<th>Software required</th>
<th>Function that is supported by the software</th>
</tr>
</thead>
<tbody>
<tr>
<td>The CCA library</td>
<td>For the secure key cryptographic functions on CCA coprocessors. For information about cryptographic CCA coprocessors, coexistence of adapter versions, and how to use CCA functions, see Secure Key Solution with the Common Cryptographic Architecture Application Programmer’s Guide, SC33-8294. You can obtain this publication at <a href="http://www.ibm.com/support/knowledgecenter/linuxonibm/iaaf/lnz_r_cca.html">www.ibm.com/support/knowledgecenter/linuxonibm/iaaf/lnz_r_cca.html</a></td>
</tr>
<tr>
<td>The libica library</td>
<td>For the clear key cryptographic functions. See libica Programmer’s Reference, SC34-2602. You can obtain this publication at <a href="http://www.ibm.com/developerworks/linux/linux390/documentation_dev.html">www.ibm.com/developerworks/linux/linux390/documentation_dev.html</a>.</td>
</tr>
<tr>
<td>APAR VM65942</td>
<td>To support z14 and z14 ZR1 hardware and the CEX6S adapter for Linux on z/VM. To support ECC with a shared cryptographic adapter in coprocessor mode for Linux on z/VM 6.4.</td>
</tr>
</tbody>
</table>
What you should know about the cryptographic device driver

Your use of the cryptographic device driver and the cryptographic hardware might require additional software. There are special considerations for Linux on z/VM, for performance, and for specific cryptographic operations.

Functions provided by the cryptographic device driver

The functions that are provided by the cryptographic device driver depend on whether the device driver finds a cryptographic accelerator, a CCA coprocessor, or an EP11 coprocessor.

For both accelerators and CCA coprocessors, 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 for any key size in the range 57 - 4096 bit.

For CCA coprocessors, 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.

For EP11 coprocessors, the device driver provides functionality to pass EP11 requests to the cryptographic coprocessor.

Adapter discovery

The cryptographic device driver provides two misc device nodes, one for cryptographic requests, and one for a device from which random numbers can be read.

Cryptographic adapters are detected automatically when the device driver is started. 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. If hardware interrupt support is available, the device driver does not use polling to detect request completion.

All requests to either of the two misc devices are routed to a cryptographic adapter using a crypto request scheduling function that, for each adapter, takes into account:

- The supported functions
- The number of pending requests
- A speed rating

The cryptographic device driver supports multiple cryptographic domains simultaneously, see “Cryptographic domains” on page 498.

The device driver uses at least one domain for all adapters. If none is given, the kernel selects a default domain. Alternatively, you can select the default domain using a kernel parameter (see “Kernel parameters” on page 500).
**Cryptographic domains**

Crypto Express hardware adapters, coprocessors or accelerators, are divided into multiple domains, also called cryptographic domains or AP domains.

Each domain acts as an independent cryptographic device (HSM) with its own state, including its own master key. Two domains in the same Crypto Express adapter are completely isolated and cannot access each other’s states. The maximum number of domains depends on the mainframe model and is the same for all Crypto Express adapters in that mainframe. For example, a z13 supports up to 85 domains (with hexadecimal domain IDs 0000 to 0054).

**Cryptographic devices on LPARs**

When you assign adapters and domains to an LPAR on the HMC or SE, you indirectly assign virtual cryptographic devices.

For example, assigning adapter ID 00 and 02 as well as domains 0002, 0003, and 0005 to an LPAR implicitly assigns six virtual cryptographic devices to the LPAR: (00,0002), (00,0003), (00,0005), (02,0002), (02,0003), and (02,0005).

You can choose between two types of access to a cryptographic domain:

**To use cryptographic functions.**

A domain that is assigned to an LPAR for usage access is called a usage domain of that LPAR on the HMC or SE.

**To manage or control the domain, including the management of the master keys.**

A domain that is assigned to an LPAR for management (control) access is called a control domain of that LPAR on the HMC or SE.

Every usage domain of an LPAR must also be a control domain of that LPAR.

The list of usage domains and the list of adapter IDs define the list of virtual cryptographic devices that are assigned to an LPAR. For example, if 00 is an adapter ID and 0002 is a usage domain ID, then the virtual cryptographic device (00,0002) is assigned to the LPAR.
Cryptographic devices on z/VM

In z/VM, the virtual cryptographic devices available to a guest are defined by using the CRYPTO directory statement:

- The CRYPTO APDEDICATE statement assigns domain IDs and adapter IDs to the guest. This assignment implicitly defines a list of dedicated virtual cryptographic devices. All virtual cryptographic devices that are determined by an ID from the adapter list of that guest and an ID from the domain list of that guest are assigned to the guest.
- The CRYPTO APVIRT statement assigns one virtual cryptographic device that can be shared among multiple guests with a guest-specific virtualized adapter ID and a virtualized domain ID.

Virtual cryptographic devices in z/VM can be either shared or dedicated, but not both.

Linux on z/VM with access to a shared cryptographic accelerator can either observe an accelerator or a CCA coprocessor.

For shared cryptographic CCA coprocessors, the following functions are available to the Linux instance:

- Random number functions
- Clear-key RSA functions
- If APAR VM65942 has been installed: Clear-key ECC functions

Other requests are rejected by z/VM. For more information about supported functions, see the z/VM publications.

Cryptographic devices on Linux

In Linux, virtual cryptographic devices are called AP queues. The name of an AP queue consists of two parts, the adapter ID and the domain ID, both in hexadecimal notation. For example, if cryptographic adapters with the IDs 00 and 02 are selected, and the domains IDs 0002, 0003 and 0005 have been configured on the cryptographic adapter, then the following AP queues are defined to Linux:

/sys/devices/ap/card00/00.0002
/sys/devices/ap/card00/00.0003
/sys/devices/ap/card00/00.0005
/sys/devices/ap/card02/02.0002
/sys/devices/ap/card02/02.0003
/sys/devices/ap/card02/02.0005

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 s390 cryptographic adapters (CONFIG_ZCRYPT)
```

*Figure 166: zcrypt kernel configuration menu option*

You can compile the entire support for s390 cryptographic adapters into the kernel or you can specify 'm' for separate modules. Regardless of this choice, the base AP bus support is compiled into the kernel.
Specifying 'm' results in the following separate modules for specific adapter or function support:

- **zcrypt**
  request dispatcher, secure key messages, and clear key messages module. Loads the *rng_core* module.
- **zcrypt_cex4**
  device driver for all CEX4, CEX5, and CEX6 adapters.
- **zcrypt_cex2a**
  device driver for CEX2A, and CEX3A adapters.
- **zcrypt_cex2c**
  device driver for CEX2C and CEX3C adapters.

### Setting up the cryptographic device driver

Configure the cryptographic device driver through the `domain=` and the `poll_thread=` kernel 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

You configure the cryptographic device driver, zcrypt, by adding parameters to the kernel parameter line.

![Diagram showing zcrypt kernel parameter syntax](image)

where

- **<domain_id>**
  is an integer that identifies the default cryptographic domain for the Linux instance. You define cryptographic domains in the LPAR activation profile on the HMC or SE.

  The default value (domain=autoselect) causes the device driver to choose one of the available domains automatically.

  **Important:** Be sure to enter an existing domain. The Trusted Key Entry (TKE) workstation does not find the cryptographic adapters if a non-existing domain is entered here. All CCA applications use the default domain, and do not work correctly if the specified domain does not exist.
**<poll_thread>**
is an integer argument and enables a polling thread to tune cryptographic performance. Valid values are 1 (enabled) or 0 (disabled, this value is the default). For details, see “Setting the polling thread” on page 505.

**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 506.

**apmask= and aqmask=**
are two 256-bit masks that specify AP queues to be ignored by the zcrypt device driver, and remain available for alternative device drivers.

Each bit of the apmask= mask addresses a cryptographic adapter. The leftmost bit corresponds to the adapter with ID 0x00. Generally, the bit number in the mask corresponds to the decimal value of the adapter ID. If an adapter bit is set to 0, all AP queues for this adapter are available for alternative device drivers, across all domains.

Each bit of the aqmask= mask addresses a cryptographic domain. The leftmost bit corresponds to domain 0x0000. Generally, the bit number in the mask corresponds to the decimal value of the domain ID. If a domain bit is set to 0, all AP queues with this domain are available for alternative device drivers, across all adapters.

Generally, AP queues are available for alternative device drivers if the bit for its adapter, or its domain, or both are set to 0. By default, all bits in both masks are 1 and hence, all AP queues are handled by the zcrypt device driver.

**<mask>**
specifies a replacement for the default mask. Valid values are 0x followed by 1 - 64 hexadecimal digits. If fewer than 64 digits are specified, the specification is padded with 0s on the right. The value is big-endian.

**<bit>**
specifies an individual bit number. 0 specifies the leftmost bit. With the plus sign (+) prefix, the bit is set to 1. With the minus sign (-) prefix, the bit is set to 0.

**<from_bit>-<to_bit>**
specifies the range of bits from bit number <from_bit> to bit number <to_bit>. With the plus sign (+) prefix, all bits in the range are set to 1. With the minus sign (-) prefix, all bits in the range are set to 0.

You can specify a comma-separated list of bits and ranges. Such lists are processed left to right.

You can change the masks on a running Linux instance, see “Marking AP queues for alternative device drivers” on page 508.

**Examples**

- The following kernel parameter line specification makes the zcrypt device driver operate within the default cryptographic domain "7" with poll_thread enabled:

  ```
  domain=7 poll_thread=1
  ```

- The following specification marks all AP queues on adapters 0x00 and any adapters with IDs greater than 0x07 and all AP queues with domain ID, 0x0005, 0x0006, 0x0008, 0x0009, and 0x000a for use by alternate device drivers.

  ```
  apmask=0x7F aqmask=-5,-a,+7
  ```

**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>
```

## Working with cryptographic devices

Typically, cryptographic devices are not directly accessed by users but through user programs. Some tasks can be performed through the sysfs interface.

- “Displaying information about cryptographic devices and AP queues” on page 502
- “Setting devices online or offline” on page 504
- “Setting the polling thread” on page 505
- “Using AP adapter interrupts” on page 506
- “Setting the polling interval” on page 506
- “Dynamically adding and removing cryptographic adapters” on page 507
- “Marking AP queues for alternative device drivers” on page 508
- “Displaying information about the AP bus” on page 508

## Displaying information about cryptographic devices and AP queues

Use the lszcrypt command to display status information about your cryptographic devices and AP queues; alternatively, you can use sysfs.

### About this task

For information about lszcrypt, see “lszcrypt - Display zcrypt devices” on page 678.

Each cryptographic adapter is represented in sysfs as a device with a directory of the form

```
/sys/bus/ap/devices/card<XX>
```

where <XX> is the two-digit hexadecimal device index for each device. For example, device 0x1a can be found under /sys/bus/ap/devices/card1a. The sysfs directory contains a number of attributes with information about the AP queue.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ap_functions</td>
<td>Read-only attribute that represents the function facilities that are installed on this adapter.</td>
</tr>
<tr>
<td>depth</td>
<td>Read-only attribute that represents the input queue length for this adapter.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hwtype</td>
<td>Read-only attribute the represents the numeric hardware type as interpreted by the device driver. The following values are defined:</td>
</tr>
<tr>
<td></td>
<td>6 CEX2A adapters.</td>
</tr>
<tr>
<td></td>
<td>7 CEX2C adapters.</td>
</tr>
<tr>
<td></td>
<td>8 CEX3A adapters.</td>
</tr>
<tr>
<td></td>
<td>9 CEX3C adapters.</td>
</tr>
<tr>
<td></td>
<td>10 CEX4A, CEX4C, or CEX4P adapters.</td>
</tr>
<tr>
<td></td>
<td>11 CEX5A, CEX5C, or CEX5P adapters.</td>
</tr>
<tr>
<td></td>
<td>12 CEX6A, CEX6C, or CEX6P adapters.</td>
</tr>
<tr>
<td></td>
<td>The hwtype attribute shows the hardware type as interpreted by the device driver. For example, any cryptographic adapter later than CEX6 might be shown as a CEX6 adapter (type 12). See also the raw_hwtype attribute.</td>
</tr>
<tr>
<td>raw_hwtype</td>
<td>Read-only attribute that represents the actual hardware type of the cryptographic adapter.</td>
</tr>
<tr>
<td>modalias</td>
<td>Read-only attribute that represents an internally used device bus-ID.</td>
</tr>
<tr>
<td>online</td>
<td>Read-write attribute that shows whether the device is online (1) or offline (0).</td>
</tr>
<tr>
<td>pendingq_count</td>
<td>Read-only attribute that represents the number of requests in the hardware queue.</td>
</tr>
<tr>
<td>request_count</td>
<td>Read-only attribute that represents the number of requests that are already processed by this device.</td>
</tr>
<tr>
<td>requestq_count</td>
<td>Read-only attribute that represents the number of outstanding requests (not including the requests in the hardware queue).</td>
</tr>
<tr>
<td>type</td>
<td>Read-only attribute with a name for the device type. The following types are defined:</td>
</tr>
<tr>
<td></td>
<td>• CEX2C, CEX2A</td>
</tr>
<tr>
<td></td>
<td>• CEX3A, CEX3C</td>
</tr>
<tr>
<td></td>
<td>• CEX4A, CEX4C, CEX4P</td>
</tr>
<tr>
<td></td>
<td>• CEX5A, CEX5C, CEX5P</td>
</tr>
<tr>
<td></td>
<td>• CEX6A, CEX6C, CEX6P</td>
</tr>
</tbody>
</table>

Each AP queue is independently configurable and represented in a subdirectory of the cryptographic device it belongs to:

```
/sys/bus/ap/devices/card<XX>/<XX>.<YYYY>
```
where <XX> is the adapter ID of the cryptographic device and <YYYY> is the domain. For example, a cryptographic device with adapter ID 1a might have domains 5 (0005), 31 (001f), and 77 (004d) configured. The cryptographic device together with its AP queues would be represented in sysfs as:

```
/sys/devices/ap/card1a
/sys/devices/ap/card1a/1a.0005
/sys/devices/ap/card1a/1a.001f
/sys/devices/ap/card1a/1a.004d
```

Actions that you take on the cryptographic device also apply to its associated AP queues. Attributes like type and hwtype are inherited by the AP queues. The sysfs directory contains a number of attributes with information about the AP queues.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>online</td>
<td>Read-write attribute that shows whether the AP queue is online (1) or offline (0).</td>
</tr>
<tr>
<td>interrupt</td>
<td>Read-only attribute that represents the interrupt state (enabled or disabled) of the AP queue, and hence the request queue.</td>
</tr>
<tr>
<td>reset</td>
<td>Read-only attribute that indicate the state of pending resets of the AP queues, and hence the request queue.</td>
</tr>
<tr>
<td>pendingq_count</td>
<td>Read-only attribute that represents the number of requests in the hardware queue.</td>
</tr>
<tr>
<td>request_count</td>
<td>Read-only attribute that represents the number of requests that are already processed by this AP queue.</td>
</tr>
<tr>
<td>requestq_count</td>
<td>Read-only attribute that represents the number of outstanding requests (not including the requests in the hardware queue).</td>
</tr>
</tbody>
</table>

To display status information about your cryptographic devices and AP queues, you can also use the `lszcrypt` command (see “lszcrypt - Display zcrypt devices” on page 678).

**Setting devices online or offline**

Use the `chzcrypt` command to set cryptographic devices and AP queues online or offline.

**Procedure**

- Preferably, use the `chzcrypt` command with the `-e` option to set cryptographic devices and AP queues online, or use the `-d` option to set them offline.

**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 and their associated AP queues offline issue:

  ```
  # chzcrypt -d -a
  ```

For more information about `chzcrypt`, see “chzcrypt - Modify the zcrypt configuration” on page 587.

- Alternatively, write 1 to the `online` sysfs attribute of a cryptographic device to set the device online, or write 0 to set the device offline.

**Examples:**
– To set a cryptographic device with adapter ID 0x3e online issue:

```bash
# echo 1 > /sys/bus/ap/devices/card3e/online
```

– To set a cryptographic device with adapter ID 0x3e offline issue:

```bash
# echo 0 > /sys/bus/ap/devices/card3e/online
```

– To check the online status of the cryptographic device with adapter ID 0x3e issue:

```bash
# 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 506. 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:
    ```bash
    # chzcrypt -p
    ```
  
  - To deactivate the polling thread issue:
    ```bash
    # chzcrypt -n
    ```

  For more information about `chzcrypt`, see “chzcrypt - Modify the zcrypt configuration” on page 587.

- Alternatively, you can set the polling thread through the `poll_thread` sysfs attribute.

  This read-write attribute can be found at the AP bus level.

  **Examples:**

  - To activate the polling thread issue:
    ```bash
    # echo 1 > /sys/bus/ap/poll_thread
    ```
  
  - To deactivate the polling thread issue:
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 initialization, the zcrypt device driver checks whether AP adapter interrupts are supported by the hardware. If so, polling is disabled and the interrupt mechanism is automatically used.

To query whether AP adapter interrupts are used, read the sysfs attribute interrupt of the device. Another interrupt attribute at the AP bus level, /sys/bus/ap/ap_interrupt, indicates that the AP bus is able to handle interrupts.

Example
To read the interrupt attribute for a device 0x3e with domain 0xf, issue:

```
# cat /sys/bus/ap/devices/card3e/3e.000f/interrupt
```

If interrupts are used, the attribute shows "Interrupts enabled", otherwise "Interrupts disabled".

Setting the polling interval
Request polling is supported at nanosecond intervals.

Procedure
- Use the lszcrypt and chzcrypt commands to read and set the polling time.

Examples:
- To find out the current polling time, issue:

```
# lszcrypt -b
... poll_timeout=250000 (nanoseconds)
```
- To set the polling time to 1 microsecond, issue:

```
# chzcrypt -t 1000
```

For more information about lszcrypt and chzcrypt, see “lszcrypt - Display zcrypt devices” on page 678 and “chzcrypt - Modify the zcrypt configuration” on page 587.
- Alternatively, you can set the polling time through the poll_timeout sysfs attribute. This read-write attribute can be found at the AP bus level.

Examples:
- To read the poll_timeout attribute for the ap bus issue:

```
# cat /sys/bus/ap/poll_timeout
```
- To set the poll_timeout attribute for the ap bus to poll, for example, every microsecond, issue:

```
# echo 1000 > /sys/bus/ap/poll_timeout
```
Dynamically adding and removing cryptographic adapters

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.

By default, the device driver scans the cryptographic configuration every 30 seconds. Newly detected cryptographic adapters are set online. If an adapter has disappeared, its associated cryptographic device is removed. Read or modify the expiration time of this rescan interval with the `lszcrypt` and `chzcrypt` commands.

For more information about `lszcrypt` and `chzcrypt`, see “`lszcrypt - Display zcrypt devices`” on page 678 and “`chzcrypt - Modify the zcrypt configuration`” on page 587.

Adding or removing of cryptographic adapters to or from an LPAR is transparent to applications that use clear key functions. If a cryptographic adapter is removed while cryptographic requests are being processed, 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/support/knowledgecenter/linuxonibm/liaaf/lnz_r_cca.html.

Alternatively, you can read or set the configuration timer through the `config_time` sysfs attribute. This read-write attribute can be found at the AP bus level. Valid values for the `config_time` sysfs attribute are in the range 5 - 120 seconds.

For the secure key cryptographic functions on EP11 coprocessors, see *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713. You can obtain it at www.ibm.com/developerworks/linux/linux390/documentation_dev.html.

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
  ```

### Marking AP queues for alternative device drivers

You can mark a set of AP queues to be ignored by the zcrypt device driver, and so remain available for alternative device drivers.

#### Before you begin

**Attention:**

Do not change the settings for adapters or domains that are in use or reserved for another exploiter. In particular, do not mark adapters or queues for the zcrypt device driver while they are assigned to a KVM guest.

#### About this task

You can mark AP queues by domain and by adapter.

**Adapter mask**

The adapter mask is a 256-bit value, each bit representing a cryptographic adapter. The leftmost bit represents the adapter with ID 0x00. In sysfs, the mask is available as the value of attribute `/sys/bus/ap/apmask`. If an adapter bit is set to 0, all AP queues on this adapter are available for alternative device drivers, across domains.

**Domain mask**

The domain mask is a 256-bit value, each bit representing a cryptographic domain. The leftmost bit represents the domain with ID 0x0000. In sysfs, the mask is available as the value of attribute `/sys/bus/ap/aqmask`. If a domain bit is set to 0, all AP queues with this domain are available for alternative device drivers, across adapters.

The sysfs representation of both masks is a big-endian, 64-bit, hexadecimal value. For example, if the bit for adapter 0x00 is 1 and the bits for adapters 0x00 to 0x03 are 0, the leftmost hexadecimal digit in the adapter mask is 8.

zcrypt handles all AP queues for which both the adapter bit and the domain bit are set to 1. The default for both masks is 1 for all bits. Hence, the default value for both masks in sysfs is `0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff` and zcrypt is the default device driver for all AP queues. To free an AP queue for alternative device drivers, the corresponding adapter bit, or the corresponding domain bit, or both must be set to 0.

You can use kernel parameters (see “Kernel parameters” on page 500) to set the mask. On a running Linux instance, you can write to the respective sysfs attribute to change the masks.

#### Procedure

Use the following methods to change a mask on a running Linux instance.

- Write a new mask value to the sysfs attribute.

  You can write a 1 - 64-digit hexadecimal number to the respective sysfs attribute to replace the mask. If fewer than 64 digits are specified, the number is padded with 0s on the right.

  **Examples:**

  - To set the bit for the adapters with ID 0x00 and 0x01 to 0 and all other bits to 1, issue the following command:
- To set the bit for the domains with ID 0x0000 and 0x0001 to 1 and all other bits to 0, issue the following command:

  # echo 0xc > /sys/bus/ap/aqmask

• Set an individual bit value.

You can set an individual bit by specifying the bit-number, counting from the left, with one of the following prefixes:

- The minus sign (-) sets the bit to 0.

+ The plus sign (+) sets the bit to 1.

**Examples:**

- To set the bit for the adapter with ID 0x01 to 0, issue the following command:

  # echo -1 > /sys/bus/ap/apmask

- To set the bit for the domain with ID 0x0000 to 1, issue the following command:

  # echo +0 > /sys/bus/ap/aqmask

**Note:** Do not omit the prefix. Plain numbers that can be interpreted as hexadecimal values replace the entire mask, even if they are specified without the 0x prefix.

• Set a range of bit values.

Using the same prefixes as for individual bits, you can specify a range of bit-numbers to set all bits in the range to the same value. Specify a range by specifying the bit numbers of the first and last bit in the range, separated by a hyphen (-).

**Examples:**

- To set the bit for the adapters with IDs 0x00, 0x01, 0x02, and 0x03 to 0, issue the following command:

  # echo -0-3 > /sys/bus/ap/apmask

- To set the bit for the domains with ID 0x0008, 0x0009, 0x000a, 0x000b, and 0x000c to 1, issue the following command:

  # echo +8-12 > /sys/bus/ap/aqmask

- To set all domains to 0 issue:

  # echo -0-255 > /sys/bus/ap/aqmask

This command is equivalent to replacing the entire mask with 0s through the following command:

  # echo 0x0 > /sys/bus/ap/aqmask

• Set multiple bits and bit ranges with a single command.

You can set multiple bits and bit ranges with the same command, by using a comma-separated list of specifications for individual bits and ranges. The expressions are processed individually from left to right.

**Examples:**
– To set the bit for the adapters with IDs 0x00 to 0, the range from 0x01 to 0x03 to 1, and 0x0a to 1 issue:

```bash
# echo -0,+1-3,+10 > /sys/bus/ap/apmask
```

– This command is equal to the following sequence of commands:

```bash
# echo -0 > /sys/bus/ap/apmask
# echo +1-3 > /sys/bus/ap/apmask
# echo +10 > /sys/bus/ap/apmask
```

– To first set the bits for all domains to zero, then set the range 0x0010 to 0x001f to 0x001f to 1, and then set 0x0015 to 0, issue the following command:

```bash
# echo -0-255,+16-31,-20 > /sys/bus/ap/aqmask
```

This command is equivalent to the following sequence of commands:

```bash
# echo -0-255 > /sys/bus/ap/aqmask
# echo +16-31 > /sys/bus/ap/aqmask
# echo -20 > /sys/bus/ap/aqmask
```

The resulting bit mask can be read from the sysfs attribute:

```bash
# cat /sys/bus/ap/aqmask
0xbfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

### Results

You can check your results by reading the masks from the sysfs attributes. In the following example, adapter 0x01, across all domains, and domains 0x0001 and 0x0002, across all adapters, are ignored by the zcrypt device driver and thus, free for alternative device drivers.

```bash
# cat /sys/bus/ap/apmask
0xbfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
# cat /sys/bus/ap/aqmask
0x9fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

### 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 678.

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 65: AP bus attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ap_domain</code></td>
<td>Read-write attribute that represents the default domain selected by the kernel. Alternatively, you can select the default domain using a kernel parameter. See “Kernel parameters” on page 500.</td>
</tr>
<tr>
<td><code>ap_max_domain_id</code></td>
<td>Read-only attribute that represents the largest possible domain ID. Domain IDs can range from 0 to this number, which depends on the mainframe model.</td>
</tr>
</tbody>
</table>
Table 65: AP bus attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ap_control_domain_mask</td>
<td>Read-only attribute that represents the installed control domain facilities as a 32-byte field in hexadecimal notation. A maximum number of 256 domains can be addressed. Each bit position represents a dedicated control domain.</td>
</tr>
<tr>
<td>ap_usage_domain_mask</td>
<td>Read-only attribute that represents the installed usage domain facilities as a 32-byte field in hexadecimal notation. A maximum number of 256 domains can be addressed. Each bit position represents a usage domain.</td>
</tr>
<tr>
<td>ap_interrupts</td>
<td>Read-only attribute that indicates whether interrupt handling for the AP bus is enabled.</td>
</tr>
<tr>
<td>apmask</td>
<td>Read-write attribute that can represent up to 256 cryptographic adapters as a 256-bit value in hexadecimal notation. In combination with the aqmask attribute, it marks a set of AP queues that are reserved for device drivers other than zcrypt. See “Marking AP queues for alternative device drivers” on page 508.</td>
</tr>
<tr>
<td>aqmask</td>
<td>Read-write attribute that can represent up to 256 cryptographic domains as a 256-bit value in hexadecimal notation. In combination with the apmask attribute, it marks a set of AP queues that are reserved for device drivers other than zcrypt. See “Marking AP queues for alternative device drivers” on page 508.</td>
</tr>
<tr>
<td>config_time</td>
<td>Read-write attribute that represents a time interval in seconds used to detect new crypto devices.</td>
</tr>
<tr>
<td>poll_thread</td>
<td>Read-write attribute that indicates whether polling for the AP bus is enabled.</td>
</tr>
<tr>
<td>poll_timeout</td>
<td>Read-write attribute that represents the time interval of the poll thread in nanoseconds.</td>
</tr>
</tbody>
</table>

Example

```
# lszcrypt -b
ap_domain=0x6
ap_max_domain_id=0x54
ap_interrupts are enabled
config_time=30 (seconds)
poll_thread is disabled
poll_timeout=250000 (nanoseconds)
```

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/CCA/include/csulincl.h
- The EP11 library /usr/include/ep11-host-devel/ep11.h and /usr/include/ep11-host-devel/ep11adm.h
ep11.h and ep11adm.h are included in the EP11 devel package. pkcs11.h is included in the opencryptoki devel package. ica_api.h 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 IBM Z hardware. You can configure both openCryptoki (by using the icatoken) and openssl (by using the ibmca engine) to use IBM Z clear-key cryptographic hardware support through libica. See *libica Programmer’s Reference*, SC34-2602 for details about the libica functions.

If you must circumvent libica and access the 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/zcrypt.h
```

**Secure key cryptographic functions**

To use secure key cryptographic functions in your user space program by accessing a CCA coprocessor adapter, see *Secure Key Solution with the Common Cryptographic Architecture Application Programmer’s Guide*, SC33-8294. You can obtain this publication at [www.ibm.com/support/knowledgecenter/linuxonibm/ liaaf/lnz_r_cca.html](http://www.ibm.com/support/knowledgecenter/linuxonibm/ liaaf/lnz_r_cca.html).

To use secure key cryptographic functions in your user space program by accessing an EP11 coprocessor adapter, see *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713. You can obtain it at [www.ibm.com/developerworks/linux/linux390/documentation_dev.html](http://www.ibm.com/developerworks/linux/linux390/documentation_dev.html).

**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 50. Pseudorandom number generator device driver

The pseudorandom number generator (PRNG) device driver provides user-space applications with pseudorandom numbers generated by the IBM Z CP Assist for Cryptographic Function (CPACF).

The PRNG device driver supports the Deterministic Random Bit Generator (DRBG) requirements that are defined in NIST Special Publication 800-90/90A. The device driver uses the SHA-512 based DRBG mechanism.

To use the SHA-512 based DRBG, the device driver requires version 5 of the Message Security Assist (MSA), which is available as of the zEC12 and zBC12 with the latest firmware level. During initialization of the prng kernel module, or, if prng is compiled into the kernel, during kernel startup, the device drivers checks for the prerequisite.

If the prerequisites for the SHA-512 based DRBG are not fulfilled, the device driver uses the Triple Data Encryption Standard (TDES) algorithm instead. In TDES mode, the PRNG device driver uses a DRBG in compliance with ANSI X9.17 based on the TDES cipher algorithm. You can force the fallback to TDES mode by using the prng.mode=1 kernel parameter or mode=1 module parameter.

Terminology hint: Various abbreviations are commonly used for Triple Data Encryption Standard, for example: TDES, triple DES, 3DES, and TDEA.

Building a kernel with the PRNG device driver

Select option CONFIG_S390_PRNG in the kernel configuration menu to build a kernel with the PRNG 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.

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

Setting up the PRNG device driver

If the device driver is compiled as a module, you must load the device driver module, create a device node and, optionally, make it available to non-root users.

Kernel parameters

Use kernel parameters to configure the PRNG device driver if it was compiled into the kernel.
where:

\texttt{prng.mode=}

specifies the mode in which the device driver runs:

0

Default. In this mode, the device driver automatically detects the MSA extension level and feature enablement. The device driver runs in SHA512 mode if the requirements are fulfilled, otherwise it falls back to TDES mode.

1

forces the device driver to run in TDES mode. The device driver starts only if the requirements for TDES mode are fulfilled.

2

forces the device driver to run in SHA512 mode. The device driver starts only if the requirements for SHA512 mode are fulfilled. The device driver does not fall back to TDES mode.

\texttt{<sizeparam>}

adjusts the random-buffer block size that the device driver uses to generate new random bytes. In TDES mode, this value can be in the range 8 - 65536, for SHA512 mode, the range is 64 - 65536. The default is 256 bytes.

\texttt{<reseedparam>}

adjusts the reseed limit in SHA512 mode. Multiply this value with the chunksize to obtain the reseed boundary in bytes. The value can be in the range 10000 - 100000. The default is 100000. In TDES mode, the reseed limit is a constant value of 4096 bytes.

The defaults were chosen for good results with most workloads. Changing these settings might degrade cryptographic performance.

**Module parameters**

You can load and configure the PRNG device driver if it was compiled as a separate module.
The variables have the same meaning as the corresponding kernel parameters with prefix "prng." in “Kernel parameters” on page 513.

Device node

User-space programs access the PRNG 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 PRNG device driver registers 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 PRNG device.

Making the device node accessible to non-root users

By default, only user root can read from the PRNG 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.

Working with the PRNG device driver

Read random numbers and control the settings of the PRNG device driver.

Tasks include:

- “Reading pseudorandom numbers” on page 516
- “Displaying PRNG information” on page 516
- “Reseeding the PRNG” on page 517
- “Setting the reseed limit” on page 517
Reading pseudorandom numbers

The PRNG device is read-only. Use the read function, cat program, or dd program to obtain random numbers.

Example

In this example bs specifies the block size in bytes for transfer, and count specifies the number of records with block size. The bytes are written to the output file.

```
    dd if=/dev/prandom of=<output file name> bs=<xxxx> count=<nnnn>
```

Displaying PRNG information

Read the attributes of the prandom device in sysfs.

About this task

The sysfs representation of a PRNG device is a directory: /sys/devices/virtual/misc/prandom. This sysfs directory contains a number of attributes with information about the device.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>chunksize</td>
<td>The size, in bytes, of the random-data bytes buffer that is used to generate new random numbers. The value can be in the range 64 bytes - 64 KB. The default is 256 bytes. It is rounded up to the next 64-byte boundary and can be adjusted as a module parameter when you start the module.</td>
</tr>
<tr>
<td>byte_counter</td>
<td>The number of random bytes generated since the PRNG device driver was started. You can reset this value only by removing and reloading the kernel module, or rebooting Linux (if PRNG was compiled into the kernel). This attribute is read-only.</td>
</tr>
<tr>
<td>errorflag</td>
<td>SHA512 mode only: 0 if the PRNG device driver is instantiated and running well. Any other value indicates a problem. If there is an error indication other than 0:</td>
</tr>
<tr>
<td></td>
<td>• The DRBG does not provide random data bytes to user space</td>
</tr>
<tr>
<td></td>
<td>• The read() function fails</td>
</tr>
<tr>
<td></td>
<td>• The error code errno is set to EPIPE (broken pipe)</td>
</tr>
<tr>
<td>mode</td>
<td>SHA512 if the PRNG device driver runs in SHA512 mode, TDES if the PRNG device driver runs in TDES mode. This attribute is read-only.</td>
</tr>
<tr>
<td>reseed</td>
<td>SHA512 mode only: An integer, writable only by root. Write any integer to this attribute to trigger an immediate reseed of the PRNG. See “Reseeding the PRNG” on page 517.</td>
</tr>
<tr>
<td>reseed_limit</td>
<td>SHA512 mode only: An integer, writable only by root to query or set the reseed counter limit. Valid values are in the range 10000 - 100000. The default is 100000. See “Setting the reseed limit” on page 517.</td>
</tr>
<tr>
<td>strength</td>
<td>SHA512 mode only: A read-only integer that shows the security strength according to NIST SP800-57. Returns the integer value of 256 in SHA512 mode.</td>
</tr>
</tbody>
</table>

Procedure

Issue a command of this form to read an attribute:

```
    # cat /sys/devices/virtual/misc/prandom/<attribute>
```
where `<attribute>` is one of the attributes of Table 66 on page 516.

**Example**

This example shows a `prandom` device that is running in SHA512 mode, set to reseed after 2.56 MB:

```bash
# cat /sys/devices/virtual/misc/prandom/chunksize
256
# cat /sys/devices/virtual/misc/prandom/mode
2
# cat /sys/devices/virtual/misc/prandom/reseed_limit
10000
```

**Setting the reseed limit**

The PRNG reseeds after `chunksize` × `reseed_limit` bytes are read. By default, 100000 × 256 = 25.6 MB can be read before an automatic reseed occurs.

**Procedure**

To set the number of times a `chunksize` amount of random data can be read from the PRNG before reseeding, write the number to the `reseed_limit` attribute.

For example:

```bash
# echo 10000 > /sys/devices/virtual/misc/prandom/reseed_limit
```

The `reseed_limit` value must be in the range 10000 - 100000.

**Reseeding the PRNG**

You can force a reseed by writing to the `reseed` attribute.

**Procedure**

To reseed the PRNG, write an integer to its `reseed` attribute:

```bash
# echo 1 > /sys/devices/virtual/misc/prandom/reseed
```

Writing any integer value to this attribute triggers an immediate reseed of the PRNG instance.
Linux on Z and LinuxONE: Device Drivers, Features, and Commands Development stream (Kernel 4.19)
Chapter 51. True random-number generator device driver

The true random number generator (TRNG) device driver provides user-space applications with random data generated from the IBM Z hardware CPACF true random source.

Building a kernel with the true random-number generator

Control the build options for the true random-number generator 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 171: CPACF kernel configuration menu options for cryptographic operations](image)

### CONFIG_HW_RANDOM_S390

This driver provides kernel-side support for the True Random Number Generator available as CPACF extension on modern s390 hardware platforms. To compile this driver as a module, choose M here: the module will be called s390_trng.

Setting up the TRNG device driver

The true random-number generator can be built into the kernel or compiled as a separate module.

The true random-number generator requires Message-Security-Assist Extension 7 (MSA 7), which is available as of the IBM z14. During initialization of the TRNG kernel module, or, if TRNG is compiled into the kernel, during kernel startup, the device driver checks for the prerequisite. If the prerequisite is not fulfilled, the device driver silently exits.

If the TRNG device driver is compiled as a module, the module registers itself to the CPU feature MSA. The device driver is then loaded automatically. However, you can activate the device driver manually with the command `modprobe s390-trng`. If the TRNG device driver is built into the kernel, no activation is necessary.

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

**Device nodes for random data**

The true random-number generator device driver provides two interfaces to user space applications: the device node `/dev/trng` for direct access, and the generic device node `/dev/hwrng`.

The `/dev/hwrng` node appears when the TRNG or another source of random data registers with the hwrng device driver. If both the TRNG and a CCA coprocessor are registered, the TRNG takes precedence.
As of the z14, the kernel random device driver also uses the CPACF TRNG true random source through the arch_get_random_seed_* functions. The kernel random device provides two device nodes, /dev/random and /dev/urandom. The arch_get_random_seed_* functions require the CPACF TRNG. The arch_get_random_seed_* functions contribute TRNG entropy data to the kernel random device driver only if the prerequisite is fulfilled. Otherwise, they return false.

### Working with the TRNG device driver

Read random numbers and retrieve the counters of the TRNG device driver.

Tasks include:

- “Reading random numbers” on page 520
- “Displaying TRNG information” on page 520

### Reading random numbers

The TRNG device is read-only. Use the read function, cat program, or dd program to obtain random numbers.

**Example**

In this example `bs` specifies the block size in bytes for transfer, and `count` specifies the number of records with block size. The bytes are written to the output file.

```
dd if=/dev/trng of=<output file name> bs=<xxxx> count=<nnnn>
```

### Displaying TRNG information

Read the byte_counter attribute of the /misc/trng device in sysfs.

**About this task**

The sysfs representation of a TRNG device is a directory: /sys/devices/virtual/misc/trng. This sysfs directory contains an attribute, byte_counter, with statistical data.

**Procedure**

Issue this command to read the byte_counter attribute:

```
# cat /sys/devices/virtual/misc/trng/byte_counter
```

**Example**

To see statistics of a TRNG device, issue:

```
# cat /sys/devices/virtual/misc/trng/byte_counter
trng: 6187
hwrng: 528
arch: 1319696
total: 1326411
```

Where:

- **trng**
  
  shows the number of bytes delivered through the /dev/trng device node.

- **hwrng**
  
  shows the bytes retrieved from the generic hw_rng device driver and contributed to /dev/hwrng.
arch
  shows the amount of data that is supplied by the arch random implementation and delivered to the
  random device driver device nodes /dev/random and /dev/urandom.

total
  shows the sum of all bytes.
Chapter 52. Protected key device driver

The protected key device driver provides functions for generating and verifying protected keys. Protected keys are encrypted with wrapping keys that, for Linux in LPAR mode, are specific to the LPAR. For guests of z/VM or KVM, the wrapping key is specific to the guest. Both the wrapping keys and the clear key values of protected keys are invisible to the operating system. Protected keys are designed for accelerated encryption and decryption with CPACF. For more information, see the chapter about protected keys in z/Architecture Principles of Operation, SA22-7832.

Functions

The device driver provides the following functions to cryptographic applications:

- Generate a random secure key, then generate a protected key from the secure key.
  
  The secure key must be available to create a new version of the protected key whenever the current protected key is invalidated.

- Generate a secure key from a clear key, then generate a protected key from the secure key.
  
  The clear key must be in memory when the protected key is generated. Thereafter, the clear key can be deleted.

- Generate a protected key from a clear key. The clear key must be in memory when the protected key is generated.
  
  The clear key must also be available to create a new protected key if the existing protected key is invalidated.

The device driver also provides an in-kernel interface to generate protected keys. This interface is used, for example, by the paes_s390 module.

Prerequisites

The protected key device driver requires the message-security-assist-extension 3 facility (MSA level 3), which was introduced with z196.

The protected key device driver requires permission for the AES key import functions. To grant this permission, go to the security settings within the profile of the applicable LPAR on the HMC or Support Element (SE). In the CPACF Key Management Operations section, select the Permit AES Key import functions option. For z/VM and KVM guests, the LPAR in which the hypervisor runs requires this option.

Secure keys are encoded with a master key that is held in Crypto Express adapters. Functions that involve secure keys require an IBM Crypto Express adapter in CCA coprocessor mode with a valid master key. For Linux on z/VM, the adapter must be dedicated to the z/VM guest virtual machine.

Building a kernel with the protected key device driver

Select the CONFIG_PKEY kernel configuration option to build a kernel with the protected key 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 can compile the protected key device driver into the kernel or as a separate module, pkey.

**Loading the device driver module**

If the protected key device driver is compiled as a separate module, you must load it before you can generate protected keys.

Load the `pkey` module with the `modprobe` command. The `pkey` module has no module parameters.

```
# modprobe pkey
```

**External programming interfaces**

Applications can use the protected key device driver through ioctls or corresponding kernel APIs.

**Programmers:** This information is intended for programmers of cryptographic applications who want to use protected keys for accelerated cryptographic operations with CPACF.

Issue ioctls on the misc character device `/dev/pkey` to generate and handle protected keys. The ioctl interface, including the required defines and structure definitions, is described in `arch/s390/include/uapi/asm/pkey.h`. Each ioctl has a matching kernel API that is also described in this file.

<table>
<thead>
<tr>
<th>Name</th>
<th>Structure passed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKEY_GENSECK</td>
<td>struct pkey_genseck</td>
<td>Obtain a random secure key from an AP queue. The secure key is encrypted with the master key of the AP queue.</td>
</tr>
<tr>
<td>PKEY_CLR2SECK</td>
<td>struct pkey克拉2secK</td>
<td>Obtain a secure key from an AP queue. The secure key is generated from a specified clear key and encrypted with the master key of the AP queue.</td>
</tr>
<tr>
<td>PKEY_SEC2PROTK</td>
<td>struct pkey克拉2protk</td>
<td>Obtain a protected key from an AP queue. The protected key is generated from a specified secure key.</td>
</tr>
<tr>
<td>PKEY_CLR2PROTK</td>
<td>struct pkey克拉2protk</td>
<td>Obtain a protected key. The protected key is generated from a specified clear key.</td>
</tr>
<tr>
<td>PKEY_FINDCARD</td>
<td>struct pkey克拉findcard</td>
<td>Find an AP queue that holds the applicable master key for a specified secure key.</td>
</tr>
<tr>
<td>Name</td>
<td>Structure passed</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PKEY_SKEY2PKEY</td>
<td>struct pkey_skey2pkey</td>
<td>Find an AP queue that holds the applicable master key for a specified secure key. Then use that AP queue to obtain a protected key that is generated from the secure key. This ioctl call combines PKEY_FINDCARD and PKEY_SEC2PROTK.</td>
</tr>
</tbody>
</table>
Chapter 53. Hardware-accelerated in-kernel cryptography

The Linux kernel implements cryptographic operations for kernel subsystems like dm-crypt and IPSec. Applications can use these operations through the kernel cryptographic API.

In-kernel cryptographic and checksum operations can be performed by platform-specific implementations instead of the generic implementations within the Linux kernel. On IBM Z, hardware-accelerated processing is available for some of these operations.

Hardware dependencies and restrictions

The cryptographic operations that can be accelerated by hardware implementations depend on your IBM Z hardware and mode of operating the kernel.

The following table provides an overview of the supported cryptographic standards and the earliest mainframe with hardware-acceleration for the associated operations:

<table>
<thead>
<tr>
<th>Algorithms for</th>
<th>Hardware-acceleration available as of</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>z990</td>
</tr>
<tr>
<td>SHA-256</td>
<td>z9®</td>
</tr>
<tr>
<td>SHA-512</td>
<td>z10</td>
</tr>
<tr>
<td>DES and TDES</td>
<td>z990 for the ECB and CBC modes</td>
</tr>
<tr>
<td></td>
<td>z196 for the CTR mode</td>
</tr>
<tr>
<td>AES</td>
<td>z9 for the ECB and CBC modes with 128-bit keys</td>
</tr>
<tr>
<td></td>
<td>z10 for the ECB and CBC modes for all AES key sizes</td>
</tr>
<tr>
<td></td>
<td>z196 for the CTR mode for all AES key sizes</td>
</tr>
<tr>
<td></td>
<td>z196 for XTS mode for 256-bit and 512-bit keys</td>
</tr>
<tr>
<td></td>
<td>z14 for GCM for 128-bit, 192-bit, and 256-bit keys</td>
</tr>
<tr>
<td>GHASH</td>
<td>z196</td>
</tr>
<tr>
<td>PAES</td>
<td>z196 (AES with protected key)</td>
</tr>
<tr>
<td>CRC32</td>
<td>z13 for CRC-32 (IEEE 802.3 Ethernet) and CRC-32C (Castagnoli)</td>
</tr>
</tbody>
</table>

CPACF dependencies

Hardware-acceleration for DES, TDES, AES, GHASH, and PAES requires the Central Processor Assist for Cryptographic Function (CPACF). Read the features line from /proc/cpuinfo to find out whether the CPACF feature is enabled on your hardware.

Example:

```
# cat /proc/cpuinfo | grep features
features           : esan3 zarch stfle msa ldisp einm dfp edat etf3eh highgps te vx sie
```

In the output line, msa indicates that the CPACF feature is enabled. For information about enabling CPACF, see the documentation for your IBM Z hardware.
Vector Extension Facility dependencies

Hardware-acceleration for CRC32 algorithms requires the Vector Extension Facility. Read the features line from /proc/cpuinfo to find out whether this facility is available on your hardware.

**Example:**

```
# grep features /proc/cpuinfo
features        : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te vx sie
```

In the output line, vx indicates that the Vector Extension Facility is available.

FIPS restrictions of the hardware capabilities

If the kernel runs in Federal Information Processing Standard (FIPS) mode, only FIPS 140-2 approved algorithms are available. DES, for example, is not approved by FIPS 140-2.

Read /proc/sys/crypto/fips_enabled to find out whether your kernel runs in FIPS mode.

**Example:**

```
# cat /proc/sys/crypto/fips_enabled
1
```

The kernel of the example runs in FIPS mode. If /proc/sys/crypto/fips_enabled does not exist or the value is other than 1, the kernel does not run in FIPS mode.

You control the FIPS mode with the `fips` kernel parameter, see “Start Linux in FIPS mode” on page 533.

For more information about FIPS, see csrg.nist.gov/publications/detail/fips/140/2/final.

Building a kernel with hardware-accelerated cryptographic operations

Control the build options for hardware-accelerated in-kernel cryptographic operations 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 173: CPACF kernel configuration menu options for cryptographic operations

**CONFIG_PKEY**
This option provides an API for creation and handling of protected keys. See Chapter 52, “Protected key device driver,” on page 523.

**CONFIG_CRYPTO_PAES_S390**
This option provides support for hardware-acceleration of the AES cipher algorithms for use with protected key. The support can be compiled into the kernel or as a separate module, paes_s390.

**CONFIG_CRYPTO_SHA1_S390**
This option provides support for hardware-acceleration of the SHA-1 secure hash standard (FIPS 180-1/DFIPS 180-2). The support can be compiled into the kernel or as a separate module, sha1_s390.

**CONFIG_CRYPTO_SHA256_S390**
This option provides support for hardware-acceleration acceleration of the SHA256 secure hash standard (DFIPS 180-2). The support can be compiled into the kernel or as a separate module, sha_256.

**CONFIG_CRYPTO_SHA512_S390**
This option provides support for hardware-acceleration acceleration of the SHA512 secure hash standard. The support can be compiled into the kernel or as a separate module, sha_512.

**CONFIG_CRYPTO_DES_S390**
This option provides support for hardware-acceleration, through CPACF, of the DES cipher algorithm (FIPS 46-2), and Triple DES EDE (FIPS 46-3). The support can be compiled into the kernel or as a separate module, des_s390.

**CONFIG_CRYPTO_AES_S390**
This option provides support for hardware-acceleration, through CPACF, of the AES cipher algorithms (FIPS-197). The support can be compiled into the kernel or as a separate module, aes_s390.

**CONFIG_CRYPTO_GHASH_S390**
This option provides support for hardware-acceleration, through CPACF, of the GHASH message digest algorithm for GCM (Galois/Counter Mode). The support can be compiled into the kernel or as a separate module, ghash_s390.
This option provides support for hardware-accelerated implementations of CRC algorithms. With this option, you can optimize the computation of CRC-32 (IEEE 802.3 Ethernet) and CRC-32C (Castagnoli). The support can be compiled into the kernel or as a separate module, crc32-vx_s390.

PAES does not have a separate kernel configuration option, but it depends on the protected key device driver (“Building a kernel with the protected key device driver” on page 523).

For hardware requirements, see Table 68 on page 527.

Support modules

A udev rule ensures that any separately compiled modules for which the hardware prerequisites are fulfilled are loaded automatically for you.

The following support modules for hardware-accelerated cryptographic operations can be compiled separately from the kernel. None of these modules have module parameters.

- **sha1_s390**: enables hardware-acceleration for SHA-1 operations. sha1_s390 requires the sha_common module.

- **sha_256**: enables hardware-acceleration for SHA-224 and SHA-256 operations. sha_256 requires the sha_common module.

- **sha_512**: enables hardware-acceleration for SHA-384 and SHA-512 operations. sha_512 requires the sha_common module.

- **ghash_s390**: enables hardware-acceleration for Galois hashes.

- **aes_s390**: enables hardware-acceleration for AES encryption and decryption for the following modes of operation:
  - ECB, CBC, and CTR for key lengths 128, 192, and 256 bits
  - XTS for key lengths 128 and 256 bits
  - GCM for key lengths 128, 192, and 256 bits

- **des_s390**: enables hardware-acceleration for DES and TDES for the following modes of operation: ECB, CBC, and CTR.

- **crc32-vx_s390**: enables hardware-acceleration for CRC-32 (IEEE 802.3 Ethernet) and CRC-32C (Castagnoli).

- **paes_s390**: enables protected key AES encryption and decryption for the following modes of operation:
  - ECB, CBC, and CTR for key lengths 128, 192, and 256 bits
  - XTS for key lengths 128 and 256 bits

The paes_s390 module requires the pkey device driver, see Chapter 52, “Protected key device driver,” on page 523. The module also requires a cryptographic adapter in CCA coprocessor mode for the creation and handling of protected keys.

The cipher in the paes_s390 module uses 64-byte CCA secure keys, for example, keys that are generated by the pkey device driver. XTS requires two secure keys.

Before the paes_s390 module uses secure keys in a cypher, it transforms them into protected keys. If a protected key becomes invalid, the paes_s390 module generates a new protected key from the master key.
Mainframe hardware prior to z14: To use CPACF for AES-GCM operations, you must load both the aes_s390 and ghash_s390 module.

**Tip:** Load the modules with `modprobe`. `modprobe` handles dependencies on other modules for you.

**Example:**

```
# modprobe sha_512
```

---

**Confirming hardware support for cryptographic operations**

Read `/proc/crypto` to confirm that cryptographic operations are performed with hardware support.

**Procedure**

Read the `driver` lines from the content of `/proc/crypto`.

**Example:**

```
# cat /proc/crypto | grep driver
driver       : sha512-s390
driver       : sha224-s390
driver       : sha256-s390
driver       : sha1-s390
driver       : ghash-s390
...
driver       : crc32c-vx
driver       : crc32be-vx
driver       : crc32-vx
```

Lines that end in `-s390` indicate hardware-acceleration for a corresponding cryptographic algorithm or mode. Lines that end in `-vx` indicate hardware-acceleration for CRC32 checksums.
Chapter 54. FIPS mode

In Federal Information Processing Standard (FIPS) mode, the kernel enforces FIPS security standards, including FIPS 140-2.

FIPS 140-2 includes, but is not limited to, the following requirements:

- Only FIPS 140-2 approved encryption algorithms can be used (see “FIPS restrictions of the hardware capabilities” on page 528).
- All kernel modules must be signed. The lengths of RSA keys used for signing are limited to 2048 or 3072 bits. For details about signing kernel modules, see Documentation/module-signing.txt in the kernel source tree.

For more information about FIPS 140-2, go to csrc.nist.gov/publications/detail/fips/140/2/final.

**Note:** Enabling FIPS mode is not sufficient to make your kernel certified according to FIPS 140-2.

FIPS 140-2 certification is specific to a particular hardware platform and kernel build. Typically, running in FIPS mode is required, but not sufficient to be FIPS 140-2 certified. Check with your distributor to find out whether your kernel is certified according to FIPS 140-2.

**Building a kernel with FIPS mode 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.

Select the common code kernel configuration option CONFIG_CRYPTO_FIPS to build a kernel that supports the FIPS mode.

```
Cryptographic API --->
*** Crypto core or helper ***
FIPS 200 compliance                          (CONFIG_CRYPTO_FIPS)
```

*Figure 174: Kernel configuration menu option for FIPS subchannel device driver*

CONFIG_CRYPTO_FIPS has complex dependencies and might be hidden in the menu. See the help for the configuration menu option for details.

**Tip:** In the menu, press the `Z` key to display hidden options.

**Start Linux in FIPS mode**

Use the `fips=` kernel parameter to start the Linux kernel in FIPS mode.

For more information about FIPS 140-2, go to csrc.nist.gov/publications/detail/fips/140/2/final.
1 enables the FIPS mode. 0, the default, disables the FIPS mode.

Example

```
fips=1
```
Chapter 55. Instruction execution protection

The instruction execution protection feature on IBM mainframes protects against data execution, similar to the NX feature on other architectures.

Instruction execution protection prevents stack-overflow exploits and generally makes a system insensitive to buffer-overflow attacks.

Data instruction protection is available on IBM mainframe hardware with the IEP feature. For Linux as a guest of a hypervisor, the hypervisor must support and use the instruction execution protection feature. For example, Linux as a guest of KVM must run on a KVM host with kernel 4.11 or later, and with the IEP feature enabled.

Setting up instruction execution protection

By default, Linux on Z uses the instruction execution protection feature if it is available. You can use the noexec kernel parameter to disable the feature in Linux.

Example

- Set and query the executable stack flag.

  ```
  # execstack -s /usr/bin/find
  # execstack -q /usr/bin/find
  X /usr/bin/find
  ```

  The leading X at the beginning of the query output line indicates that the stack is executable.

  **Hint:** You can also use the readelf command to confirm that the stack can be executed.

  ```
  # readelf -a /usr/bin/find | grep GNU_STACK -A 1
  GNU_STACK 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 RWE B
  ```

  The RWE towards the end of the output line means read/write/execute. You can obtain the readelf command as part of the binutils package. For command details, see the man page.
• Clear and query the executable stack flag.

```bash
# execstack -c /usr/bin/find
# execstack -q /usr/bin/find
- /usr/bin/find
```

The leading - at the beginning of the query output line indicates that the stack is not executable.

**Hint:** You can also use the `readelf` command to confirm that the stack cannot be executed.

```bash
# readelf -a /usr/bin/find | grep GNU_STACK -A 1
GNU_STACK 0x0000000000000000 0x0000000000000000 0x0000000000000000
0x0000000000000000 0x0000000000000000 RW 8
```

The RW towards the end of the output line means read/write, but not execute.
Part 8. Performance measurement using hardware facilities

The Z hardware provides performance data that can be accessed by Linux on 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 435.

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 161 for DASD performance and “Starting and stopping collection of QETH performance statistics” on page 277 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
Chapter 56. Channel measurement facility

The IBM Z architecture provides a channel measurement facility to collect statistical data about I/O on the channel subsystem.

Data collection can be enabled for all CCW devices. User space applications can access this data through the sysfs.

The channel measurement facility provides the following features:

- Basic channel measurement format for concurrently collecting data on up to 4096 devices. (Specifying 4096 or more channels causes high memory consumption, and enabling data collection might not succeed.)
- Extended channel measurement format for concurrently collecting data on an unlimited number of devices.
- Data collection for all channel-attached devices, except those using QDIO (that is, except qeth and SCSI-over-Fibre channel attached devices)

**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.

![Diagram](image.png)

*Figure 177: Channel measurement facility kernel parameters*

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” on page 540).

- To switch off data collection issue a command of this form:

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

  When data collection for a device is switched off, the subdirectory /sys/bus/ccw/devices/<device_bus_id>/cmf and its content are deleted.

Example

In this example, data collection for a device /sys/bus/ccw/devices/0.0.b100 is already active and reset:

````
# cat /sys/bus/ccw/devices/0.0.b100/cmb_enable
1
# echo 1 > /sys/bus/ccw/devices/0.0.b100/cmb_enable
```

Reading data

Read the sysfs attributes with collected I/O data, for example with the cat command.

Procedure

To read one of the attributes issue a command of this form:

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

where /sys/bus/ccw/devices/<device_bus_id> is the directory that represents the device, and <attribute> the attribute to be read. Table 69 on page 540 summarizes the available attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssch_rsch_count</td>
<td>An integer that represents the ssch rsch count value.</td>
</tr>
</tbody>
</table>
### Table 69: Attributes with collected I/O data (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample_count</td>
<td>An integer that represents the sample count value.</td>
</tr>
<tr>
<td>avg_device_connect_time</td>
<td>An integer that represents the average device connect time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_function_pending_time</td>
<td>An integer that represents the average function pending time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_device_disconnect_time</td>
<td>An integer that represents the average device disconnect time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_control_unit_queuing_time</td>
<td>An integer that represents the average control unit queuing time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_initial_command_response_time</td>
<td>An integer that represents the average initial command response time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_device_active_only_time</td>
<td>An integer that represents the average device active only time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_device_busy_time</td>
<td>An integer that represents the average value device busy time, in nanoseconds, per sample.</td>
</tr>
<tr>
<td>avg_utilization</td>
<td>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.</td>
</tr>
<tr>
<td>avg_sample_interval</td>
<td>An integer that represents the average time, in nanoseconds, between two samples during the measurement period. Can be &quot;-1&quot; if no measurement data has been collected.</td>
</tr>
<tr>
<td>avg_initial_command_response_time</td>
<td>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.</td>
</tr>
<tr>
<td>avg_device_busy_time</td>
<td>An integer that represents the average time in nanoseconds of the subchannel being in the &quot;device busy&quot; state when initiating a start or resume function. Available in extended format only.</td>
</tr>
</tbody>
</table>

### Example

To read the **avg_device_busy_time** attribute for a device `/sys/bus/ccw/devices/0.0.b100`:

```bash
# cat /sys/bus/ccw/devices/0.0.b100/cmf/avg_device_busy_time
21
```
Chapter 57. 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
- MT-diagnostic counter set

A further common counter set, the Coprocessor group counter set, cannot be accessed from Linux on Z.

**Sampling facility**

The sampling facility includes the following sampling modes:

- Basic-sampling mode
- Diagnostic-sampling mode

The diagnostic-sampling mode is intended for use by IBM support only.

The number and type of individual counters and the details of the sampling facility depend on your 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 656). For details, see IBM The CPU-Measurement Facility Extended Counters Definition for z10, z196/z114, zEC12/zBC12, z13/z13s, and z14, 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 178 on page 543 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 178: 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” on page 544
- “Reading CPU-measurement counters” on page 545
- “Collecting CPU-measurement sample data” on page 546
- “Setting limits for the sampling facility buffer” on page 547
- “Obtaining details about the CPU-measurement facilities” on page 548

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 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 656).

**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 656).

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, zEC12/zBC12, z13/z13s, and z14, 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:
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
```

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 `ls cpumf -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 580.

**Before you begin**

For each CPU, the CPU-measurement sampling facility has a buffer for writing sample data. The required buffer size depends on the sampling function and the sampling interval that is used by the perf tool. The sampling facility starts with an initial buffer size that depends on the expected requirements, your IBM Z hardware, and the available hardware resources. During the sampling process, the sampling facility increases the buffer size if required.

The sampling facility is designed for autonomous buffer management, and you do not usually need to intervene. You might want to change the minimum or maximum buffer size, for example, for one of the following reasons:

- There are considerable resource constraints on your system that cause perf sampling to malfunction and sample data to be lost.
- As an expert user of perf and the sampling facility, you want to explore results with particular buffer settings.

**Procedure**

Use the chcpumf command to set the minimum and maximum buffer sizes.

1. Optional: Specify the lscpumf command with the -i parameter to display the current limits for the buffer size (see “lscpumf - Display information about the CPU-measurement facilities” on page 656).

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:

```bash
# chcpumf -V -m 500 -x 1000
```

Sampling buffer sizes:
- Minimum: 500 sample-data-blocks
- Maximum: 1000 sample-data-blocks

Obtaining details about the CPU-measurement facilities

You can obtain version information for the CPU-measurement counter and sampling facility and check which counter sets are authorized on your LPAR.

Procedure

1. Issue the `lscpumf` command with the `-i` parameter to display detailed information and debug data about the CPU-measurement facilities.

   Example:

   ```bash
   # 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:

   ```bash
   # 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 53.

   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 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

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 58. 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:

```bash
# 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.

```bash
# 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 59. Obtaining QDIO performance statistics

There is a debugfs interface for QDIO performance statistics. These statistics apply to FCP devices and to qeth devices.

QDIO performance statistics are available by device. These statistics are located in

```
<debugfs_mount>/qdio/<device_bus_id>/statistics
```

where `<debugfs_mount>` is the mount point for debugfs and `<device_bus_id>` is the bus ID of an FCP or qeth device.

Write 1 to the statistics file of a device to start collecting performance data for that device. Write 0 to this file to stop collecting performance data. By default no data is collected. Data is written to a statistics file and one or more files named `input_*` and `output_*`.

You can use the `cat` command to read the data from the `input_*`, `output_*`, and `statistics` files.

**Example**

Assuming that debugfs is mounted at `/sys/kernel/debug`, the following command starts performance data collection for a device with bus ID 0.0.fc00:

```
# echo 1 > /sys/kernel/debug/qdio/0.0.fc00/statistics
```

The following command reads the collected data:

```
# cat /sys/kernel/debug/qdio/0.0.fc00/statistics
```
Chapter 60. Control program identification

For Linux in LPAR mode, you can provide data about the Linux instance to the control program identification (CPI) feature.

The data is used, for example, to represent the Linux instance on the HMC or SE.

You provide data to the CPI feature in two steps:

1. Write values for one or more of the following items to specific sysfs attributes. Use the sysfs attributes in /sys/firmware/cpi for testing purposes. Use the sysfs attributes in /etc/sysconfig/cpi to make the changes persistent across reboots:
   - The system name
   - The sysplex name (if applicable)
   - The operating system type
   - The operating system level

2. Transfer the data to the SE, see “Sending system data to the SE” on page 557.

   Linux provides a systemd service unit, cpi, to set the attributes automatically during reboot. To automatically set the attributes during reboot, issue a command as follows:

   ```bash
   # systemctl enable cpi
   ```

   **Note:** As of kernel 4.5, the sclp_cpi module is no longer available. Use the sysfs interface to provide information to the CPI feature.

Specifying a system name

Use the system_name attribute in the /etc/sysconfig/cpi directory in sysfs to specify a system name for your Linux instance.

**About this task**
The system name is a string that consists of up to eight characters of the following set: A-Z, 0-9, $, @, #, and blank.

**Example**

```bash
# echo LPAR12 > /etc/sysconfig/cpi/system_name
```

To make a change to the currently running system, use:

```bash
# echo LPAR12 > /sys/firmware/cpi/system_name
```

**What to do next**
To make the setting take effect immediately, transfer the data from /sys/firmware/cpi/system_name to the SE (see “Sending system data to the SE” on page 557).
Specifying a sysplex name

Use the `sysplex_name` attribute in the `/etc/sysconfig/cpi` directory in sysfs to specify a sysplex name.

About this task
The sysplex name is a string that consists of up to eight characters of the following set: A-Z, 0-9, $, @, #, and blank.

Example

```
# echo SYSPLEX1 > /etc/sysconfig/cpi/sysplex_name
```

To make a change to the currently running system, use:

```
# echo LPAR12 > /sys/firmware/cpi/sysplex_name
```

What to do next
To make the setting take effect, transfer the data from `/sys/firmware/cpi/system_name` to the SE (see “Sending system data to the SE” on page 557).

Specifying a system type

Linux uses the `/etc/sysconfig/cpi/system_type` sysfs attribute to identify itself as a Linux instance.

About this task
Unless your distribution sets this value for you, write LINUX to the attribute.

Example

```
# cat /etc/sysconfig/cpi/system_type
"
# echo LINUX > /etc/sysconfig/cpi/system_type
```

What to do next
To make the setting take effect, transfer the data to the SE (see “Sending system data to the SE” on page 557).

Specifying the system level

Linux uses the `/etc/sysconfig/cpi/system_level` sysfs attribute for the kernel version.

About this task
Write the kernel version to the attribute unless your distribution sets the correct value for you. The value must conform to this format:

```
0x<hh>00000000<aa><bb><cc>
```

where:
are two hexadecimal digits. A value of 80 indicates that the Linux instance acts as a hypervisor. The SE ignores this value.

<aa>
are two hexadecimal digits for the major version of the kernel.

<bb>
are two hexadecimal digits for the minor version of the kernel.

<cc>
are two hexadecimal digits for the stable version of the kernel.

Example

Linux kernel 4.19 displays as

```bash
# cat /etc/sysconfig/cpi/system_level
0x000000000041300
```

What to do next

To make a change to the currently running system, use:

```bash
# echo 0x000000000041305 > /sys/firmware/cpi/system_level
```

To make the setting take effect, transfer the data to the SE (see “Sending system data to the SE” on page 557).

Sending system data to the SE

Use the set attribute in the /etc/sysconfig/cpi directory in sysfs to send data to the Support Element (SE).

About this task

To send the data in attributes sysplex_name, system_level, system_name, and system_type to the SE, write an arbitrary string to the set attribute.

Example

```bash
# echo 1 > /sys/firmware/cpi/set
```

Attribute values in /sys/firmware/cpi are not persistent across reboots. To make the changes persistent, use the systemd service unit cpi and its corresponding configuration file /etc/sysconfig/cpi.
Chapter 61. 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 Support 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 179 on page 559).

Device Drivers --->
  ...
  Character devices --->
    ...
    --- S/390 character device drivers (depends on S390) ---
    --- Support for Call Home via Asynchronous SCLP Records (CONFIG_SCLP_ASYNC)

Figure 179: 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:

```bash
# 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:

```bash
# echo <flag> > /proc/sys/kernel/callhome
```

This command is equivalent to the following:

```bash
# 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:

  ```bash
  # echo 1 > /proc/sys/kernel/callhome
  ```

- To deactivate the Call Home support, issue:

  ```bash
  # echo 0 > /proc/sys/kernel/callhome
  ```
Chapter 62. Displaying system information

You can display information about the resources and capabilities of your Linux instance and about the hardware and hypervisor on which your Linux instance runs.

Displaying hardware and hypervisor information

You can display information about the physical and virtual hardware on which your Linux instance runs.

**Procedure**

Issue the following command:

```bash
# cat /proc/sysinfo
```

The output of the command is divided into several blocks.

- The first two blocks provide information about the mainframe hardware.
- The third block provides information about the LPAR on which the Linux instance runs, either in LPAR mode or as a guest of a hypervisor.
- Further blocks are present only if the Linux instance runs as a guest of a hypervisor. The field names in these sections have a prefix, VM<nn>, where <nn> is the hypervisor level.

If the hypervisor runs in LPAR mode, there is only one such block, with prefix VM00. If the hypervisor runs as a guest of another hypervisor, there are multiple such blocks with prefixes VM00, VM01, and so on. The highest prefix number describes the hypervisor that is closest to the Linux instance.

You can use the information from `/proc/sysinfo`, for example, to verify that a guest relocation has taken place.

The following example shows the command output for an instance of Linux on z/VM. For an example for Linux as a KVM guest, see Device Drivers, Features, and Commands for Linux as a KVM Guest, SC34-2754.

**Example:**

```bash
# cat /proc/sysinfo
Manufacturer:         IBM
...
CPUs Total:           45
...
LPAR Number:          31
...
LPAR Name:            LP000031
...
LPAR Extended Name:   Partition 31 Test System
LPAR UUID:            93724168-fda3-429b-8b28-a5d245dcb3ff
...
VM00 Name:            VM310012
VM00 Control Program: z/VM 6.3.0
VM00 Adjustment:    83
VM00 CPUs Total:       2
VM00 CPUs Configured: 2
VM00 CPUs Standby:            0
VM00 CPUs Reserved:       0
```

The fields with prefix LPAR include information that identifies and labels the partition:

**Number**

- shows a number that identifies the partition within the mainframe.
Name
  shows a partition name of up to 8 characters, as assigned on the HMC.

Extended Name
  depending on your hardware, this field can contain an extended partition name, it can be empty, or it
  might be omitted.

UUID
  shows the universally unique identifier (UUID) of the partition.
  This field is present only if a UUID is assigned to the partition. On hardware that does not support
  UUIDs for partitions, this field is always omitted.

The fields with prefix VM<nn> show the following information:

Name
  shows the name of the z/VM guest virtual machine according to the z/VM directory.

Control Program
  shows hypervisor information.

Adjustment
  does not show useful information for Linux on z/VM.

CPUs Total
  shows the number of virtual CPUs that z/VM provides to Linux.

CPUs Configured
  shows the number of virtual CPUs that are online to Linux.

CPUs Standby
  shows the number of virtual CPUs that are available to Linux but offline.

CPUs Reserved
  shows the number of extra virtual CPUs that z/VM could make available to Linux. This is the difference
  between the maximum number of CPUs in the z/VM directory entry for the guest virtual machine and
  the number of CPUs that are currently available to Linux.

Retrieving STHYI data

Store Hypervisor Information (STHYI) includes information about the IBM Z hardware, LPAR and, if
applicable, the hypervisor host system on which your Linux instance runs.

STHYI includes, but is not limited to, the following information:

• The CPU count, by type (CP or IFL)
• Limitations for shared CPUs
• CEC and LPAR identifiers

The methods that you can use to retrieve this information differ between Linux in LPAR mode and Linux as
a guest operating system of z/VM or of KVM.

<table>
<thead>
<tr>
<th>Method</th>
<th>Linux in LPAR mode</th>
<th>Linux as a z/VM or KVM guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>STHYI instruction with the GCC inline assembly</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>For an example, see arch/s390/kernel/sthyi.c in the Linux source tree.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 70: Available methods by environment (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Linux in LPAR mode</th>
<th>Linux as a z/VM or KVM guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>qclib</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>See the readme file of the qclib package for details.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s390_sthyi() system call</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>See the man page for details.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The return data for both the STHYI instruction and the s390_sthyi() system call matches the content of the STHYI response buffer as described in z/VM: CP Programming Services, SC24-6272. The qclib library provides an API for querying the information. See the readme file of the qclib package about obtaining the API description.

You can find the qclib package and more information about qclib on developerWorks at www.ibm.com/developerworks/linux/linux390/qclib.html.

Check whether the Linux instance can be a hypervisor

An instance of Linux on Z must have the SIE (Start Interpretive Execution) capability to be able to act as a hypervisor, such as a KVM host.

Procedure

1. Issue the following command to find out whether you can operate your Linux instance as a hypervisor.

   ```bash
   # cat /proc/cpuinfo
   vendor_id : IBM/S390
   # processors : 1
   bogomips per cpu: 14367.00
   features : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh
   highgprs sie
   cache0 : level=1 type=Data scope=Private size=128K
   ...
   ```

2. Examine the features line in the command output. If the list of features includes sie, the Linux instance can be a hypervisor.

The Linux instance of the example can be a hypervisor.
Chapter 63. 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 0.12
# chchp -v 1 0.12
```

**Rationale:** Linux does not always receive a notification (machine check) when the status of a path changes (especially for a path that comes online again). To make sure Linux has up-to-date information about the usable paths, path verification is triggered through the Linux vary operation.

Determining channel path usage

To determine the usage of a specific channel path on LPAR, for example, to check whether traffic is distributed evenly over all channel paths, use the channel path measurement facility.

See “Channel path measurement” on page 13 for details.

Configuring LPAR I/O devices

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 758 and “Changing the exclusion list” on page 759.

**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 00000000 00000000 00144D7A
```

Check if cio_ignore is used and verify that the console device, which is typically device number 0.0.0009, is not ignored.

### Excessive guest swapping

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 48, “Cooperative memory management,” on page 489 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:
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.

Auto-configuration overrides persistent configuration

After upgrading Linux, persistent device configurations that were specified with chzdev are overridden with the auto-configuration.

Ensure that your device is configured with the zdev:early=1 attribute. See “Activating a device early during the boot process” on page 595.

For zfcp devices that were configured with a chzdev version earlier than 2.5, you must also perform a one-time migration of udev rules. To migrate the rules, issue the following command:

```
# chzdev zfcp-lun --configured --enable --persistent --force
```

For more information about auto-configuration data, see Chapter 3, “Device auto-configuration for Linux in LPAR mode,” on page 19.

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 9, “Shutdown actions,” on page 105 for details.
Chapter 64. Kernel messages

IBM 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 IBM 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

www.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 4.19 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 IBM 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 569 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:

```bash
man <message_number>
```

**Example**
Enter the following command to display the man page for message xpram.ab9aa4:

```bash
# man xpram.ab9aa4
```

The corresponding man page looks like this example:

```text
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 parameter or with the 'xpram.parts' kernel parameter must be an integer in the range 1 to 32. The XPRAM device driver created a maximum of 32 partitions that are probably not configured as intended.
User action               If the XPRAM device driver has been compiled as a separate module, unload the module and load it again with a correct value for the 'devs' module parameter. If the XPRAM device driver has been compiled into the kernel, correct the 'xpram.parts' parameter in the kernel parameter line and restart Linux.
```

Viewing messages with the IBM Doc Buddy app

You can view documentation for IBM Z specific Linux kernel messages through IBM Doc Buddy, an app for mobile devices. IBM Doc Buddy is helpful in environments from where the message documentation on the internet is not directly accessible.

**Before you begin**
Obtain IBM Doc Buddy from Apple App Store or from Google Play. While your device is online, you can display message documentation by entering the IDs for messages of interest in the main search field of the app.
Procedure
Perform the following steps to enable IBM Doc Buddy in offline mode:
1. While your mobile device is online, start IBM Doc Buddy and tap My Account in the lower right corner of the main pane.
2. Under Messages and codes, tap Components to list the available components in alphabetical order.
3. Scroll down to "Linux on Z and LinuxONE" and download the component.
4. Return to the main pane.

Results
You can now display message documentation for Linux on Z and LinuxONE while your mobile device is offline.

Related information
ibmdocbuddy.mybluemix.net
Use these commands, kernel parameters, kernel options to configure Linux on Z. Be aware of the z/VM DIAG calls required by Linux on Z.

**Newest version**
You can find the newest version of this publication at

**Restrictions**
For prerequisites and restrictions see
Chapter 65. Commands for Linux on Z

You can use Z specific commands to configure and work with the z/Architecture device drivers and features.

Most of the commands described in this section are included in the s390-tools package (version 2.7.0) 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/. You can extract any missing files from the etc subdirectory in the s390-tools package.

Commands described elsewhere

• For the zipl command, see Chapter 6, “Initial program loader for IBM Z - zipl,” on page 59.
• For the snipl command, see Chapter 11, “Remotely controlling virtual hardware - snipl,” on page 115.
• 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:

```
<command> Other command options
  -h
  --help
  --version
```

Figure 180: Common command options

where command can be any of the commands described in this section.

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 752 to work with CCW_GROUP devices. For more information about CCW devices and CCW group devices, see “Device categories” on page 7.

The chccwdev command uses cio_settle before it changes anything to ensure that sysfs reflects the latest device status information and includes newly available devices.

chccwdev syntax

Where:

-e or --online
sets the device online.

-d or --offline
sets the device offline.

-s or --safeoffline
waits until all outstanding I/O requests complete, and then tries to set the device offline. Valid for DASDs only.

-f or --forceonline
forces a boxed device online, if this action is supported by the device driver.

-a 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**

```
chchp
```

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

```
chcpumf -V
-m <min_sdb> or --min <min_sdb>
-x <max_sdb> or --max <max_sdb>
```

where:

- **-m <min_sdb> or --min <min_sdb>**
  - specifies the minimum sampling facility buffer size in sample-data-blocks. A sample-data-block occupies approximately 4 KB. The sampling facility starts with this buffer size if it exceeds the initial buffer size that is calculated by the sampling facility.

- **-x <max_sdb> or --max <max_sdb>**
  - specifies the maximum sampling facility buffer size in sample-data-blocks. A sample-data-block occupies approximately 4 KB. While it is running, the sampling facility dynamically adjusts the buffer size to a suitable value, but cannot exceed this limit.

- **-V or --verbose**
  - displays the buffer size settings after the changes.

- **-v or --version**
  - displays the version number of chcpumf and exits.

- **-h or --help**
  - displays out a short help text, then exits. To view the man page, enter `man chcpumf`.

Example

To change the minimum buffer size to 500 times the size of a sample-data-block and the maximum buffer size to 1000 times the size of a sample-data-block, issue:

```
# chcpumf -V -m 500 -x 1000
Sampling buffer sizes:
  Minimum: 500 sample-data-blocks
  Maximum: 1000 sample-data-blocks
```
**chreipl - Modify the re-IPL configuration**

Use the chreipl tool to modify the re-IPL configuration for Linux on 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**

<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 64.

<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.

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. The -b option can be used if the device node or directory maps to a SCSI disk.
Note: You cannot load Linux kernel 4.13 or later from an NSS. The NSS could contain a Linux distribution with NSS support or another mainframe operating system, for example, CMS.

-L or --loadparm <parameter>
specifies the entry in the boot menu to be used for the next reboot. This parameter applies to DASD re-IPL devices 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 83, “Example for a DASD menu configuration (LPAR)” on page 88, and “Menu configurations” on page 76 for details.

A SCSI re-IPL device might hold a distribution that supports this parameter for SCSI devices. See the distribution-specific documentation for this distribution about valid specifications.

-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 71 on page 582.

Table 71: Maximum characters for additional kernel parameters

<table>
<thead>
<tr>
<th>Virtual hardware where Linux runs</th>
<th>DASD re-IPL device</th>
<th>SCSI re-IPL device</th>
<th>NSS re-IPL device</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/VM guest virtual machine</td>
<td>64</td>
<td>3452</td>
<td>56</td>
</tr>
<tr>
<td>LPAR</td>
<td>none</td>
<td>3452</td>
<td>n/a</td>
</tr>
</tbody>
</table>

If you omit this parameter, the existing boot parameters in the next boot configuration are used without any changes.

-f or --force
With this option, you can force the re-IPL from a target device even if the target cannot be verified by the system. This is the case, for example, if the device is on the cio_ignore exclusion list (blacklist).

Note: Use this option with great care. Specifying a non-existing device causes the re-IPL to fail.

-h or --help
displays help information for the command. To view the man page, enter man chreipl.

-v or --version
displays version information.

For disk-type re-IPL devices, the command accepts but does not require an initial statement:

ccw
declares that the following parameters refer to a DASD 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 0x50050763030c562, 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-36005076303ff562000000000010b4, 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 0x50050763030c562 0x401040b300000000

  – Using bus ID, WWPN, and LUN with the optional fcp statement:

    # chreipl fcp 0.0.1700 0x50050763030c562 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 0x50050763030c562 -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-36005076303ff562000000000010b4
– Using a directory within the file system on the SCSI disk:

```bash
# 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:

```bash
# chreipl 0.0.7e78 -L 1
Re-IPL type: ccw
Device:  0.0.7e78
Loadparm: "1"
Bootparms: 
```

• The following examples configure a DASD with bus ID 0.0.7e78 as the re-IPL device. To also display all kernel messages on the console, add ignore_loglevel to the existing kernel parameters in the boot configuration:

```bash
# chreipl 0.0.7e78 -p "ignore_loglevel"
Re-IPL type: ccw
Device:  0.0.7e78
Loadparm: ""
Bootparms: "ignore_loglevel"
```
**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 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 9, “Shutdown actions,” on page 105.

**chshut syntax**

```
chshut halt poff reboot ipl reipl stop vcmd "<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.

- **vcmd "<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 vcmd 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 devices that are managed by zcrypt and modify zcrypt's AP bus attributes.

In sysfs, AP queues are listed as children of a cryptographic device with a name of the form: `card<adapter_id>`. For example, if cryptographic devices with the adapter IDs 00 and 02 are selected, and the domain IDs 0002, 0003 and 000e have been configured, then the following cryptographic devices and AP queues are defined to Linux:

```
/sys/devices/ap/card00
/sys/devices/ap/card00/00.0002
/sys/devices/ap/card00/00.0003
/sys/devices/ap/card00/00.000e
/sys/devices/ap/card02
/sys/devices/ap/card02/02.0002
/sys/devices/ap/card02/02.0003
/sys/devices/ap/card02/02.000e
```

Setting a cryptographic device online or offline with `chzcrypt` affects its AP queues. To display the attributes, use “lszcrypt - Display zcrypt devices” on page 678.

**chzcrypt syntax**

```
chzcrypt -e -d -a <device ID> -p -n -c <timeout> -t <time> -q <domain>
```

Where:

- **-e or --enable**
  sets the given cryptographic devices and AP queues online.

- **-d or --disable**
  sets the given cryptographic devices and AP queues offline.

- **-a or --all**
  sets all available cryptographic devices online or offline.

- **<device ID>**
  specifies a cryptographic device that is to be set online or offline. A cryptographic device 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.
chzcrypt

-\texttt{-t} \texttt{\langle time\rangle} or --\texttt{poll-timeout=}\texttt{\langle time\rangle}
  
  sets the high-resolution polling timer to \texttt{\langle time\rangle} nanoseconds. To display the value, use \texttt{lszcrypt -b}.

-\texttt{-q} or --\texttt{default-domain} \texttt{\langle domain\rangle}

  changes the default domain. Specify the domain as either a hexadecimal or decimal value.

\textbf{Important:} Be sure to enter an existing domain. The Trusted Key Entry (TKE) workstation does not find the cryptographic adapters if a non-existing domain is entered here. All CCA applications use the default domain, and do not work correctly if the specified domain does not exist.

-\texttt{-h} or --\texttt{help}

  displays help information for the command. To view the man page, enter \texttt{man chzcrypt}.

-\texttt{-v} or --\texttt{version}

  displays version information.

\textbf{Examples}

These examples illustrate common uses for \texttt{chzcrypt}.

- To set the cryptographic devices with the adapter IDs 0, 1, 4, 5, and 12 and their associated AP queues online (in decimal notation):

  \begin{verbatim}
  chzcrypt -e 0 1 4 5 12
  \end{verbatim}

  Or, in hexadecimal notation:

  \begin{verbatim}
  chzcrypt -e 0x00 0x01 0x04 0x05 0x0C
  \end{verbatim}

- To set all available cryptographic devices, including all AP queues, offline:

  \begin{verbatim}
  chzcrypt -d -a
  \end{verbatim}

- To set the AP queue defined by adapter ID 00 and domain 77 (0x4d) offline:

  \begin{verbatim}
  chzcrypt -d 00.004d
  \end{verbatim}

- To set the configuration timer for rescanning the AP bus to 60 seconds and disable zcrypt’s poll thread:

  \begin{verbatim}
  chzcrypt -c 60
  \end{verbatim}

  \begin{verbatim}
  chzcrypt -n
  \end{verbatim}

- To change the default domain to 77 (0x4d):

  \begin{verbatim}
  chzcrypt -q 0x4d
  \end{verbatim}

  or

  \begin{verbatim}
  chzcrypt -q 77
  \end{verbatim}
Use the `chzdev` command to configure devices and device drivers on IBM Z. Supported devices include storage devices (DASD and zFCP) and networking devices (QETH, CTC, and LCS). For more examples of how to use the command, see Chapter 29, “Persistent device configuration,” on page 377.

You can apply configuration changes to the active configuration of the currently running system, or to the persistent configuration stored in configuration files:

- Changes to the active configuration are effective immediately. They are lost on reboot, when a device driver is unloaded, or when a device becomes unavailable.
- Changes to the persistent configuration are applied when the system boots, when a device driver is loaded, or when a device becomes available.

By default, `chzdev` applies changes to the active and the persistent configuration.

You can also temporarily remove existing auto-configuration data, see “Managing auto-configuration data” on page 20.

`chzdev` supports enabling and disabling devices, exporting and importing configuration data to and from a file, and displaying a list of available device types and attributes.

**Figure 181: chzdev actions and options**

Notes:

1 This option is intended for utilities that run early in the boot process.

*Figure 181: chzdev actions and options*
Device selection

- `<type>`
- `<device>`
- `<from_dev>-<to_dev>`
- `--by-attrib <key=value>`
- `--by-interface <interface>`
- `--by-node <device_node>`
- `--by-path <path>`
- `--all`
- `--configured`
- `--existing`
- `--online`
- `--offline`

Device type selection

- `<type>`
- `--type`

Attribute settings

- `<attribute=value>`
- `--remove <attrib>`
- `--remove-all`

Notes:

1 Specify at least one of the options
Figure 182: chzdev help functions

where:

- `<type>`
  restricts the scope of an action to the specified device type:
  - Specify a device type and optionally a device ID to work on devices with matching type and ID only.
  - Specify a device type together with the --type option to manage the configuration of the device type itself.

  **Note:**
  As a precaution, use the most specific device type when you configure a device by ID. Otherwise, the same device ID might accidentally match other devices of a different subtype. To get a list of supported device types, use the --list-types option.

- `<device>`
  selects a single device or a range of devices by device ID. Separate multiple IDs or ranges with a comma (,). To select a range of devices, specify the ID of the first and the last device in the range separated by a hyphen (-).

  `-t` or `--type <device_type>`
  selects a device type as target for a configuration or query action.

- `<attribute=value>`
  specifies a device attribute and its value. To specify multiple attributes, separate attribute-value pairs with a blank.

  You can use the --list-attributes option to display a list of available attributes and the --help-attribute to get more detailed information about a specific attribute.

  **Tip:** To specify an attribute that is not known to chzdev, use the --force option.

  `-r` or `--remove <attrib>`
  removes the setting for attribute `<attrib>`.

  **Active configuration**
  For attributes that maintain a list of values, clears all values for that list.

  **Persistent configuration**
  Removes any setting for the specified attribute. When the device or device driver is configured again, the attribute is set to its default value.

  Some attributes cannot be removed.

  `-R` or `--remove-all`
  removes the settings for all attributes of the selected device or device driver.
Active configuration
For attributes that maintain a list of values, clears all values for that list.

Persistent configuration
Removes all attribute settings that can be removed. When the device or device driver is configured
again, the attribute is set to its default value.

Some attributes cannot be removed.

--by-attrib <attrib=value> | <attrib!=value>
selects devices with a specific attribute, <attrib> that has a value of <value>. When specified as
<attrib>!=<value>, selects all devices that do not provide an attribute named <attrib> with a value of
<value>.

Tip: You can use the --list-attributes option to display a list of available attributes and the --help-attribute
to get more detailed information about a specific attribute.

--by-interface <interface>
selects devices by network interface, for example, eth0. <interface> must be the name of an existing
networking interface.

--by-node <device_node>
selects devices by device node, for example, /dev/sda. <device_node> must be the path to the
device node for a block device or character device.

Note: If <device_node> is the device node for a logical device (such as a device mapper device),
lszdev tries to resolve the corresponding physical device nodes. The lsblk tool must be available
for this resolution to work.

--by-path <path>
selects devices by file-system path, for example, /usr. The <path> parameter can be the mount point
of a mounted file system, or a path on that file system.

Note: If the file system that provides <path> is stored on multiple physical devices (such as supported
by btrfs), lszdev tries to resolve the corresponding physical device nodes. The lsblk tool must be
available and the file system must provide a valid UUID for this resolution to work.

--all
selects all existing and configured devices.

--configured
narrow the selection to those devices for which a persistent configuration exists.

--existing
narrow the selection to all devices that are present in the active configuration.

--configured --existing
specifying both --configured and --existing narrows the selection to devices that are present in both
configurations, persistent and active.

--online
narrow the selection to devices that are enabled in the active configuration.

--offline
narrow the selection to devices that are disabled in the active configuration.

-a or --active
applies changes to the active configuration only. The persistent configuration is not changed unless
you also specify --persistent.

Note: Changes to the active configuration are effective immediately. They are lost on reboot, when a
device driver is unloaded, or when a device becomes unavailable.

-p or --persistent
applies changes to the persistent configuration only. The persistent configuration takes effect when
the system boots, when a device driver is loaded, or when a device becomes available.
--auto-config
appplies changes to the auto-configuration. Changes to the auto-configuration take effect when a device becomes available, but do not persist across reboots. This option is primarily intended for use by the boot process. For details about auto-configuration data, see Chapter 3, “Device auto-configuration for Linux in LPAR mode,” on page 19.

--export <filename>|- writes configuration data to a text file called <filename>. If a single hyphen (-) is specified instead of a file name, data is written to the standard output stream. The output format of this option can be used with the --import option. To reduce the scope of exported configuration data, you can select specific devices, a device type, or define whether to export only data for the active or persistent configuration.

--import <filename>|- reads configuration data from <filename> and applies it. If a single hyphen (-) is specified instead of a file name, data is read from the standard input stream. The input format must be the same as the format produced by the --export option.

By default, all configuration data that is read is also applied. To reduce the scope of imported configuration data, you can select specific devices, a device type, or define whether to import only data for the active or persistent configuration.

-a or --apply
applies the persistent configuration of all selected devices and device types to the active configuration.

e or --enable
enables the selected devices. Any steps necessary for the devices to function are taken, for example: create a CCW group device, remove a device from the CIO exclusion list, or set a CCW device online.

Active configuration
Performs all setup steps required for a device to become operational, for example, as a block device or as a network interface.

Persistent configuration
Creates configuration files and settings associated with the selected devices.

d or --disable
disables the selected devices.

Active configuration
Disables the selected devices by reverting the configuration steps necessary to enable them.

Persistent configuration
Removes configuration files and settings associated with the selected devices.

--dry-run
processes the actions and displays command output without changing the configuration of any devices or device types. Combine with --verbose to display details about skipped configuration steps.

--base <path> | <key=value>
changes file system paths that are used to access files. If <path> is specified without an equal sign (=), it is used as base path for accessing files in the active and persistent configuration. If the specified parameter is in <key=value> format, only those paths that begin with <key> are modified. For these paths, the initial <key> portion is replaced with <value>.

Example: lszdev --persistent --base /etc=/mnt/etc

-f or --force
overrides safety checks and confirmation questions, including:

- More than 256 devices selected
- Configuring unknown attributes
- Combining apparently inconsistent settings
--no-root-update
   skips any additional steps that are required to change the root device configuration persistently. Typically such steps include rebuilding the initial RAM disk, or modifying the kernel command line.

-y or --yes
   answers all confirmation questions with "yes".

-q or --quiet
   prints only minimal run-time information.

-l or --list-attributes
   lists all supported device or device type attributes, including a short description. Use the --help-attribute option to get more detailed information about an attribute.

-L or --list-types
   lists the name and a short description for all device types supported by chzdev.

--no-settle
   continue without waiting for udev processing to complete. This option is intended for utilities that run during the early initial RAM disc stage of the boot process, when udev is not fully functional.

-V or --verbose
   prints additional run-time information.

-v or --version
   displays the version number of chzdev, then exits.

-h or --help
   displays help information for the command.

-H or --help-attribute
   displays help information for the command.

Examples

• To enable an FCP device with device number 0.0.198d, WWPN 0x50050763070bc5e3, and LUN 0x4006404600000000, and create a persistent configuration, issue:

   chzdev --enable zfcp-lun 0.0.198d:0x50050763070bc5e3:0x4006404600000000

• To enable the same FCP device without creating a persistent configuration, issue:

   chzdev --enable --active zfcp-lun 0.0.198d:0x50050763070bc5e3:0x4006404600000000

• To export configuration data for all FCP devices to a file called config.txt, issue:

   chzdev zfcp-lun --all --export config.txt

• To enable a QETH device and create a persistent configuration, issue:

   chzdev --enable qeth 0.0.a000:0.0.a001:0.0.a002

• To enable a QETH device without creating a persistent configuration, issue:

   chzdev --enable --active qeth 0.0.a000:0.0.a001:0.0.a002

• To get help for the QETH-device attribute layer2, issue:

   chzdev qeth --help-attribute layer2

• To enable a device that provides networking interface eth0, issue:

   chzdev --by-interface eth0 --active

• To enable DASD 0.0.8000 and create a persistent configuration, issue:
# chzdev -e dasd 8000

- To enable DASDs 0.0.1000 and 0.0.2000 through 0.0.2010, issue:

  # chzdev dasd 1000,200-2010 -e

- To change the dasd device type parameter eer_pages to 14, issue:

  # chzdev dasd --type eer_pages=14

- To remove the persistent use_diag setting of DASD 0.0.8000, issue:

  # chzdev dasd 8000 --remove use_diag --persistent

- To persistently configure the root device, issue:

  # chzdev --by-path / --persistent

See the man page for information about the command exit codes.

Activating a device early during the boot process

Use the zdev:early device attribute to activate a device early during the boot process and to override any existing auto-configuration with a persistent device configuration.

zdev:early=1
The device is activated during the initial RAM disc phase according to the persistent configuration.

zdev:early=0
The device is activated as usual during the boot process. This is the default. If auto-configuration data is present, the device is activated during the initial RAM disc phase according to the auto-configuration.

Example: To assure that the qeth device with bus-ID 0.0.f5f0 is enabled early in the boot process in layer 2 mode, issue:

  # chzdev -e qeth 0.0.f5f0:0.0.f5f1:0.0.f5f2 layer2=1 zdev:early=1

See also Chapter 3, “Device auto-configuration for Linux in LPAR mode,” on page 19.

Files used

The chzdev command uses these files:

/etc/udev/rules.d/
chzdev creates udev rules to store the persistent configuration of devices. File names start with s390x-41-.

/etc/modprobe.d/
chzdev creates modprobe configuration files to store the persistent configuration of certain device types. File names start with s390x-.

/run
holds udev rules that represent auto-configuration data. chzdev can remove entries so the auto-configuration settings do not apply as devices appear on the running Linux instance.
cio_ignore - Manage the I/O exclusion list

Use the cio_ignore command to specify I/O devices that are to be ignored by Linux.

When an instance of Linux on Z boots, it senses and analyzes all available I/O devices. You can use the cio_ignore kernel parameter (see “cio_ignore - List devices to be ignored” on page 758) 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

```
cio_ignore -a -r -A -R -l -i <device_bus_id> -L -k -u -p -h -v
```

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 4, “Kernel and module parameters,” on page 23.

-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:
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 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
```
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**

```
cmsfs-fuse -a
```

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>
ssets 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.

<nnode>
   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

• To set record length 80 for an empty fixed record format file, PROFILE.EXEC, on a z/VM minidisk that is
  mounted on /mnt:

  # setfattr -n user.record_lrecl -v 80 /mnt/PROFILE.EXEC
**cpacfstats** - Monitor CPACF cryptographic activity

Use the `cpacfstats` command to display the number of cryptographic operations that are performed by the Central Processor Assist for Cryptographic Function (CPACF). You can display and enable, disable, or reset specific hardware counters for AES, DES, TDES, SHA, and pseudo random functions.

CPACF performance counters are available on LPARs only.

All counters are initially disabled and must be enabled in the LPAR activation profile on the SE or HMC to measure CPACF activities. There is a slight performance penalty with CPACF counters enabled.

**Note:** All cryptographic counters are bound to physical CPUs. If you use dedicated CPUs for your LPAR, you obtain only your CPACF activities. If your CPUs are shared with other LPARs, their cryptographic activities might distort the results.

**Prerequisites**

- The kernel must be built with the kernel configuration option `CONFIG_PERF_EVENTS`.
- libpfm version 4 or later is required. If your distribution does not provide the libpfm package, fetch the sources from the Git tree on sourceforge:

  ```
  # git clone git://perfmon2.git.sourceforge.net/gitroot/perfmon2/libpfm4
  ```

- On the HMC or SE, authorize the LPAR for each counter set you want to use. Customize the LPAR activation profile and modify the Counter Facility Security Options. You need to activate the "Crypto activity counter set authorization control" checkbox.
- The `cpacfstatsd` daemon must be running. Check the syslog for the message: `cpacfstatsd: Running`. To start the daemon, issue:

  ```
  # cpacfstatd
  ```

  The daemon requires root privileges to open and work with the perf kernel API functions. Issue `man cpacfstatd` for more information about the daemon.

**Note:** The counter value is increased once per API call and also for every additional 4096 bytes of data.

**Setting up the cpacfstats group**

Only root and members of the group cpacfstats are allowed to communicate with the daemon process. You must create the group and add users to it.

1. Create the group cpacfstats:

   ```
   # groupadd cpacfstats
   ```

2. Add all users who are allowed to run the cpacfstats client application to the group:

   ```
   usermod -a -G cpacfstats <user>
   ```

All users in the cpacfstats group are also able to modify the CPACF counter states (enable, disable, reset).
cpacfstats syntax

Where:

- **e or --enable <counter>**
  enables one or all CPACF performance counters. The optional counter argument can be one of:
  - des
    counts all DES- and 3DES-related cipher message CPACF instructions.
  - aes
    counts all AES-related cipher message CPACF instructions.
  - sha
    counts all message digest (that is, SHA-1 through SHA-512) related CPACF instructions.
  - rng
    counts all pseudo-random related CPACF instructions.
  - all
    counts all CPACF instructions.
  If you omit the counter, all performance counters are enabled. Enabling a counter does not reset it. New events are added to the current counter value.

- **d or --disable <counter>**
  disables one or all CPACF performance counters. If you omit the counter, all performance counters are disabled. Disabling a counter does not reset it. The counter value is preserved when a counter is disabled, and counting resumes with the preserved value when the counter is re-enabled.

- **r or --reset <counter>**
  resets one or all CPACF performance counters. If you omit the counter, all performance counters are reset to 0.

- **p or --print <counter>**
  displays the value of one or all CPACF performance counters. If you omit the counter, all performance counters are displayed.

- **h or --help**
  displays help information for the command. To view the command man page, enter
  man cpacfstats.

- **v or --version**
  displays version information for cpacfstats.

The default command is **--print all**.

Examples

• To print status and values of all CPACF performance counters:

```bash
# cpacfstats
des counter: disabled
aes counter: disabled
sha counter: disabled
rng counter: disabled
```
• To enable the AES CPACF performance counter:

```
# cpacfstats --enable aes
aes counter: 0
```

• To enable all CPACF performance counters:

```
# cpacfstats -e
des counter: 0
aes counter: 192
sha counter: 0
rng counter: 0
```

For the already enabled aes counter, the value is not reset.
**cpuplugd - Control CPUs and memory**

Use the cpuplugd command and a set of rules in a configuration file to dynamically enable or disable CPUs. For Linux on z/VM, you can also dynamically add or remove memory.

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.

**Note:** Do not use cpuplugd with NUMA emulation. cpuplugd can distort the balance of CPU assignments to NUMA nodes. See Chapter 27, “NUMA emulation,” on page 367.

You can start cpuplugd from the command line in two ways:

- With the service utility
- From a command line

**Note:** Do not run multiple instances of cpuplugd simultaneously. Depending on your distribution, cpuplugd might be started automatically.

**cpuplugd service utility syntax**

If you run the cpuplugd daemon through the service utility, you configure the daemon through specifications in the `/etc/cpuplugd.conf` configuration file.

```
service cpuplugd
  start
  stop
  status
  restart
```

Where:

**start**

Starts the cpuplugd daemon with the configuration in `/etc/cpuplugd.conf`. Do not run multiple instances of cpuplugd simultaneously. Check the cpuplugd status before starting a new instance.

**stop**

stops the cpuplugd daemon.

**status**

shows current status of cpuplugd.

**restart**

stops and restarts the cpuplugd daemon. Useful to re-read the configuration file when it was changed.

**Examples**

- To stop a running instance of cpuplugd:

  ```
  # service cpuplugd stop
  ```

- To display the status:

  ```
  # service cpuplugd status
  ...
  Active: active (running) ...
  ```

  The command output can vary, depending on your distribution.
**cpuplugd command-line syntax**

You can start cpuplugd through a command interface.

**Before you begin:** Do not run multiple instances of cpuplugd simultaneously. Check the cpuplugd status through the service utility before you issue the cpuplugd command (see “cpuplugd service utility syntax” on page 606).

```plaintext
__cpuplugd__
```

Where:

- **-c or --config <config file>**
  specifies the path to the configuration file with the rule (see “Configuration file structure” on page 607). You can find a commented sample configuration file at `/etc/cpuplugd.conf`.

  **Note:** For kernels earlier than version 4.5, the sample configuration file was 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/cpuplugd.conf`:

  ```bash
  # cpuplugd -c /etc/cpuplugd.conf
  ```

- To run cpuplugd in the foreground with verbose messages and with a configuration file `/etc/cpuplugd.conf`:

  ```bash
  # cpuplugd -V -f -c /etc/cpuplugd.conf
  ```

**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.
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 183 on page 608 has been reduced to the specifications that are required for dynamically enabling or disabling CPUs.

```
UPDATE="10"
CPU_MIN="2"
CPU_MAX="10"
HOTPLUG = "idle < 10.0"
HOTUNPLUG = "idle > 100"
```

Figure 183: Simplified configuration file with CPU hotplug rules

In the configuration file:

**UPDATE**
- specifies the time interval, in seconds, at which cpuplugd evaluates the rules and, if a rule is met, enables or disables CPUs. This variable is also required for controlling memory (see “Basic configuration file for memory control” on page 609).
- In the example, the rules are evaluated every 10 seconds.

**CPU_MIN**
- specifies the minimum number of CPUs. Even if the rule for disabling CPUs is met, cpuplugd does not reduce the number of CPUs to less than this number.
- In the example, the number of CPUs cannot become less than 2.

**CPU_MAX**
- specifies the maximum number of CPUs. Even if the rule for enabling CPUs is met, cpuplugd does not increase the number of CPUs to more than this number. If 0 is specified, the maximum number of CPUs is the number of CPUs available on the system.
- In the example, the number of CPUs cannot become more than 10.

**HOTPLUG**
- specifies the rule for dynamically enabling CPUs. The rule resolves to a boolean true or false. Each time this rule is true, cpuplugd enables one CPU, unless the number of CPUs has already reached the maximum specified with CPU_MAX.
- Setting HOTPLUG to 0 disables dynamically adding CPUs.
- In the example, a CPU is enabled when the idle times of all active CPUs sum up to less than 10.0%. See “Keywords for CPU hotplug rules” on page 610 for information about available keywords.

**HOTUNPLUG**
- specifies the rule for dynamically disabling CPUs. The rule resolves to a boolean true or false. Each time this rule is true, cpuplugd disables one CPU, unless the number of CPUs has already reached the minimum specified with CPU_MIN.
- Setting HOTUNPLUG to 0 disables dynamically removing CPUs.
- In the example, a CPU is disabled when the idle times of all active CPUs sum up to more than 100%. See “Keywords for CPU hotplug rules” on page 610 for information about available keywords.

If one of these variables is set more than once, only the last occurrence is used. These variables are not case sensitive.
If both the HOTPLUG and HOTUNPLUG rule are met simultaneously, HOTUNPLUG is ignored.

**Basic configuration file for memory control**

For Linux on z/VM, you can also use cpuplugd to dynamically add or take away memory. There are several required specifications for memory control.

The configuration file sample of Figure 184 on page 609 was reduced to the specifications that are required for dynamic memory control.

```plaintext
UPDATE="10"
CMM_MIN="0"
CMM_MAX="131072"  # 512 MB
CMM_INC="10240"    # 40 MB
MEMPLUG = "swaprate > 250"
MEMUNPLUG = "swaprate < 10"
```

*Figure 184: Simplified configuration file with memory hotplug rules*

In the configuration file:

**UPDATE**

specifies the time interval, in seconds, at which cpuplugd evaluates the rules and, if a rule is met, adds or removes memory. This variable is also required for controlling CPUs (see "Basic configuration file for CPU control" on page 608).

In the example, the rules are evaluated every 10 seconds.

**CMM_MIN**

specifies the minimum amount of memory, in 4 KB pages, that Linux surrenders to the CMM static page pool (see “Cooperative memory management background” on page 436). 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 436). Even if the MEMUNPLUG rule for removing memory from Linux and adding it to the CMM static page pool is met, cpuplugd does not increase this amount.

In the example, the amount of memory that is surrendered to the static page pool cannot become more than 131072 pages of 4 KB (512 MB).

**CMM_INC**

specifies the amount of memory, in 4 KB pages, that is removed from Linux when the MEMUNPLUG rule is met. Removing memory from Linux increases the amount that is surrendered to the CMM static page pool.

In the example, the amount of memory that is removed from Linux is 10240 pages of 4 KB (40 MB) at a time.

**CMM_DEC**

Optional: specifies the amount of memory, in 4 KB pages, that is added to Linux when the MEMPLUG rule is met. Adding memory to Linux decreases the amount that is surrendered to the CMM static page pool.

If this variable is omitted, the amount of memory that is specified for CMM_INC is used.

In the example, CMM_DEC is omitted and the amount of memory added to Linux is 10240 pages of 4 KB (40 MB) at a time, as specified with CMM_INC.
MEMPLUG specifies the rule for dynamically adding memory to Linux. The rule resolves to a Boolean true or false. Each time this rule is true, cpuplugd adds the number of pages that are specified by CMM_DEC, unless the CMM static page pool already reached the minimum that is specified with CMM_MIN.

Setting MEMPLUG to 0 disables dynamically adding memory to Linux.

In the example, memory is added to Linux if there are more than 250 swap operations per second. See “Keywords for memory hotplug rules” on page 611 for information about available keywords.

MEMUNPLUG specifies the rule for dynamically removing memory from Linux. The rule resolves to a Boolean true or false. Each time this rule is true, cpuplugd removes the number of pages specified by CMM_INC, unless the CMM static page pool already reached the maximum that is specified with CMM_MAX.

Setting MEMUNPLUG to 0 disables dynamically removing memory from Linux.

In the example, memory is removed from Linux when there are less than 10 swap operations per second. See “Keywords for memory hotplug rules” on page 611 for information about available keywords.

If any of these variables are set more than once, only the last occurrence is used. These variables are not case-sensitive.

If both the MEMPLUG and MEMUNPLUG rule are met simultaneously, MEMUNPLUG is ignored.

CMM_DEC and CMM_INC can be set to a decimal number or to a mathematical expression that uses the same algebraic operators and variables as the MEMPLUG and MEMUNPLUG hotplug rules (see “Keywords for memory hotplug rules” on page 611 and “Writing more complex rules” on page 612).

Predefined keywords
There is a set of predefined keywords that you can use for CPU hotplug rules and a set of keywords that you can use for memory hotplug rules.

All predefined keywords are case sensitive.

**Keywords for CPU hotplug rules**
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 611 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 610 and “Keywords for memory hotplug rules” on page 611 for details about these keywords.

Use the suffixes [<n>] to retrieve the data of <n> intervals in the past, where <n> can be in the range 0 - 100.
Examples

cpustat.idle
  yields the current value for the counted idle ticks.

cpustat.idle[1]
  yields the idle ticks as counted one interval ago.

cpustat.idle[5]
  yields the idle ticks as counted five intervals ago.

cpustat.idle - cpustat.idle[5]
  yields the idle ticks during the past five intervals.

time - time[1]
  yields the length of an update interval in seconds.

cpustat.total_ticks - cpustat.total_ticks[5]
  yields the total number of ticks during the past five intervals.

(cpustat.idle - cpustat.idle[5]) / (cpustat.total_ticks - cpustat.total_ticks[5])
  yields the average ratio of idle ticks to total ticks during the past five intervals.

  Multiplying this ratio with 100 yields the percentage of idle ticks during the last five intervals.

  Multiplying this ratio with 100 * onumcpus yields the accumulated percentage of idle ticks for all processors during the last five intervals.

Writing more complex rules

In addition to numbers and keywords, you can use mathematical and Boolean operators, and you can use user-defined variables to specify rules.

  • The keywords of “Predefined keywords” on page 610
  • Decimal numbers
  • The mathematical operators
    +  addition
    -  subtraction
    *  multiplication
    /  division
    <  less than
    >  greater than
  • Parentheses ( and ) to group mathematical expressions
  • The Boolean operators
    &  and
    |  or
    !  not
  • User-defined variables
You can specify complex calculations as user-defined variables, which can then be used in expressions. User-defined variables are case-sensitive and must not match a pre-defined variable or keyword. In the configuration file, definitions for user-defined variables must precede their use in expressions.

Variable names consist of alphanumeric characters and the underscore (\_) character. An individual variable name must not exceed 128 characters. All user-defined variable names and values, in total, must not exceed 4096 characters.

**Examples**

- `HOTPLUG = "loadavg > onumcpus + 0.75"
- `HOTPLUG = "(loadavg > onumcpus + 0.75) & (idle < 10.0)"

```plaintext
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.

```plaintext
# 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] +
vstat.pgscan_direct_movable[0]"
vstat.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])"
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])"
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 185: Sample configuration file for CPU and memory hotplug

Attention: The sample file of Figure 185 on page 614 illustrates the syntax of the configuration file. Useful rules might differ considerably, depending on the workload, resources, and requirements of the system for which they are designed.

After you install cpuplugd with the s390-tools RPM, a commented sample configuration file is available at /etc/cpuplugd.conf. This file is used by the cpuplugd service.
dasdfmt - Format a DASD

Use the dasdfmt command to low-level format ECKD-type direct access storage devices (DASD). dasdfmt uses an ioctl call to the DASD driver to format tracks. A block size (hard sector size) can be specified. The formatting process can take quite a long time (hours for large DASD).

Tips:
- For DASDs that have previously been formatted with dasdfmt, use the dasdfmt quick format mode.
- Use the -p option to monitor the progress.

⚠️ CAUTION: As on any platform, formatting irreversibly destroys data on the target disk. Be sure not to format a disk with vital data unintentionally.

dasdfmt syntax

![Diagram of dasdfmt syntax]

Notes:
1. You are prompted for the block size if you omit it for an option that requires this specification.
2. If neither the -l option nor the -k option are specified, a VOLSER is generated from the device number through which the volume is accessed.

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.
  For the quick and expand modes and for the --check option, you can omit the block size. Otherwise, you are prompted if you do not specify a value for the block size. You can then press Enter to accept 4096 or specify a different value.
Tip: Set `<block_size>` as large as possible (ideally 4096); the net capacity of an ECKD DASD decreases for smaller block sizes. For example, a DASD formatted with a block size of 512 byte has only half of the net capacity of the same DASD formatted with a block size of 4096 byte.

`<node>`
specifies the device node of the device to be formatted, for example, `/dev/dasdzzz`. See “DASD naming scheme” on page 142 for more details about device nodes.

`-d <disklayout>` or `--disk_layout=<disklayout>`
formats the device with the compatible disk layout (cdl) or the Linux disk layout (ldl). If the parameter is not specified, the default (cdl) is used.

`-L` or `--no_label`
valid for `-d ldl` only, where it suppresses the default LNX1 label.

`-l <volser>` or `--label=<volser>`
specifies the volume serial number (see VOLSER) to be written to the disk. If the VOLSER contains special characters, it must be enclosed in single quotation marks. In addition, any '$' character in the VOLSER must be preceded by a backslash (`\`).

`-k` or `--keep_volser`
keeps the volume serial number when writing the volume label (see VOLSER). Keeping the volume serial number is useful, for example, if the volume serial number was written with a z/VM tool and should not be overwritten.

`-M` or `--mode=<mode>`
specifies the mode to be used for formatting the device. Valid modes are:

- **full**
  Format the entire disk with the specified block size. This is the default mode.

- **quick**
  formats the first two tracks and writes label and partition information. Only use this option if you are sure that the target DASD already contains a regular format with the specified block size.

- **expand**
  format all unformatted tracks at the end of the target DASD. This mode assumes that tracks at the beginning of the DASD volume have already been correctly formatted, while a consecutive set of tracks at the end are unformatted. You can use this mode to make added space available for Linux use after dynamically increasing the size of a DASD volume.

  For the quick and expand modes, omit the block size specification (`-b` option) to use the existing block size. If you specify a block size, `dasdfmt` checks that the specification matches the existing block size before formatting.

- **p** or `--progressbar`
displays a progress bar. Do not use this option if you are using a line-mode terminal console driver. For example, if you are using a 3215 terminal device driver or a line-mode hardware console device driver.

- **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).

- **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.

- **--check**
  performs a complete format check on a DASD volume.

    Omit the block size specification (`-b` option) to check for a consistent format for any valid block size. Specify a block size to confirm that the DASD has been formatted consistently with that particular block size.

- **y**
  starts formatting immediately without prompting for confirmation.
**-F or --force**
formats the device without checking whether it is mounted.

**-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.

**-C or --check_host_count**
checks the host-access open count to ensure that the device is not online to another operating system instance. Use this option to ensure that the operation is safe, and cancel it if other operating system instances are accessing the volume.

**-- norecordzero**
prevents a format write of record zero. This option is intended for experts: Subsystems in DASD drivers are by default granted permission to modify or add a standard record zero to each track when needed. Before you revoke the permission with this option, you must ensure that the device contains standard record zeros on all tracks.

**-V or --version**
displays the version number of dasdfmt and exits.

**-h or --help**
displays an overview of the syntax. Any other parameters are ignored. To view the man page, enter man dasdfmt.

**Examples**

- To format a 100 cylinder z/VM minidisk with the standard Linux disk layout and a 4 KB blocksize with device node /dev/dasdc:

```
# dasdfmt -b 4096 -d ldl -p /dev/dasdc
Drive Geometry: 100 Cylinders * 15 Heads = 1500 Tracks

I am going to format the device /dev/dasdc in the following way:
  Device number of device : 0x192
  Labelling device        : yes
  Disk label              : LNX1
  Disk identifier         : 0X0192
  Extent start (trk no)   : 0
  Extent end (trk no)     : 1499
  Compatible Disk Layout  : no
  Blocksize               : 4096
  Mode                    : Full

--->> ATTENTION! <<---
All data of that device will be lost.
Type yes to continue, no will leave the disk untouched: yes
 Formatting the device. This may take a while (get yourself a coffee).
cyl 100 of 100 |################################|100% [1s]

Finished formatting the device.
Rereading the partition table... ok
```

- To format the same disk with the compatible disk layout (accepting the default value of the -d option).

```
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
Mode                    : Full

--- ATTENTION! ----
All data of that device will be lost.
Type yes to continue, no will leave the disk untouched: yes

Formatting the device. This may take a while (get yourself a coffee).

cyl  100 of  100 |#################################|100% [2s]

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

• To format a previously formatted DASD in quick format mode.

  # dasdfmt -b 4096 -p --mode=quick /dev/dasdf

• To format tracks that have been added at the end of an already formatted DASD.

  # dasdfmt -b 4096 -p --mode=expand /dev/dasdg

• To check whether a DASD has been correctly formatted with a block size of 4096 bytes.

  # dasdfmt -b 4096 -p --check /dev/dasdg

  Checking format of the entire disk...
cyl 1113 of 1113 |#################################|100% [19s]

  Done. Disk is fine.

• To ensure that the DASD is not online to an operating system instance in a different LPAR when you start formatting the DASD:

  # dasdfmt -b 4096 -p -C /dev/dasdh
dasdfmt always checks the host-access open count. If the count indicates access by another operating system instance, the response depends on the -C option. With this option, the command is canceled. Otherwise, a warning is displayed before you are prompted to confirm that you want to proceed.
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 146).
- You need a kernel that supports debugfs and debugfs must be mounted.

Dasdstat syntax

```
dasdstat
```

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.

1 Omit the -e, -d, and -r options to read statistics.
limits the command to the specified items. For \texttt{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.

\textbf{-v or --version}\n
displays the version number of \texttt{dasdstat}, then exits.

\textbf{-h or --help}\n
displays help information for the command.

\textbf{Examples}\n
- This command starts data collection for dasda, 0.0.b301, and for a summary of all available DASDs.

\texttt{# dasdstat -e dasda 0.0.b301 0.0.b302 global}

- This command resets the statistics counters for dasda.

\texttt{# 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
> Histogram of sizes (512B secs)
> 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 0 0
> Histogram of I/O times (microseconds)
> 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
> Histogram of I/O time till ssch
> 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 0 0 0
> Histogram of I/O time between ssch and irq
> 0 3623 468 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 0 0 0
> # of req in chan at enqueuing (0..31)
> 0 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

For details about the data items, see “Interpreting the data rows” on page 163.
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 138.

dasdview syntax

```
-dasdview
  -b 0
  -b <begin>
  -s 128
  -s <size>
  -1
  -2
  -i
  -x
  -j
  -l
  -c
  -t <spec>

<node>
```

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 with the compatible disk layout option (-d cdl. In this situation, the DASD driver pads shorter blocks with zeros to maintain a constant blocksize. All Linux applications (including dasdview) process according to this rule.
Here are some examples of how this option can be used:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-b 32</td>
<td>(start printing at Byte 32)</td>
</tr>
<tr>
<td>-b 32k</td>
<td>(start printing at kByte 32)</td>
</tr>
<tr>
<td>-b 32m</td>
<td>(start printing at MByte 32)</td>
</tr>
<tr>
<td>-b 32b</td>
<td>(start printing at block 32)</td>
</tr>
<tr>
<td>-b 32t</td>
<td>(start printing at track 32)</td>
</tr>
<tr>
<td>-b 32c</td>
<td>(start printing at cylinder 32)</td>
</tr>
</tbody>
</table>

- **-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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-s 16</td>
<td>(use a 16 Byte size)</td>
</tr>
<tr>
<td>-s 16k</td>
<td>(use a 16 kByte size)</td>
</tr>
<tr>
<td>-s 16m</td>
<td>(use a 16 MByte size)</td>
</tr>
<tr>
<td>-s 16b</td>
<td>(use a 16 block size)</td>
</tr>
<tr>
<td>-s 16t</td>
<td>(use a 16 track size)</td>
</tr>
<tr>
<td>-s 16c</td>
<td>(use a 16 cylinder size)</td>
</tr>
</tbody>
</table>

- **-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 compatible disk layout format)

  **VTOC pointer**
  - is only valid for 'VOL1' labels

  **ldl_version**
  - is only valid for 'LNX1' labels

  **formatted_blocks**
  - is only valid for 'LNX1' labels and when the (EBCDIC) ldl_version field 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.

<nnode>
specifies the device node of the device for which you want to display information, for example, /dev/dasdzzz. See “DASD naming scheme” on page 142 for more details about device nodes).

-h or --help
displays short usage text on console. To view the man page, enter man dasdview.

-v or --version
displays version number on console, and exit.

Examples
• To display basic information about a DASD:

  # dasdview -i /dev/dasdzzz

  This example displays:

  --- general DASD information -----------------------------------------------
  device node       : /dev/dasdzzz
  busid             : 0.0.0193
  type              : ECKD
  device type       : hex 3390       dec 13200

  --- DASD geometry --------------------------------------------------------
  number of cylinders : hex 64         dec 100
  tracks per cylinder : hex f          dec 15
  blocks per track    : hex c           dec 12
  blocksize           : hex 1000       dec 4096

• To display device characteristics:

  # dasdview -c /dev/dasda

  This example displays:
encrypted disk : no
solid state device : no

• To include extended information:

```bash
# 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 242d1502 dfe8001
0677080f 007f4a00 1b350000 00000000
configuration_data : dc010100 4040f2f2f1 f0f54040 40c9c2d4
dc000000 4040f2f1 f0f54040 40c9c2d4
f1f3f0f0 f0f0f0f0 f0c6c3f1 f1f30509
dc000000 4040f2f1 f0f54040 40c9c2d4
f1f3f0f0 f0f0f0f0 f0c6c3f1 f1f30509
dc000000 4040f2f1 f0f54040 40c9c2d4
f1f3f0f0 f0f0f0f0 f0c6c3f1 f1f30509
f0000001 4040f2f1 f0f54040 40c9c2d4
f1f3f0f0 f0f0f0f0 f0c6c3f1 f1f30509
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
800000a1 0001e00 51400009 0999a188
0140c009 7cb76b7 00000000 00000000
```

• To display volume label information for a disk formatted with the compatible disk layout:

```bash
# dasdview -l /dev/dasdzzz
```

This example displays:
--- volume label -----------------------------------------------

<table>
<thead>
<tr>
<th>Field</th>
<th>ASCII</th>
<th>EBCDIC</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>volume label key</td>
<td>'åÖÓñ'</td>
<td>VOL1</td>
<td>e5d6d3f1</td>
</tr>
<tr>
<td>volume label identifier</td>
<td>'åÖÓñ'</td>
<td>VOL1</td>
<td>e5d6d3f1</td>
</tr>
<tr>
<td>volume identifier</td>
<td>'ðçðñùó'</td>
<td>0X0193</td>
<td>f0e7f0f1f9f3</td>
</tr>
<tr>
<td>security byte</td>
<td></td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>VTOC pointer</td>
<td></td>
<td>0000000101</td>
<td>cyl 0, trk 1, blk 1</td>
</tr>
<tr>
<td>reserved</td>
<td>'@@@@@'</td>
<td></td>
<td>40404040</td>
</tr>
<tr>
<td>CI size for FBA</td>
<td>'@@@@@'</td>
<td></td>
<td>40404040</td>
</tr>
<tr>
<td>blocks per CI (FBA)</td>
<td>'@@@@@'</td>
<td></td>
<td>40404040</td>
</tr>
<tr>
<td>labels per CI (FBA)</td>
<td>'@@@@@'</td>
<td></td>
<td>40404040</td>
</tr>
<tr>
<td>reserved</td>
<td>'@@@@@'</td>
<td></td>
<td>40404040</td>
</tr>
</tbody>
</table>
| owner code for VTOC       | '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
| reserved                  | '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
| reserved                  | '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
| ld1_version              | '0'   |        | 40     |
| formatted_blocks         |       | 16565899579919558117 | e5e5e5e5e5e5e5e5# |

• To display volume label information for a disk formatted with the Linux disk layout:

dasdview
• To display partition information:

```bash
# dasdview -t info /dev/dasdzz
```

This example displays:
--- VTOC info ---------------------------------------------------------------
The VTOC contains:
| format 1 label(s) |
| format 4 label(s) |
| format 5 label(s) |
| format 7 label(s) |
| format 8 label(s) |
| format 9 label(s) |

Other S/390 and zSeries operating systems would see the following data sets:

<table>
<thead>
<tr>
<th>data set</th>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINUX.V0X0193.PART0001.NATIVE</td>
<td>trk</td>
<td>trk</td>
</tr>
<tr>
<td>data set serial number : '0X0193'</td>
<td>2</td>
<td>500</td>
</tr>
<tr>
<td>system code : 'IBM LINUX '</td>
<td>cyl/trk</td>
<td>cyl/trk</td>
</tr>
<tr>
<td>creation date : year 2009, day 55</td>
<td>0/ 2</td>
<td>33/ 5</td>
</tr>
<tr>
<td>LINUX.V0X0193.PART0002.NATIVE</td>
<td>trk</td>
<td>trk</td>
</tr>
<tr>
<td>data set serial number : '0X0193'</td>
<td>501</td>
<td>900</td>
</tr>
<tr>
<td>system code : 'IBM LINUX '</td>
<td>cyl/trk</td>
<td>cyl/trk</td>
</tr>
<tr>
<td>creation date : year 2009, day 55</td>
<td>33/ 6</td>
<td>60/ 0</td>
</tr>
<tr>
<td>LINUX.V0X0193.PART0003.NATIVE</td>
<td>trk</td>
<td>trk</td>
</tr>
<tr>
<td>data set serial number : '0X0193'</td>
<td>901</td>
<td>1499</td>
</tr>
<tr>
<td>system code : 'IBM LINUX '</td>
<td>cyl/trk</td>
<td>cyl/trk</td>
</tr>
<tr>
<td>creation date : year 2009, day 55</td>
<td>60/ 1</td>
<td>99/ 14</td>
</tr>
</tbody>
</table>

• To display VTOC format 4 label information:

```bash
# dasdview -t f4 /dev/dasdzzz
```

This example displays:

--- VTOC format 4 label ------------------------------------------------------

<table>
<thead>
<tr>
<th>DS4KEYCD</th>
<th>DS4IDFMT</th>
<th>DS4HPCHR</th>
<th>DS4DSREC</th>
<th>DS4HCCHH</th>
<th>DS4NOATK</th>
<th>DS4VTOCI</th>
<th>DS4NOEXT</th>
<th>DS4SMSFG</th>
<th>DS4DEVAC</th>
<th>DS4DSYL</th>
<th>DS4DSTRK</th>
<th>DS4DEVF2</th>
<th>DS4AMTIM</th>
<th>DS4AMCAT</th>
<th>DS4R2TIM</th>
<th>res1</th>
<th>DS4F6PTR</th>
<th>DS4VTOCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>040404040404040404040404040404040404040404040404040404040404040404...</td>
<td>dec 244, hex f4</td>
<td>0000000105 (cyl 0, trk 1, blk 5)</td>
<td>dec 7, hex 0007</td>
<td>0000000000 (cyl 0, trk 0)</td>
<td>dec 0, hex 0000</td>
<td>dec 1, hex 01</td>
<td>dec 0, hex 01</td>
<td>dec 0, hex 00</td>
<td>dec 0, hex 00</td>
<td>dec 15, hex 006f</td>
<td>dec 58786, hex e5a2</td>
<td>dec 0, hex 00</td>
<td>hex 0000000000000000</td>
<td>hex 0000</td>
<td>hex 0000000000000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS4EFLVL</td>
<td>DS4EFPTR</td>
<td>DS4DCYL</td>
<td>DS4DEVDT</td>
<td>DS4DEVDB</td>
<td>DS4AMTIM</td>
<td>DS4AMCAT</td>
<td>DS4R2TIM</td>
<td>res1</td>
<td>DS4F6PTR</td>
<td>DS4VTOCE</td>
<td>typeind : dec 1, hex 01</td>
<td>seqno : dec 0, hex 00</td>
<td>llimit : hex 00000001 (cyl 0, trk 1)</td>
<td>ulimit : hex 00000001 (cyl 0, trk 1)</td>
<td>res2 : hex 0000000000000000</td>
<td>DS4F6PTR : dec 0, hex 00</td>
<td>DS4AMTIM : hex 0000000000000000</td>
<td>DS4AMCAT : hex 00000000</td>
</tr>
<tr>
<td>DS4EFLVL</td>
<td>DS4EFPTR</td>
<td>DS4DCYL</td>
<td>DS4DEVDT</td>
<td>DS4DEVDB</td>
<td>DS4AMTIM</td>
<td>DS4AMCAT</td>
<td>DS4R2TIM</td>
<td>res1</td>
<td>DS4F6PTR</td>
<td>DS4VTOCE</td>
<td>typeind : dec 1, hex 01</td>
<td>seqno : dec 0, hex 00</td>
<td>llimit : hex 00000001 (cyl 0, trk 1)</td>
<td>ulimit : hex 00000001 (cyl 0, trk 1)</td>
<td>res2 : hex 0000000000000000</td>
<td>DS4F6PTR : dec 0, hex 00</td>
<td>DS4AMTIM : hex 0000000000000000</td>
<td>DS4AMCAT : hex 00000000</td>
</tr>
<tr>
<td>DS4EFLVL</td>
<td>DS4EFPTR</td>
<td>DS4DCYL</td>
<td>DS4DEVDT</td>
<td>DS4DEVDB</td>
<td>DS4AMTIM</td>
<td>DS4AMCAT</td>
<td>DS4R2TIM</td>
<td>res1</td>
<td>DS4F6PTR</td>
<td>DS4VTOCE</td>
<td>typeind : dec 1, hex 01</td>
<td>seqno : dec 0, hex 00</td>
<td>llimit : hex 00000001 (cyl 0, trk 1)</td>
<td>ulimit : hex 00000001 (cyl 0, trk 1)</td>
<td>res2 : hex 0000000000000000</td>
<td>DS4F6PTR : dec 0, hex 00</td>
<td>DS4AMTIM : hex 0000000000000000</td>
<td>DS4AMCAT : hex 00000000</td>
</tr>
</tbody>
</table>

• To print the contents of a disk to the console starting at block 2 (volume label):

```bash
# dasdview
```
• 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 |      E000     | E7F6F0F9 F34BD7C1   | X0193.PA  | ?????K?? |
|         57360 |      E010     | D9E3F0F0 F0814BD5   | RT0801.N  | ?????K?? |
|         57368 |      E018     | C1E3C9E5 C5404040   | ???????   | ??????? |
|         57376 |      E020     | 40404040 40404040   | ???????   | ??????? |
|         57384 |      E028     | 40404040 F1F0E7F0   | ???.??.?.. | ??????? |
|         57392 |      E030     | C1E3C9E5 C5404040   | ???????   | ??????? |
|         57400 |      E038     | 63016D01 0000D9C2   | ???????   | ??????? |
|         57408 |      E040     | D4403C9D 5E4E740    | LINUX ?A   | ??????? |
|         57416 |      E048     | 40404065 01300000   | ??????.   | ?????
|         57424 |      E050     | 00000000 80000100   | ...?..    | ...?.. |
|         57432 |      E058     | 00000000 00000800   | ???????   | ??????? |
|         57440 |      E060     | 00000000 00000800   | ???????   | ??????? |
|         57448 |      E068     | 00000000 00000200   | ???????   | ??????? |
|         57456 |      E070     | C1000500 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 00000000 00000000 00000000  | IPL1............ | ?????............ |
| 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 CB6E095E   | "?????o????4??>z; | R??????S??y??n?^ |
| EF49C03C 513542E7 D8F17D90 06DC4F7    | ?{????XQ1'?????7 | ?I?<Q5B????D? |
| 92963D5B 0200B0FA 53745C12 C3B45125   | ko?$?........... | ??=?[?........... |
| 066040C2 F933381E 7A4C4797 F40FEDAB   | ?-?B9????:<?p4?? | ??@??3B?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:
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:
<table>
<thead>
<tr>
<th>cylinder 0, head 0, record 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>count area:</strong></td>
</tr>
<tr>
<td>hex: 0000000000000008</td>
</tr>
<tr>
<td>cylinder: 0</td>
</tr>
<tr>
<td>head: 0</td>
</tr>
<tr>
<td>record: 0</td>
</tr>
<tr>
<td>key length: 0</td>
</tr>
<tr>
<td>data length: 8</td>
</tr>
</tbody>
</table>
+---------------------------+

| key area:                  |
| HEXADECIMAL               |
| 01....04 05....08 09....12 13....16 | EBCDIC | ASCII |
| C9D7D3F1                  | IPL1... | ???... |
+---------------------------+

| data area:                |
| HEXADECIMAL               |
| 01....04 05....08 09....12 13....16 | EBCDIC | ASCII |
| 000A0000 000000F 03000000 00000001 | .?....??... | .?....??... |
+---------------------------+

<table>
<thead>
<tr>
<th>cylinder 0, head 0, record 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>count area:</strong></td>
</tr>
<tr>
<td>hex: 00000000001040018</td>
</tr>
<tr>
<td>cylinder: 0</td>
</tr>
<tr>
<td>head: 0</td>
</tr>
<tr>
<td>record: 1</td>
</tr>
<tr>
<td>key length: 4</td>
</tr>
<tr>
<td>data length: 24</td>
</tr>
</tbody>
</table>
+-------------------------------+

| key area:                    |
| HEXADECIMAL                  |
| 01....04 05....08 09....12 13....16 | EBCDIC | ASCII |
| C9D7D3F1                     | IPL1... | ???... |
+-------------------------------+

| data area:                   |
| HEXADECIMAL                  |
| 01....04 05....08 09....12 13....16 | EBCDIC | ASCII |
| 000A0000 0000000F 03000000 00000001 | .?....??... | .?....??... |
+-------------------------------+
Use the `fdasd` command to manage partitions on ECKD-type DASD that were formatted with the compatible disk layout.

See “`dasdfmt - Format a DASD`” on page 615 for information about formatting a DASD. With `fdasd` you can create, change, and delete partitions, and also change the volume serial number.

`fdasd` checks that the volume has a valid volume label and VTOC. If either is missing or incorrect, `fdasd` re-creates it. See “IBM Z compatible disk layout” on page 139 for details about the volume label and VTOC.

Calling `fdasd` with a node, but without options, enters interactive mode. In interactive mode, you are given a menu through which you can display DASD information, add or remove partitions, or change the volume identifier. Your changes are not written to disk until you type the `write` option on the menu. You can quit without altering the disk at any time before this.

For more information about partitions, see “The IBM label partitioning scheme” on page 138.

**Before you begin:**

- To partition a SCSI disk, use `fdisk` rather than `fdasd`.
- The disk must be formatted with `dasdfmt`, using the compatible disk layout.

\* **Attention:** Careless use of `fdasd` can result in loss of data.

---

**fdasd syntax**

```
```

**Notes:**

1. If neither the `-l` option nor the `-k` option is specified, a VOLSER is generated from the device number through which the volume is accessed.

**Where:**

**-s or --silent**

suppresses messages.

**-r or --verbose**

displays additional messages that are normally suppressed.

**-a or --auto**

auto-creates one partition using the whole disk in non-interactive mode.
-k or --keep_volser
keeps the volume serial number when writing the volume label (see VOLSER). Keeping the serial number is useful, for example, if the volume serial number was written with a z/VM tool and should not be overwritten.

-l <volser> or --label <volser>
specifies the volume serial number (see VOLSER).

A volume serial consists of one through six alphanumeric characters or the following special characters:

$ # @ %

All other characters are ignored. Avoid using special characters in the volume serial. Special characters can cause problems when accessing a disk by VOLSER. If you must use special characters, enclose the VOLSER in single quotation marks. In addition, any '$' character in the VOLSER must be preceded by a backslash ('\').

For example, specify:

-1 'a@b\$c#' to get:
A@B$C#

VOLSER is interpreted as an ASCII string and is automatically converted to uppercase, padded with blanks and finally converted to EBCDIC before 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 IBM 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
- -f3390,4096
- --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:

- -f3390,512
- --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 142 for more details about device nodes.

-C or --check_host_count
checks the host-access open count to ensure that the device is not online to another operating system instance. The operation is canceled if another operating system instance is accessing the device.

-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.
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.

re-creates the VTOC and deletes all partitions.

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.

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
```

quits fdasd without updating the disk. Any changes that you have made (in this session) are discarded.

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:
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
  /dev/dasdzzz1          2      700      699    1  Linux native
  /dev/dasdzzz2        701     1100      400    2  Linux native
  1101     1499      399       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): 2
Using default value 2
Last track or +size[c|k|M] (2-[1499]): +400
You have selected track 700

Command (m for help): n
First track (1 track = 48 KByte) ([701]-1499): 701
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:
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:
This command creates a partition as follows:

<table>
<thead>
<tr>
<th>Device</th>
<th>start</th>
<th>end</th>
<th>length</th>
<th>Id</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/dasdzzz1</td>
<td>2</td>
<td>1499</td>
<td>1498</td>
<td>1</td>
<td>Linux native</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Device</th>
<th>start</th>
<th>end</th>
<th>length</th>
<th>Id</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/dasdzzz1</td>
<td>2</td>
<td>500</td>
<td>499</td>
<td>1</td>
<td>Linux native</td>
</tr>
<tr>
<td>/dev/dasdzzz2</td>
<td>501</td>
<td>1100</td>
<td>600</td>
<td>2</td>
<td>Linux swap</td>
</tr>
<tr>
<td>/dev/dasdzzz3</td>
<td>1101</td>
<td>1499</td>
<td>399</td>
<td>3</td>
<td>Linux native</td>
</tr>
</tbody>
</table>
Use the `hmcdrvfs` command for read-only access to contents in a DVD, CD, or USB-attached storage in the media drive of an HMC.

**Before you begin:**

- The `fuse.hmcdrvfs` file system needs access to device node `/dev/hmcdrv`. This node is created automatically when the `hmcdrv` kernel module is loaded, see Chapter 36, “HMC media device driver,” on page 413.
- On the HMC, the media must be assigned to the associated system image (use menu Access Removable Media).
- In a z/VM environment, the z/VM guest virtual machine must have at least privilege class B. The media must be assigned to the LPAR where the z/VM hypervisor runs.
- For Linux in LPAR mode, the LPAR activation profile must allow issuing SCLP requests.

With the media assigned to your Linux instance, this command creates a `fuse.hmcdrvfs` file system with the media content at the specified mount point.

To unmount file systems that you mounted with `hmcdrvfs`, you can use `fusermount`, whether root or non-root user. See the `fusermount` man page for details.

**hmcdrvfs syntax**

```plaintext
hmcdrvfs <mount-point> <fuse.hmcdrvfs-options> <mount-options> <fuse-options>
```

Where:

**-o or --opt**
FUSE or mount command options; for the FUSE options see the following lists, for mount options see the `mount` man page.

**<fuse.hmcdrvfs-options>**
options specific to the `fuse.hmcdrvfs` file system:

- `hmclang=<language>`
  specifies the language setting on the HMC; for valid values, see the `locale` man page.

- `hmctz=<time zone>`
  specifies the time zone setting on the HMC; for valid values, see the `tzset` man page.

**<mount-options>**
options as available for the `mount` command. See the `mount` man page for details.

**<fuse-options>**
options for FUSE. The following options are supported by the `cmsfs-fuse` command. To use an option, it must also be supported by the version of FUSE that you have.

- `-d` or `-d debug`
  enables debug output (implies `-f`).

- `-f`
  runs the command as a foreground operation.

- `-s`
  disables multi-threaded operation.
-o allow_other
  allows access to the file system by other users.

- o allow_root
  allows access to the file system by root.

- o nonempty
  allows mounts over files and non-empty directories.

- o default_permissions
  enables permission checking by the kernel.

- o fsname=<name>
  sets the file system name.

- o subtype=<type>
  sets the file system type.

- o max_read=<n>
  sets maximum size of read requests.

- o direct_io
  uses direct I/O.

- o kernel_cache
  caches files in the kernel.

- o [no]auto_cache
  enables or disables caching based on modification times.

- o umask=<mask>
  sets file permissions (octal).

- o uid=<n>
  sets the file owner.

- o gid=<n>
  sets the file group.

- o entry_timeout=<secs>
  sets the cache timeout for names. The default is 1.0 second.

- o attr_timeout=<secs>
  sets the cache timeout for attributes. The default is 1.0 second.

- o ac_attr_timeout=<secs>
  sets the auto cache timeout for attributes. The default is the attr_timeout value.

- o max_readahead=<n>
  sets the maximum read ahead value.

- o async_read
  performs reads asynchronously (default).

- o sync_read
  performs reads synchronously.

- o no_remote_lock
  disables remote file locking.

- o intr
  allows requests to be interrupted

- o intr_signal=<num>
  specifies the signal to send on interrupt.

- v or --version
  displays version information for the command.

- h or --help
  displays a short help text, then exits. To view the man page, enter man hmcdrvfs.

The following options for mount policy can be set in the file /etc/fuse.conf file:
mount_max=<number>
sets the maximum number of FUSE mounts allowed for non-root users. The default is 1000.

user_allow_other
allows non-root users to specify the allow_other or allow_root mount options.

Examples
• To mount the contents of the HMC media drive at /mnt/hmc without any special options, use:

```
# hmcdrvfs /mnt/hmc
```

• If the hmcdrv kernel module is not loaded, load it before you issue the hmcdrvfs command:

```
# modprobe hmcdrv
# hmcdrvfs /mnt/hmc
```

• To translate the UID and GID of files on the HMC media drive to your system users and groups along with overriding the permissions, issue, for example:

```
# hmcdrvfs /mnt/hmc -o uid=500 -o gid=1000 -o umask=0337
```

• To speed up transfer rates to frequently accessed directories, use the cache timeout option:

```
# hmcdrvfs /mnt/hmc -o entry_timeout=60
```

• If the HMC is in a different timezone and is configured for a different language use, for example:

```
# hmcdrvfs /mnt/hmc -o hmclang=de_DE -o hmctz=Europe/Berlin
```

• To also disregard any Daylight Saving Time, specifying hours west of the Prime Meridian (Coordinated Universal Time):

```
# hmcdrvfs /mnt/hmc -o hmclang=de_DE -o hmctz="GMT-1"
```

• To unmount the HMC media drive contents mounted on /mnt/hmc, issue:

```
# fusermount -u /mnt/hmc
```
**hyptop - Display hypervisor performance data**

Use the hyptop command to obtain a dynamic real-time view of a hypervisor environment on IBM Z.

It works with both the z/VM 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.

System names provided by hyptop are either LPAR names as shown on the SE or HMC, or z/VM guest IDs that identify z/VM guest virtual machines.

The hyptop command provides two main windows:

- A list of systems that the hypervisor is currently running (sys_list).
- One system in more detail (sys).

You can run hyptop in interactive mode (default) or in batch mode with the -b option.

**Before you begin:** The 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`
  - z/VM: `<debugfs mount point>/s390_hypfs/diag_0c`
    (Required only for management time data, identifiers m and M. See “z/VM fields” on page 648)
  - LPAR: `<debugfs mount point>/s390_hypfs/diag_204`
- You can always monitor the guest operating system where hyptop is running. To monitor any other operating system instances running on the same hypervisor as hyptop, you will need additional permissions:
  - For z/VM: The guest virtual machine must be assigned privilege class B.
  - For LPAR: On the HMC or SE security menu of the LPAR activation profile, select the **Global performance data control** check box.
hytpop syntax

Where:

-w <window name> or --window=<window name>
selects the window to display, either sys or sys_list. Use the options --sys, --fields, and --sort to modify the current window. The last window that is specified with the --window option is used as the start window. The default window is sys_list.

-s <system> or --sys=<system>
selects systems for the current window. If you specify this option, only the selected systems are shown in the window. For the sys window, you can specify only one system. <system> can be an LPAR name as shown on the SE or HMC, or it can be a z/VM guest ID that identifies a z/VM guest virtual machine. Enter hytpop without any options to display the names of all available systems.

-f <field>[::<unit>] or --fields=<field>[::<unit>]
selects fields and units in the current window. The <field> variable is a one letter unique identifier for a field (for example "c" for CPU time). The <unit> variable specifies the unit that is used for the field (for example "us" for microseconds). See “Available fields and units” on page 647 for definitions. If the --fields option is specified, only the selected fields are shown.

Note: If your field specification includes the number sign (#), enclose the specification in double quotation marks. Otherwise, the command shell might interpret the number sign and all characters that follow as a comment.

-S <field> or --sort=<field>
selects the field that is used to sort the data in the current window. To reverse the sort order, specify the option twice. See “Available fields and units” on page 647 for definitions.

-t <type> or --cpu_types=<type>
selects CPU types that are used for dispatch time calculations. See “CPU types” on page 649 for definitions.

-b or --batch_mode
uses batch mode. Batch mode can be useful for sending output from hytpop to another program, a file, or a line mode terminal. In this mode no user input is accepted.

-d <seconds> or --delay=<seconds>
specifies the delay between screen updates.

-n <iterations> or --iterations=<iterations>
specifies the maximum number of screen updates before the program ends.
-h or --help
prints usage information, then exits. To view the man page, enter man hyptop.

-v or --version
displays the version of hyptop, then exits.

Navigating between windows
Use letter or arrow keys to navigate between the windows.

When you start the hyptop command, the sys_list window opens in normal mode. Data is updated at regular intervals, and sorted by dispatch time. You can navigate between the windows as shown in Figure 186 on page 646.

Figure 186: 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 72 on page 646.

<table>
<thead>
<tr>
<th>Arrow key</th>
<th>Letter key equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>[←]</td>
<td>[H]</td>
</tr>
<tr>
<td>[↓]</td>
<td>[J]</td>
</tr>
<tr>
<td>[\uparrow]</td>
<td>[K]</td>
</tr>
<tr>
<td>[\downarrow]</td>
<td>[L]</td>
</tr>
</tbody>
</table>

Selecting data
You can scroll windows and select data rows.

To enter select mode, press the \[\downarrow\] key. The display is frozen so that you can select rows. Select rows by pressing the \[\uparrow\] and \[\downarrow\] keys and mark the rows with the Spacebar. Marked rows are displayed in bold font. Leave the select mode by pressing the \[\uparrow\] 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 \[\downarrow\] key. The sys window for the system opens. The \[\uparrow\] key always returns you to the previous window.

To scroll any window, press the \[\uparrow\] and \[\downarrow\] 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 187 on page 647.

![Figure 187: Accessing the fields and CPU-type selection windows](image)

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 C 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 647, “z/VM fields” on page 648, and “CPU types” on page 649) 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.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Column label</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>core</td>
<td>Core dispatch time per second</td>
</tr>
<tr>
<td>e</td>
<td>the</td>
<td>Thread time per second</td>
</tr>
<tr>
<td>m</td>
<td>mgm</td>
<td>Management time per second</td>
</tr>
<tr>
<td>C</td>
<td>Core+</td>
<td>Total core dispatch time</td>
</tr>
<tr>
<td>E</td>
<td>thE+</td>
<td>Total thread time</td>
</tr>
<tr>
<td>M</td>
<td>Mgm+</td>
<td>Total management time</td>
</tr>
<tr>
<td>o</td>
<td>online</td>
<td>Online time</td>
</tr>
</tbody>
</table>

If multithreading is not available or not enabled, the values for core and for thread are identical.

In the sys_list window only:
### hyptop

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Column label</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>system</td>
<td>Name of the LPAR</td>
</tr>
<tr>
<td>#</td>
<td>#core</td>
<td>Number of cores (sum of initial and reserved)</td>
</tr>
<tr>
<td>T</td>
<td>#The</td>
<td>Number of threads (sum of initial and reserved)</td>
</tr>
</tbody>
</table>

In the sys window only:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Column label</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>coreid</td>
<td>Core identifier (always shown)</td>
</tr>
<tr>
<td>p</td>
<td>type</td>
<td>CPU type. See “CPU types” on page 649</td>
</tr>
<tr>
<td>v</td>
<td>visual</td>
<td>Visualization of core dispatch time per second</td>
</tr>
</tbody>
</table>

### 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:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Column label</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>cpu</td>
<td>CPU time per second</td>
</tr>
<tr>
<td>m</td>
<td>mgm</td>
<td>Management time per second</td>
</tr>
<tr>
<td>C</td>
<td>Cpu+</td>
<td>Total CPU time</td>
</tr>
<tr>
<td>M</td>
<td>Mgm+</td>
<td>Total management time</td>
</tr>
<tr>
<td>o</td>
<td>online</td>
<td>Online time</td>
</tr>
</tbody>
</table>

**Note:** Data for the management time, identifiers m and M, is available only for the z/VM guest virtual machine on which hyptop runs.

In the sys_list window only:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Column label</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>system</td>
<td>Name of the z/VM guest virtual machine (always shown)</td>
</tr>
<tr>
<td>#</td>
<td>#cpu</td>
<td>Number of CPUs</td>
</tr>
<tr>
<td>O</td>
<td>#cpuop</td>
<td>Number of operating CPUs</td>
</tr>
<tr>
<td>u</td>
<td>memuse</td>
<td>Used memory</td>
</tr>
<tr>
<td>a</td>
<td>memmax</td>
<td>Maximum memory</td>
</tr>
<tr>
<td>r</td>
<td>wcur</td>
<td>Current weight</td>
</tr>
<tr>
<td>x</td>
<td>wmax</td>
<td>Maximum weight</td>
</tr>
</tbody>
</table>

In the sys window only:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Column label</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>cpuid</td>
<td>CPU identifier (always shown)</td>
</tr>
<tr>
<td>v</td>
<td>visual</td>
<td>Visualization of CPU time per second</td>
</tr>
</tbody>
</table>
**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:

<table>
<thead>
<tr>
<th>Unit</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>us</td>
<td>Microseconds ((10^{-6} \text{ seconds}))</td>
</tr>
<tr>
<td>ms</td>
<td>Milliseconds ((10^{-3} \text{ seconds}))</td>
</tr>
<tr>
<td>%</td>
<td>Hundreds of a second ((10^{-2} \text{ seconds})) or percent</td>
</tr>
<tr>
<td>s</td>
<td>Seconds</td>
</tr>
<tr>
<td>m</td>
<td>Minutes</td>
</tr>
<tr>
<td>hm</td>
<td>Hours and minutes</td>
</tr>
<tr>
<td>dhm</td>
<td>Days, hours, and minutes</td>
</tr>
</tbody>
</table>

**Units of time:**

**Units of memory:**

<table>
<thead>
<tr>
<th>Unit</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>KiB</td>
<td>Kibibytes ((1 \text{,024 bytes}))</td>
</tr>
<tr>
<td>MiB</td>
<td>Mebibytes ((1 \text{,048,576 bytes}))</td>
</tr>
<tr>
<td>GiB</td>
<td>Gibibytes ((1 \text{,073,741,824 bytes}))</td>
</tr>
</tbody>
</table>

**Other units:**

<table>
<thead>
<tr>
<th>Unit</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>String</td>
</tr>
<tr>
<td>#</td>
<td>Count or number</td>
</tr>
<tr>
<td>vis</td>
<td>Visualization</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Column label</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>IFL</td>
<td>Integrated Facility for Linux. On older machines IFLs might be shown as CPs.</td>
</tr>
<tr>
<td>p</td>
<td>CP</td>
<td>CP processor type.</td>
</tr>
<tr>
<td>u</td>
<td>UN</td>
<td>Unspecified processor type (other than CP or IFL).</td>
</tr>
</tbody>
</table>
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) 7:00:00
system  #core core mgm Coze+ Mgm+ online
(St)  (%) (%) (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 8:12 10:04:24
S05LP02  1 99.09 0.00 269:57 0:00 11:05:58
TRX2CF  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 0.80 0.00 2:22 0:00 11:05:52
S05LP66  3 0.00 0.00 0:00 0:00 11:05:52
```

– 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) 7:00:00
system  #cpu cpu Cpu+ online memuse memmax wcux
(St)  (%) (%) (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 0.00 30:22:55 0.32 0.75 100
T6360113 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
TCP1P    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
```

At the top, the sys and sys_list windows show a list of the CPU types that are used for the current CPU and core dispatch time calculation.

• To start hyptop with the sys window showing performance data for LPAR MYLPAR, enter:

```
# hyptop -w sys -s mylpar
```

The result looks like the following example:
• To start hyptop with the sys_list window in batch mode, enter:

```
# hyptop -b
```

• To start hyptop with the sys_list window in interactive mode, with the fields dispatch time (in milliseconds), and online time (unit default), and sort the output according to online time, enter:

```
# hyptop -f c:ms,o -S o
```

• To start hyptop with the sys_list window in batch mode with update delay 5 seconds and 10 iterations, enter:

```
# hyptop -b -d 5 -n 10
```

• To start hyptop with the sys_list window and use only CPU types IFL and CP for dispatch time calculation, enter:

```
# hyptop -t ifl,cp
```

• To start hyptop on Linux in LPAR mode with the sys_list window and display all LPAR fields, including the thread information, enter:

```
# hyptop -f "#,T,c,e,m,C,E,M,o"
```

The result looks like the following example:

```
13:47:42 cpu-t: IFL(0) CP(38) UN(0) 2=help
system #core #The core the mgm Core+ thE+ Mgm+ online (str) (#) (%) (%) (hm) (hm) (hm) (dhm)
S35LP41 12 24 101.28 179.28 0.28 1056:10 1756:11 8:45 158:04:04
S35LP42 16 32 35.07 40.07 0.44 5194:52 6193:52 12:45 158:04:04
S35LP64 3 3 1.20 1.20 0.00 0:31 0:31 0:00 12:03:54
```

In the example, the Linux instances in LPARs S35LP41 and S35LP43 run with 2 threads per core. The thread time, as the sum of the two threads, exceeds the core dispatch time.

The Linux instance in LPAR S35LP64 does not use simultaneous multithreading.

• To start hyptop on Linux on z/VM with the sys_list window and display a selection of z/VM fields, including the management time, enter:

```
# hyptop -f "#,c,m,C,M,o"
```

The result looks like the following example:
In the example, hyptop runs on a Linux instance in z/VM guest virtual machine G3545010. In the sys_list window, this is the only guest virtual machine for which management data is displayed.

**Scenario**

Perform the steps described in this scenario to start hyptop with the sys window with system MYLPAR with the fields dispatch time (unit milliseconds) and total dispatch time (unit default), sort the output according to the total dispatch time, and then reverse the sort order.

**Procedure**

1. Start hyptop.

   ```
   # hyptop
   ```

2. Go to select mode by pressing the `>` key. The display will freeze.

3. Navigate to the row for the system you want to look (in the example MYLPAR) at using the `^` and `v` keys.

4. Open the sys window for MYLPAR by pressing the `>` key.

5. Press the `F` key to go to the fields selection window:

   ```
   Select Fields and Units
   K S ID UNIT AGG DESCRIPTION
   p * type stz none CPU type
   c * core % sum Core dispatch time per second
   e the % sum Thread time per second
   m * mgm % sum Management time per second
   C core+ hm sum Total core dispatch time
   E thE+ % sum Total thread time
   m mgm+ hm sum Total management time
   o online dhm max Online time
   v * visual vis none Visualization of CPU time per second
   ```

   ```
   12:15:51 MYLPAR cpu-t: IFL(18) CP(3) UN(2) 2=help
   ```

   ```
   Select Fields and Units
   K S ID UNIT AGG DESCRIPTION
   p * type stz none CPU type
   c * core % sum Core dispatch time per second
   e the % sum Thread time per second
   m * mgm % sum Management time per second
   C core+ hm sum Total core dispatch time
   E thE+ % sum Total thread time
   m mgm+ hm sum Total management time
   o online dhm max Online time
   v * visual vis none Visualization of CPU time per second
   ```
Ensure that dispatch time per second and total dispatch time are selected and for dispatch time microseconds are used as unit:

a. Press the P key, the M key, and the V key to disable CPU type, Management time per second, and Visualization.

b. Press the C key to enable Total core dispatch time.

c. Then select the Core dispatch time 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
K S ID UNIT AGG DESCRIPTION
p type str none CPU type
c core ms sum Core dispatch time per second
e the % sum Thread time per second
m mgm % sum Management time per second
C core+ hm sum Total core dispatch time
E thE+ % sum Total thread time
M mgm+ hm sum Total management time
o online dhm max Online time
v visual vis none Visualization of CPU time per second

Press the key twice to return to the sys window.

6. To sort by Total core dispatch time press the keys:

To reverse the sort order, press the keys again:

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

Commands for Linux on Z 653
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 1 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:

```bash
# lschp
CHPID Vary Cfg. Type Cmg Shared PCHID
======================================
... ...
0.40 1 1 1b 2 1 0580 ...
... ...
```

The value under `Cfg.` shows that the channel path is configured (1).
lscpumf - Display information about the CPU-measurement facilities

Use the lscpumf command to display information about the CPU-measurement facilities.

**lscpumf syntax**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i or --info</td>
<td>displays detailed information about available and supported CPU measurement facilities.</td>
</tr>
<tr>
<td>-c or --list-counters</td>
<td>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.</td>
</tr>
<tr>
<td>-C or --list-all-counters</td>
<td>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.</td>
</tr>
<tr>
<td>-s or --list-sampling-events</td>
<td>lists perf raw events that activate the sampling facility.</td>
</tr>
<tr>
<td>-v or --version</td>
<td>displays the version number of lscpumf and exits.</td>
</tr>
<tr>
<td>-h or --help</td>
<td>displays out a short help text, then exits. To view the man page, enter man lscpumf.</td>
</tr>
</tbody>
</table>

**Examples**

- To display the supported facilities, issue:

  ```bash
  # 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 IBM Z hardware, issue:
# lscpumf -C
Perf event counter list for IBM z13
==============================================================================
<table>
<thead>
<tr>
<th>Raw event</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>CPU_CYCLES</td>
<td>Cycle Count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Counter 0 / Basic Counter Set.</td>
</tr>
<tr>
<td>r1</td>
<td>INSTRUCTIONS</td>
<td>Instruction Count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Counter 1 / Basic Counter Set.</td>
</tr>
<tr>
<td>r2</td>
<td>L1I_DIR_WRITES</td>
<td>Level-1 I-Cache Directory Write Count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Counter 2 / Basic Counter Set.</td>
</tr>
<tr>
<td>r3</td>
<td>L1I_PENALTY_CYCLES</td>
<td>Level-1 I-Cache Penalty Cycle Count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Counter 3 / Basic Counter Set.</td>
</tr>
<tr>
<td>r4</td>
<td>L1D_DIR_WRITES</td>
<td>Level-1 D-Cache Directory Write Count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Counter 4 / Basic Counter Set.</td>
</tr>
<tr>
<td>r5</td>
<td>L1D_PENALTY_CYCLES</td>
<td>Level-1 D-Cache Penalty Cycle Count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Counter 5 / Basic Counter Set.</td>
</tr>
<tr>
<td>r20</td>
<td>PROBLEM_STATE_CPU_CYCLES</td>
<td>Problem-State Cycle Count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Counter 32 / Problem-State Counter Set.</td>
</tr>
<tr>
<td>r21</td>
<td>PROBLEM_STATE_INSTRUCTIONS</td>
<td>Problem-State Instruction Count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Counter 33 / Problem-State Counter Set.</td>
</tr>
</tbody>
</table>
...
**lscss - List subchannels**

Use the `lscss` command to gather subchannel information from sysfs and display it in a summary format.

**lscss syntax**

```
lscss
-s  
-u  
--avail
--vpm
--io
--chsc
--eadm
-a  
-t  <devicetype> / <model>
-d  <bus_id>, <from_bus_id>-<to_bus_id>
```

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

<table>
<thead>
<tr>
<th>Device</th>
<th>Subchan.</th>
<th>DevType</th>
<th>CU Type</th>
<th>Use</th>
<th>PIM</th>
<th>PAM</th>
<th>POM</th>
<th>CHPIDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.2f08</td>
<td>0.0.0a78</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td>yes</td>
<td>c0</td>
<td>c0</td>
<td>ff</td>
<td>34400000 00000000</td>
</tr>
<tr>
<td>0.0.2f0e</td>
<td>0.0.0b55</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td></td>
<td>c0</td>
<td>c0</td>
<td>bf</td>
<td>34400000 00000000</td>
</tr>
<tr>
<td>0.0.2f0f</td>
<td>0.0.0b56</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td></td>
<td>c0</td>
<td>c0</td>
<td>bf</td>
<td>34400000 00000000</td>
</tr>
<tr>
<td>0.0.2f10</td>
<td>0.0.0b57</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td>yes</td>
<td>c0</td>
<td>c0</td>
<td>ff</td>
<td>34400000 00000000</td>
</tr>
</tbody>
</table>

• This command limits the output to the subchannel range 0.0.0b00-0.0.0bff:

### lscss 0.0.0b00-0.0.0bff

<table>
<thead>
<tr>
<th>Device</th>
<th>Subchan.</th>
<th>DevType</th>
<th>CU Type</th>
<th>Use</th>
<th>PIM</th>
<th>PAM</th>
<th>POM</th>
<th>CHPIDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.2f0e</td>
<td>0.0.0b55</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td></td>
<td>c0</td>
<td>c0</td>
<td>bf</td>
<td>34400000 00000000</td>
</tr>
<tr>
<td>0.0.2f0f</td>
<td>0.0.0b56</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td></td>
<td>c0</td>
<td>c0</td>
<td>bf</td>
<td>34400000 00000000</td>
</tr>
<tr>
<td>0.0.2f10</td>
<td>0.0.0b57</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td>yes</td>
<td>c0</td>
<td>c0</td>
<td>ff</td>
<td>34400000 00000000</td>
</tr>
</tbody>
</table>

• This command limits the output to subchannels 0.0.0a78 and 0.0.0b57 and shows the availability:

### lscss --avail 0a78,0b57

<table>
<thead>
<tr>
<th>Device</th>
<th>Subchan.</th>
<th>DevType</th>
<th>CU Type</th>
<th>Use</th>
<th>PIM</th>
<th>PAM</th>
<th>POM</th>
<th>CHPIDs</th>
<th>Avail.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.2f08</td>
<td>0.0.0a78</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td>yes</td>
<td>c0</td>
<td>c0</td>
<td>ff</td>
<td>34400000 00000000</td>
<td>good</td>
</tr>
<tr>
<td>0.0.2f10</td>
<td>0.0.0b57</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td>yes</td>
<td>c0</td>
<td>c0</td>
<td>ff</td>
<td>34400000 00000000</td>
<td>good</td>
</tr>
</tbody>
</table>

• This command limits the output to subchannel 0.0.0a78 and prints uppercase output:

### lscss -u 0a78

<table>
<thead>
<tr>
<th>Device</th>
<th>Subchan.</th>
<th>DevType</th>
<th>CU Type</th>
<th>Use</th>
<th>PIM</th>
<th>PAM</th>
<th>POM</th>
<th>CHPIDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.2f08</td>
<td>0.0.0a78</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td>yes</td>
<td>c0</td>
<td>c0</td>
<td>ff</td>
<td>34400000 00000000</td>
</tr>
</tbody>
</table>

• 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

<table>
<thead>
<tr>
<th>Device</th>
<th>Subchan.</th>
<th>DevType</th>
<th>CU Type</th>
<th>Use</th>
<th>PIM</th>
<th>PAM</th>
<th>POM</th>
<th>CHPIDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.2f08</td>
<td>0.0.0a78</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td>yes</td>
<td>c0</td>
<td>c0</td>
<td>ff</td>
<td>34400000 00000000</td>
</tr>
<tr>
<td>0.0.2f0e</td>
<td>0.0.0b55</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td></td>
<td>c0</td>
<td>c0</td>
<td>bf</td>
<td>34400000 00000000</td>
</tr>
<tr>
<td>0.0.2f0f</td>
<td>0.0.0b56</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td></td>
<td>c0</td>
<td>c0</td>
<td>bf</td>
<td>34400000 00000000</td>
</tr>
<tr>
<td>0.0.2f10</td>
<td>0.0.0b57</td>
<td>3390/0a</td>
<td>3990/e9</td>
<td>yes</td>
<td>c0</td>
<td>c0</td>
<td>ff</td>
<td>34400000 00000000</td>
</tr>
<tr>
<td>0.0.6194</td>
<td>0.0.3700</td>
<td>3390/0c</td>
<td>3990/e9</td>
<td></td>
<td>fc</td>
<td>fc</td>
<td>ff</td>
<td>32333435 40410000</td>
</tr>
<tr>
<td>0.0.6195</td>
<td>0.0.3701</td>
<td>3390/0c</td>
<td>3990/e9</td>
<td></td>
<td>fc</td>
<td>fc</td>
<td>ff</td>
<td>32333435 40410000</td>
</tr>
<tr>
<td>0.0.6196</td>
<td>0.0.3702</td>
<td>3390/0c</td>
<td>3990/e9</td>
<td></td>
<td>fc</td>
<td>fc</td>
<td>ff</td>
<td>32333435 40410000</td>
</tr>
<tr>
<td>0.0.6197</td>
<td>0.0.3703</td>
<td>3390/0c</td>
<td>3990/e9</td>
<td></td>
<td>fc</td>
<td>fc</td>
<td>ff</td>
<td>32333435 40410000</td>
</tr>
<tr>
<td>0.2.5600</td>
<td>0.2.0040</td>
<td>1732/01</td>
<td>1731/01</td>
<td>yes</td>
<td>80</td>
<td>80</td>
<td>ff</td>
<td>00</td>
</tr>
</tbody>
</table>

• 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.
lsdasd

**lsdasd - List DASD devices**

Use the `lsdasd` command to gather information about DASD devices from sysfs and display it in a summary format.

**lsdasd syntax**

```
```

Where:

- **-a** or **--offline**
  includes devices that are currently offline.

- **-b** or **--base**
  omits PAV alias devices. Lists only base devices.

- **-s** or **--short**
  strips the bus ID in the command output down to the four-digit device number.

- **-v** or **--verbose**
  Obsolete. This option has no effect on the output.

- **-l** or **--long**
  extends the output to include attributes, the UID, and path information.

- **-c** or **--compat**
  creates output of this command as with versions earlier than 1.7.0.

- **-u** or **--uid**
  includes and sorts output by UID.

- **-H** or **--host_access_list**
  shows information about all operating system instances that use this device.

- `<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
```
• The following command shows information only for the DASD with device number 0x4d80 and strips the bus ID in the command output down to the device number:

```
# lsdasd -s 0.0.4d80
```

<table>
<thead>
<tr>
<th>Bus-ID</th>
<th>Status</th>
<th>Name</th>
<th>Device</th>
<th>Type</th>
<th>BlkSz</th>
<th>Size</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>4d80</td>
<td>active</td>
<td>dasda</td>
<td>94:0</td>
<td>ECKD</td>
<td>4096</td>
<td>4695MB</td>
<td>1202040</td>
</tr>
</tbody>
</table>

• The following command shows only online DASDs in the format of `lsdasd` versions earlier than 1.7.0:

```
# lsdasd -c
0.0.4d80(ECKD) at ( 94: 0) is dasda : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4f19(ECKD) at ( 94: 4) is dasdb : active at blocksize 4096, 5896800 blocks, 23034 MB
0.0.4d81(ECKD) at ( 94: 8) is dasdc : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4d82(ECKD) at ( 94:12) is dasdd : active at blocksize 4096, 1202040 blocks, 4695 MB
0.0.4d83(ECKD) at ( 94:16) is dasde : active at blocksize 4096, 1202040 blocks, 4695 MB
```

• The following command shows the device geometry, UID, path information, and some of the settings for the DASD with device bus-ID 0.0.4d82:

```
lsdasd -l 0.0.4d82
0.0.4d82/dasdd/94:12
  status:                             active
  type:                               ECKD
  blksz:                              4096
  size:                               4695MB
  blocks:                             1202040
  use_diag:                           0
  readonly:                           0
  eer_enabled:                        0
  erplog:                             0
  hpf:                                1
  uid:                                IBM.75000000010671.4d82.16
  paths_installed:                    30 31 32 33 3c 3d
  paths_in_use:                       31 32 33
  paths_nonPreferred:                 3c
  paths_cuir_quiesced:                30
  paths_invalid_cabling:              3c
  paths_invalid_hpf_characteristics:  3d
  paths_error_threshold_exceeded:     
```

In the example, three of the installed paths are unused for different reasons:

- The path with CHPID 3c is not used because of a cabling error to the storage system. This channel path does not connect to the same physical disk space as the other channel path for this device.
- The path with CHPID 30 is not used because of a control-unit initiated reconfiguration (CUIR).
- The path with CHPID 3d is not used because its High Performance FICON characteristics do not match with the paths currently in use.

• The following command shows whether other operating system instances access device 0.0.bf45:
# lsdasd -H bf45

Host information for 0.0 bf45

<table>
<thead>
<tr>
<th>Path-Group-ID</th>
<th>LPAR</th>
<th>CPU</th>
<th>FL</th>
<th>Status</th>
<th>Sysplex</th>
<th>Max_Cyls</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>88000d29e72964ce8570b8</td>
<td>0d</td>
<td>29e7</td>
<td>50</td>
<td>ON</td>
<td>TRX1LNX1</td>
<td>268434453</td>
<td>0</td>
</tr>
<tr>
<td>88000e29e72964ce8570c3</td>
<td>0e</td>
<td>29e7</td>
<td>50</td>
<td>ON</td>
<td></td>
<td>268434453</td>
<td>0</td>
</tr>
<tr>
<td>88000f29e72964ce8570d1</td>
<td>0f</td>
<td>29e7</td>
<td>50</td>
<td>ON</td>
<td></td>
<td>268434453</td>
<td>0</td>
</tr>
<tr>
<td>88011d29e72964ce8570d4</td>
<td>1d</td>
<td>29e7</td>
<td>50</td>
<td>ON</td>
<td></td>
<td>268434453</td>
<td>0</td>
</tr>
<tr>
<td>88011e29e72964ce8570d9</td>
<td>1e</td>
<td>29e7</td>
<td>50</td>
<td>ON</td>
<td></td>
<td>268434453</td>
<td>0</td>
</tr>
<tr>
<td>88011f29e72964ce8570e3</td>
<td>1f</td>
<td>29e7</td>
<td>50</td>
<td>ON</td>
<td></td>
<td>268434453</td>
<td>0</td>
</tr>
<tr>
<td>88022d29e72964ce8570e6</td>
<td>2d</td>
<td>29e7</td>
<td>50</td>
<td>ON</td>
<td></td>
<td>268434453</td>
<td>0</td>
</tr>
<tr>
<td>88022e29e72964ce8570e9</td>
<td>2e</td>
<td>29e7</td>
<td>50</td>
<td>ON</td>
<td></td>
<td>268434453</td>
<td>0</td>
</tr>
<tr>
<td>88022f29e72964ce8570f1</td>
<td>2f</td>
<td>29e7</td>
<td>50</td>
<td>ON</td>
<td></td>
<td>268434453</td>
<td>0</td>
</tr>
<tr>
<td>88033d29e72964ce8570f7</td>
<td>3d</td>
<td>29e7</td>
<td>50</td>
<td>ON</td>
<td></td>
<td>268434453</td>
<td>0</td>
</tr>
<tr>
<td>88033e29e72964ce8570fe</td>
<td>3e</td>
<td>29e7</td>
<td>50</td>
<td>ON</td>
<td></td>
<td>268434453</td>
<td>0</td>
</tr>
<tr>
<td>88033f29e72964ce85710e</td>
<td>3f</td>
<td>29e7</td>
<td>50</td>
<td>ON</td>
<td></td>
<td>268434453</td>
<td>0</td>
</tr>
<tr>
<td>880429e72964ce7dce74</td>
<td>42</td>
<td>29e7</td>
<td>00</td>
<td>OFF</td>
<td></td>
<td>65520</td>
<td></td>
</tr>
<tr>
<td>88042a29e72964ce7db69d</td>
<td>4a</td>
<td>29e7</td>
<td>00</td>
<td>OFF</td>
<td></td>
<td>65520</td>
<td></td>
</tr>
<tr>
<td>800003c29e72964ce8481a6</td>
<td>3c</td>
<td>29e7</td>
<td>00</td>
<td>OFF</td>
<td></td>
<td>65520</td>
<td></td>
</tr>
<tr>
<td>80004629e72964ce7f1c13</td>
<td>46</td>
<td>29e7</td>
<td>70</td>
<td>ON-RSV</td>
<td></td>
<td>65520</td>
<td>1424174863</td>
</tr>
</tbody>
</table>

Status values are:

**ON**

The device is online.

**OFF**

The device is offline.

**ON-RSV**

The device is online and reserved.

**OFF-RSV**

The device is offline and reserved by an operating system instance in another LPAR.

The meaning of the columns is as follows:

**Path-group-ID**

A 22-digit hexadecimal number assigned by the operating system when setting the DASD online. This ID uniquely identifies the operating system to the storage server.

**LPAR**

A 2-digit LPAR ID.

**CPU**

A 4-digit CPU ID, as it is defined in the HMC or can be read from /proc/cpuinfo.

**FL**

A 2-digit hexadecimal flag. 0x20 means reserved, 0x50 means online.

**Sysplex**

The 8-character EBCDIC name of the SYSPLEX.

**MAX_CYLS**

The maximum number of cylinders per volume that are supported by the host.

**TIME**

Time the device has been reserved in seconds since July 1, 1970.
**lshmc - List media contents in the HMC media drive**

Use the `lshmc` command to display the contents of the media in the HMC media drive.

**Before you begin:** To be able to use this command, you need the `hmcdrv` module (see Chapter 36, “HMC media device driver,” on page 413).

### lshmc syntax

```
  lshmc <filepath> [-s]
```

Where:

- `<filepath>` specifies a directory or path to a file to be listed. Path specifications are relative to the root of the file system on the media. You can use the asterisk (*) and question mark (?) as wildcards. If this specification is omitted, the contents of the root directory are listed.

- `-s` or `--short` limits the output to regular files in a short listing format. Omits directories, symbolic links, and device nodes and other special files.

- `-v` or `--version` displays version information for the command.

- `-h` or `--help` displays a short help text, then exits. To view the man page, enter `man lshmc`.

### Examples

- To list the files in the root directory of the media in the HMC's media drive, issue:
  ```
  # lshmc
  ```

- If the `hmcdrv` kernel module is not loaded, load it before you issue the `lshmc` command:
  ```
  # modprobe hmcdrv
  # lshmc
  ```

- To list all HTML files in subdirectory `www`, issue:
  ```
  # lshmc /www/*.html
  ```
isluns - Discover LUNs, or show encryption state of attached LUNs

Use the `lsluns` command to list logical unit numbers (LUNs) discovered in the Fibre Channel storage area networks (SAN), or to show the encryption state of zfcp-attached LUNs.

`lsluns` is designed for environments where SCSI devices are attached through the zfcp device driver.

`lsluns` lists all LUNs discovered in the Fibre Channel SAN. See “Discover LUNs in the Fibre Channel storage area network (SAN)” on page 666.

`lsluns` -a shows the encryption state of the attached LUNs. See “Show the encryption state of zfcp-attached LUNs” on page 667.

For all other uses, such as listing attached LUNs or properties other than encryption, use other tools such as:

- `lszfcp -D` See “lszfcp - List zfcp devices” on page 687
- `lszdev zfcp-lun -ii` See “lszdev - Display IBM Z device configurations” on page 682
- `lsscsi -tvxx` See the man page for more details.

Discover LUNs in the Fibre Channel storage area network (SAN)

Discovering LUNs only makes sense for NPIV-enabled FCP devices without zfcp automatic LUN scan. zfcp automatic LUN scan is available as of kernel version 2.6.37, if not disabled with `zfcp.allow_lun_scan=0`. See “Setting up the zfcp device driver” on page 183.

**Note:** Discovering LUNs causes extra SAN traffic for each target port WWPN.

Temporary LUN Attachment

If not attached already, `lsluns` temporarily attaches LUN 0 (or if this fails, the WLUN 0xc101000000000000) during runtime. Do not terminate `lsluns` with a signal. Signals interfere with the removal of temporarily attached LUNs.

Storage Products

Some storage products return a peripheral device type of 31==0x1f with peripheral qualifier 0 in a SCSI standard INQUIRY command for an unmapped FCP LUN 0. Examples are: IBM Storwize® products, including IBM V7000, IBM V840, IBM V9000, and IBM SAN Volume Controller. For `lsluns` to work with such storage products, you must have a host mapping on the storage side, which maps some volume to exported FCP LUN 0x0000000000000000 (Storwize host map property "SCSI ID" 0) for each used FCP-device initiator WWPN. The volume can be a minimum-sized thin-provisioned shared stand-in volume.

Filter the listing by specifying one or more FCP device bus-IDs, target port WWPNs, or both.

**Isluns syntax**

```
lsluns [-c <device_bus_id>] [-p <wwpn>]
```

Where:

- `c <device_bus_id> or --ccw <device_bus_id>`
  - filters LUNs by one or more adapters with the specified FCP device-bus IDs. When used in conjunction with `-p`, only those LUNs are listed that also satisfy at least one of the `-p` constraints.

- `p <wwpn> or --port <wwpn>`
  - filters LUNs by one or more target ports with the specified WWPNs. When used in conjunction with `-c`, only those LUNs are listed that also satisfy at least one of the `-c` constraints.
-v or --version
displays version information and exits.

-h or --help
displays an overview of the syntax. To view the man page, enter man lsluns.

Examples

• This example lists all LUNs discovered in the FC SAN on adapter 0.0.3922:

```
# lsluns -c 0.0.3922
```

• This example shows all LUNs discovered in the FC SAN on target port 0x500507630300c562:

```
# lsluns -p 0x500507630300c562
Scanning for LUNs on adapter 0.0.5922 at port 0x500507630300c562:
  0x4010400000000000
  0x4010400100000000
  0x4010400200000000
  0x4010400300000000
  0x4010400400000000
  0x4010400500000000
```

• This example shows all LUNs discovered in the FC SAN on:
  - Adapter 0.0.3922 and port 0x5005123456789000
  - Adapter 0.0.3922 and port 0x5005abcdefabc000
  - Adapter 0.0.fc00 and port 0x5005123456789000
  - Adapter 0.0.fc00 and port 0x5005abcdefabc000

```
# lsluns -c 0.0.3922 -c 0.0.fc00 -p 0x5005123456789000 -p 0x5005abcdefabc000
```

Show the encryption state of zfcp-attached LUNs

lsluns -a shows the encryption state of the attached LUNs.

Note: Running lsluns -a causes extra SAN traffic for each attached LUN.

Filter the listing by specifying one or more FCP device bus-IDs, target port WWPNs, or both.

lsluns syntax

```
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lsluns</td>
<td></td>
</tr>
<tr>
<td>-a</td>
<td>-a</td>
</tr>
<tr>
<td>-c</td>
<td>-c</td>
</tr>
<tr>
<td>-p</td>
<td>-p</td>
</tr>
<tr>
<td>--active</td>
<td>--active</td>
</tr>
<tr>
<td>--ccw</td>
<td>--ccw</td>
</tr>
<tr>
<td>--port</td>
<td>--port</td>
</tr>
</tbody>
</table>
```

Where:

- **-a or --active**
  shows the encryption state of the attached LUNs. Encrypted devices are indicated with a bracketed X immediately after the LUN number.

- **-c <device_bus_id> or --ccw=<device_bus_id>**
  filters LUNs by one or more adapters with the specified FCP device-bus IDs. When used in conjunction with -p, only those LUNs are listed that also satisfy at least one of the -p constraints.

- **-p <wwpn> or --port=<wwpn>**
  filters LUNs by one or more target ports with the specified WWPNs. When used in conjunction with -c, only those LUNs are listed that also satisfy at least one of the -c constraints.
**lsluns**

- **v** or **--version**
  displays version information and exits.

- **h** or **--help**
  displays an overview of the syntax. To view the man page, enter man  lsluns.

**Examples**

- This example shows the encryption status of attached LUNs:

```
# lsluns -a
adapter = 0.0.3c02
  port = 0x500507630300c562
    lun = 0x401040a200000000(X) /dev/sg0 Disk IBM:2107900
    lun = 0x401040a300000000 /dev/sg1 Disk IBM:2107900
  ...
  port = 0x500507630303c562
adapter = 0.0.593a
  ...
```

The (X) after the LUN number indicates that the device is encrypted.
lsqeth - List qeth-based network devices

Use the lsqeth command to display a summary of information about qeth-based network devices.

**Before you begin:** To be able to use this command, you must also install qethconf (see “qethconf - Configure qeth devices” on page 704). 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_10GIG
  cdev0                   : 0.0.f5a2
  cdev1                   : 0.0.f5a3
  cdev2                   : 0.0.f5a4
  chpid                   : B5
  online                  : 1
  portname                : no portname required
  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
  switch_attrs            : [802.1] rr
  ```

- 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  128    
  0.0.f5a2/0.0.f5a3/0.0.f5a4 xB5   eth0   OSD_10000       0        sw     always_q_2 no no  64    
  0.0.fba2/0.0.fba3/0.0.fba4 xB0   eth1   OSD_10000       0        sw     always_q_2 no no  64    
  ```

---

**lsqeth**

Commands for Linux on Z 669
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.

This example shows the current re-IPL settings:

```
# lsreipl
Re-IPL type:     fcp
WWPN:            0x500507630300c562
LUN:             0x401040b300000000
Device:          0.0.1700
bootprog:        0
br_lba:          0
Loadparm:        ""
Bootparms:        ""
```
**lsscm - List storage-class memory increments**

Use the `lsscm` command to list status and other information about available storage-class memory increments.

**lsscm syntax**

```
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lsscm</code></td>
<td>Displays help information.</td>
</tr>
<tr>
<td><code>-h</code></td>
<td>Displays help information.</td>
</tr>
<tr>
<td><code>--help</code></td>
<td>Displays version information.</td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Displays version information.</td>
</tr>
<tr>
<td><code>--version</code></td>
<td>Displays version information.</td>
</tr>
</tbody>
</table>
```

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:
<table>
<thead>
<tr>
<th>SCM Increment</th>
<th>Size</th>
<th>Name</th>
<th>Rank</th>
<th>D_state</th>
<th>O_state</th>
<th>Pers</th>
<th>ResID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000000000</td>
<td>16384MB</td>
<td>scma</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0000000000000000</td>
<td>16384MB</td>
<td>scmb</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

# lsscm

Linux on Z and LinuxONE: Device Drivers, Features, and Commands Development stream (Kernel 4.19)
**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 9, "Shutdown actions," on page 105.

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 diagram]

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
```
Use the `lstape` command to gather information about tape devices and display it in a summary format. It gathers information about the following types of tape devices:

- **CCW-attached tape devices**
- **Tape drive and medium charger devices** that are available through the sysfs SCSI bus (see “Displaying tape information” on page 231)

For information about SCSI tape devices, the command uses the following sources for the information displayed:

- The IBMtape or the open source lin_tape driver.
- The `sg_inq` command from the scsi/sg3_utils package.

**Note:** Issuing `lstape` without option `--ccw-only` causes extra SAN traffic for each SCSI tape or changer device.

- The st SCSI tape device driver in the Linux kernel.
- The ch SCSI medium changer device driver in the Linux kernel.

**lstape syntax**

```
lstape [-s] [-t <device_type>] --online | --offline
--ccw-only | --scsi-only
--verbose

Notes:
1 specify the first device bus-ID with a leading blank.
```

**Notes:**

- **-s or --shortid**
  strips the "0.<n>." from the device bus-IDs in the command output. For CCW-attached devices only.

- **-t or --type**
  limits the output to information about the specified type or types of CCW-attached devices only.

- **--ccw-only**
  limits the output to information about CCW-attached devices only.

- **--scsi-only**
  limits the output to information about tape devices that are attached to the SCSI bus.

- **--online | --offline**
  limits the output to information about online or offline CCW-attached tape devices only.

- **<device_bus_id>**
  limits the output to information about the specified CCW-attached tape device or devices.
-V or --verbose
For tape devices attached to the SCSI bus only. Displays the serial of the tape and information about
the FCP or virtio-scsi-ccw 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:

```bash
#> lstape
FICON/ESCON tapes (found 1):
<table>
<thead>
<tr>
<th>TapeNo</th>
<th>BusID</th>
<th>CuType/Model</th>
<th>DevType/Model</th>
<th>BlkSize</th>
<th>State</th>
<th>Op</th>
<th>MedState</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0.0480</td>
<td>3480/01</td>
<td>3480/04</td>
<td>auto</td>
<td>UNUSED</td>
<td>---</td>
<td>UNLOADED</td>
</tr>
</tbody>
</table>

SCSI tape devices (found 2):
<table>
<thead>
<tr>
<th>Generic</th>
<th>Device</th>
<th>Target</th>
<th>Vendor</th>
<th>Model</th>
<th>Type</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>sg4</td>
<td>IBMchanger0</td>
<td>0:0:0:0</td>
<td>IBM</td>
<td>03590H11</td>
<td>changer</td>
<td>running</td>
</tr>
<tr>
<td>sg5</td>
<td>IBMtape0</td>
<td>0:0:0:1</td>
<td>IBM</td>
<td>03590H11</td>
<td>tapedrv</td>
<td>running</td>
</tr>
</tbody>
</table>
```

If only the st tape device driver and the ch changer device driver are loaded, the output lists those
names in the device section:

```bash
#> lstape
FICON/ESCON tapes (found 1):
<table>
<thead>
<tr>
<th>TapeNo</th>
<th>BusID</th>
<th>CuType/Model</th>
<th>DevType/Model</th>
<th>BlkSize</th>
<th>State</th>
<th>Op</th>
<th>MedState</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0.0480</td>
<td>3480/01</td>
<td>3480/04</td>
<td>auto</td>
<td>UNUSED</td>
<td>---</td>
<td>UNLOADED</td>
</tr>
</tbody>
</table>

SCSI tape devices (found 2):
<table>
<thead>
<tr>
<th>Generic</th>
<th>Device</th>
<th>Target</th>
<th>Vendor</th>
<th>Model</th>
<th>Type</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>sg0</td>
<td>sch0</td>
<td>0:0:0:0</td>
<td>IBM</td>
<td>03590H11</td>
<td>changer</td>
<td>running</td>
</tr>
<tr>
<td>sg1</td>
<td>st0</td>
<td>0:0:0:1</td>
<td>IBM</td>
<td>03590H11</td>
<td>tapedrv</td>
<td>running</td>
</tr>
</tbody>
</table>
```

This command displays information about all available CCW-attached tapes.

```bash
# lstape --ccw-only
FICON/ESCON tapes (found 1):
<table>
<thead>
<tr>
<th>TapeNo</th>
<th>BusID</th>
<th>CuType/Model</th>
<th>DevType/DevMod</th>
<th>BlkSize</th>
<th>State</th>
<th>Op</th>
<th>MedState</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0.0132</td>
<td>3590/50</td>
<td>3590/11</td>
<td>auto</td>
<td>IN_USE</td>
<td>---</td>
<td>LOADED</td>
</tr>
<tr>
<td>1</td>
<td>0.0.0110</td>
<td>3490/10</td>
<td>3490/40</td>
<td>auto</td>
<td>Unused</td>
<td>---</td>
<td>UNLOADED</td>
</tr>
<tr>
<td>2</td>
<td>0.0.0133</td>
<td>3590/50</td>
<td>3590/11</td>
<td>auto</td>
<td>IN_USE</td>
<td>---</td>
<td>LOADED</td>
</tr>
<tr>
<td>3</td>
<td>0.0.012a</td>
<td>3480/01</td>
<td>3480/04</td>
<td>auto</td>
<td>Unused</td>
<td>---</td>
<td>UNLOADED</td>
</tr>
<tr>
<td>N/A</td>
<td>0.0.01f8</td>
<td>3480/01</td>
<td>3480/04</td>
<td>N/A</td>
<td>Offline</td>
<td>---</td>
<td>N/A</td>
</tr>
</tbody>
</table>
```

This command limits the output to tapes of type 3480 and 3490.

```bash
# lstape -t 3480,3490
FICON/ESCON tapes (found 1):
<table>
<thead>
<tr>
<th>TapeNo</th>
<th>BusID</th>
<th>CuType/Model</th>
<th>DevType/DevMod</th>
<th>BlkSize</th>
<th>State</th>
<th>Op</th>
<th>MedState</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0.0110</td>
<td>3490/10</td>
<td>3490/40</td>
<td>auto</td>
<td>Unused</td>
<td>---</td>
<td>UNLOADED</td>
</tr>
<tr>
<td>3</td>
<td>0.0.012a</td>
<td>3480/01</td>
<td>3480/04</td>
<td>auto</td>
<td>Unused</td>
<td>---</td>
<td>UNLOADED</td>
</tr>
<tr>
<td>N/A</td>
<td>0.0.01f8</td>
<td>3480/01</td>
<td>3480/04</td>
<td>N/A</td>
<td>Offline</td>
<td>---</td>
<td>N/A</td>
</tr>
</tbody>
</table>
```

This command limits the output to those tapes of type 3480 and 3490 that are currently online.

```bash
# lstape -t 3480,3490 --online
FICON/ESCON tapes (found 1):
<table>
<thead>
<tr>
<th>TapeNo</th>
<th>BusID</th>
<th>CuType/Model</th>
<th>DevType/DevMod</th>
<th>BlkSize</th>
<th>State</th>
<th>Op</th>
<th>MedState</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0.0110</td>
<td>3490/10</td>
<td>3490/40</td>
<td>auto</td>
<td>Unused</td>
<td>---</td>
<td>UNLOADED</td>
</tr>
<tr>
<td>3</td>
<td>0.0.012a</td>
<td>3480/01</td>
<td>3480/04</td>
<td>auto</td>
<td>Unused</td>
<td>---</td>
<td>UNLOADED</td>
</tr>
</tbody>
</table>
```

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.
• 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.

```bash
#> lstape --scsi-only --verbose
```

<table>
<thead>
<tr>
<th>Generic Device</th>
<th>Target</th>
<th>Vendor</th>
<th>Model</th>
<th>Type</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>HBA</td>
<td>WWPN</td>
<td>Serial</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sg0</td>
<td>0:0:0:0</td>
<td>IBM</td>
<td>03590H11</td>
<td>tapedrv</td>
<td>running</td>
</tr>
<tr>
<td>0.0.1708</td>
<td>0x500507630040727b</td>
<td>NO/INQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sg1</td>
<td>0:0:0:2</td>
<td>IBM</td>
<td>03590H11</td>
<td>changer</td>
<td>running</td>
</tr>
<tr>
<td>0.0.1708</td>
<td>0x500507630040727b</td>
<td>NO/INQ</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Example details about a zfcp-attached SCSI tape library with multiple paths operated by the IBM lin_tape device driver instead of Linux st or ch.

```bash
# lstape --scsi-only --verbose
```

<table>
<thead>
<tr>
<th>Generic Device</th>
<th>Target</th>
<th>Vendor</th>
<th>Model</th>
<th>Type</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>HBA</td>
<td>WWPN</td>
<td>Serial</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sg0</td>
<td>0:0:0:0</td>
<td>IBM</td>
<td>ULT3580-TD6</td>
<td>tapedrv</td>
<td>running</td>
</tr>
<tr>
<td>0.0.5080</td>
<td>0x2002000e1115c62f</td>
<td>10WT037733</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sg1</td>
<td>0:0:0:1</td>
<td>IBM</td>
<td>3573-TL</td>
<td>changer</td>
<td>running</td>
</tr>
<tr>
<td>0.0.5080</td>
<td>0x2002000e1115c62f</td>
<td>10WT037701</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sg4</td>
<td>0:0:1:0</td>
<td>IBM</td>
<td>00L4U78W6497_LL0</td>
<td>tapedrv</td>
<td>running</td>
</tr>
<tr>
<td>0.0.5080</td>
<td>0x2008000e1115c62f</td>
<td>10WT037733</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sg5</td>
<td>0:0:1:1</td>
<td>IBM</td>
<td>3573-TL</td>
<td>changer</td>
<td>running</td>
</tr>
<tr>
<td>0.0.5080</td>
<td>0x2008000e1115c62f</td>
<td>10WT037701</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sg6</td>
<td>1:0:0:0</td>
<td>IBM</td>
<td>ULT3580-TD6</td>
<td>tapedrv</td>
<td>running</td>
</tr>
<tr>
<td>0.0.5080</td>
<td>0x2008000e1115c62f</td>
<td>10WT037733</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sg7</td>
<td>1:0:0:1</td>
<td>IBM</td>
<td>3573-TL</td>
<td>changer</td>
<td>running</td>
</tr>
<tr>
<td>0.0.5080</td>
<td>0x2008000e1115c62f</td>
<td>10WT037701</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sg2</td>
<td>1:0:1:0</td>
<td>IBM</td>
<td>ULT3580-TD6</td>
<td>tapedrv</td>
<td>running</td>
</tr>
<tr>
<td>0.0.5080</td>
<td>0x2008000e1115c62f</td>
<td>10WT037733</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sg3</td>
<td>1:0:1:1</td>
<td>IBM</td>
<td>3573-TL</td>
<td>changer</td>
<td>running</td>
</tr>
<tr>
<td>0.0.5080</td>
<td>0x2008000e1115c62f</td>
<td>10WT037701</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Data fields for SCSI tape devices**

There are specific data fields for SCSI tape devices.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic</td>
<td>SCSI generic device file for the tape drive (for example <code>/dev/sg0</code>). This attribute is &quot;N/A&quot; if the SCSI generic device driver, sg, is not available.</td>
</tr>
<tr>
<td>Device</td>
<td>Main device file for accessing the tape drive, for example:</td>
</tr>
<tr>
<td></td>
<td>• <code>/dev/st0</code> for a tape drive that is attached through the Linux st device driver</td>
</tr>
<tr>
<td></td>
<td>• <code>/dev/sch0</code> for a medium changer device that is attached through the Linux ch device driver</td>
</tr>
<tr>
<td></td>
<td>• <code>/dev/IBMchanger0</code> for a medium changer that is attached through the IBMtape or lin_tape device driver</td>
</tr>
<tr>
<td></td>
<td>• <code>/dev/IBMtape0</code> for a tape drive that is attached through the IBMtape or lin_tape device driver</td>
</tr>
<tr>
<td>Target</td>
<td>The ID in Linux used to identify the SCSI device.</td>
</tr>
<tr>
<td>Vendor</td>
<td>The vendor field from the tape drive.</td>
</tr>
<tr>
<td>Model</td>
<td>The model field from the tape drive.</td>
</tr>
<tr>
<td>Type</td>
<td>&quot;tapedrv&quot; for a tape drive or &quot;changer&quot; for a medium changer.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>State</td>
<td>The state of the SCSI device in Linux. This state is an internal state of the Linux kernel, any state other than &quot;running&quot; can indicate problems.</td>
</tr>
<tr>
<td>HBA</td>
<td>The bus-ID of the FCP device or of the virtio-scsi-ccw virtual HBA to which the tape drive is attached. &quot;N/A&quot; if the device does not have a sysfs ancestor with subsystem ccw.</td>
</tr>
<tr>
<td>WWPN</td>
<td>The WWPN (worldwide port name) of the tape drive in the SAN. &quot;N/A&quot; if the device is not attached through zfcp.</td>
</tr>
<tr>
<td>Serial</td>
<td>The serial number field from the tape drive. &quot;NO/SG&quot; if the SCSI generic device driver, sg, is not available. &quot;NO/INQ&quot; if sg is available, but the sg_inq command from the scsi/sg3_utils package is unavailable.</td>
</tr>
</tbody>
</table>
lszcrypt - Display zcrypt devices

Use the `lszcrypt` command to display information about cryptographic devices that are managed by zcrypt and its AP bus attributes.

To set the attributes, use “chzcrypt - Modify the zcrypt configuration” on page 587. The following information can be displayed for each cryptographic device:

- The card type
- The online status
- The hardware card type
- The card capability
- The hardware queue depth
- The request count
- The zcrypt submodule or alternative device driver that handles the device

For information about alternative device drivers, see “Marking AP queues for alternative device drivers” on page 508.

The following AP bus attributes can be displayed:

- The default AP domain
- The configuration timer
- The poll thread status
- The zcrypt submodule that handles the queue

lszcrypt syntax

```
lszcrypt
-b
-c <device_ID>
-d
-V <device_ID>
```

Where:

- `<device_ID>` specifies a cryptographic adapter to display. A cryptographic device can be either an adapter ID or an AP queue device. If no devices are specified, information about all available devices is displayed. Both the adapter ID representation and the AP queue device representation are hexadecimal.

- `-b` or `--bus` displays the AP bus attributes.

- `-c <device_ID>` or `--capability <device_ID>` shows the capabilities of a cryptographic adapter. The capabilities depend on the card type and the installed function facilities. A cryptographic device can provide one or more of the following capabilities:
  - RSA 2K Clear Key
  - RSA 4K Clear Key
• CCA Secure Key (full function set)
• CCA Secure Key (restricted function set)
• EP11 Secure Key
• Long RNG

The restricted function set for CCA Secure Key applies to shared adapters for z/VM guests (see “Cryptographic devices on z/VM” on page 499).

The installed functions are shown, as a sequence of letters, in the FUNCTION column of the verbose output mode, with the following meaning:

S  APSC facility available
M and C  RSA 4096 bit support
D  CCA Coprocessor function available
A  Accelerator function available
X  EP11 Coprocessor function available
N  APXA facility available
F  Full function set available
R  Restricted function set (only stateless)

-d or --domains
shows the usage and control domains of the cryptographic device. The displayed domains of the cryptographic device depends on the initial cryptographic configuration.

• "C" indicates a control domain
• "U" indicates a usage domain
• "B" indicates both (control and usage domain)

-V or --verbose
enables the verbose level for cryptographic device information. It displays card type, online status, hardware card type, hardware queue depth, request count, pending request queue count, outstanding request queue count, and installed function facilities.

-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 devices and AP queues:

```
# lszcrypt
```

This command lists all devices grouped by cryptographic device, similar to the following example. The domain IDs are hexadecimal values.
• To display AP bus information:

```bash
# lszcrypt -b
```

This command displays output similar to the following example:

```
ap_domain=0x6
ap_max_domain_id=0x54
ap_interrupts are enabled
config_time=30 (seconds)
poll_thread is disabled
poll_timeout=250000 (nanoseconds)
```

• To display the capabilities for the cryptographic device with adapter ID 0x0b:

```bash
# lszcrypt -c 0x0b
```

This command displays output similar to the following example:

```
Coprocessor card0b provides capability for:
CCA Secure Key (full function set)
RSA 4K Clear Key
Long RNG
```

• To list the usage and control domains of the cryptographic devices:

```bash
# lszcrypt -d
```

This command displays a table that lists all domains (in hex notation) similar to the following example:

```
<table>
<thead>
<tr>
<th>DOMAIN</th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>0a</th>
<th>0b</th>
<th>0c</th>
<th>0d</th>
<th>0e</th>
<th>0f</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

C: Control domain
U: Usage domain
B: Both (Control + Usage domain)

• To display detailed information of all available cryptographic devices:

680 Linux on Z and LinuxONE: Device Drivers, Features, and Commands Development stream (Kernel 4.19)
This example shows two CEX5S cryptographic devices, one in accelerator mode and the other in CCA coprocessor mode (IDs 0x00, 0x01). It also shows a CEX6P device in EP11 coprocessor mode (ID 0x05). The configured domains are 5 (0x05), 31 (0x1f), and 77 (0x4d). Adapter IDs and domain IDs are hexadecimal values.

```
lszcrypt -V
CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS PENDING HWTYPE QDEPTH  FUNCTIONS  DRIVER
--------------------------------------------------------------------------------------------
00          CEX5A Accelerator online    159433      19     11     08  -MC-A-NF-  cex4card
00.0005     CEX5A Accelerator online     53078       5     11     08  -MC-A-NF-  cex4queue
00.001f     CEX5A Accelerator online     53157       5     11     08  -MC-A-NF-  cex4queue
01          CEX5C CCA-Coproc online 0 0 11 08  S--D--NF-  cex4card
01.0005     CEX5C CCA-Coproc online     0 0 11 08  S--D--NF-  -no-driver-
01.001f     CEX5C CCA-Coproc online     0 0 11 08  S--D--NF-  -no-driver-
01.004d     CEX5C CCA-Coproc online 0 0 12 08  S--D--NF-  -no-driver-
05          CEX6P EP11-Coproc online 240073      13     12     08  -----XNF-  cex4card
05.0005     CEX6P EP11-Coproc online 88060       3     12     08  -----XNF-  cex4queue
05.001f     CEX6P EP11-Coproc online 79984       3     12     08  -----XNF-  cex4queue
05.004d     CEX6P EP11-Coproc online 0 0 12 08  -----XNF-  -no-driver-
...```

"-no-driver-" in the DRIVER column means that the AP queue has been marked for use by device drivers other than zcrypt, but currently no such device driver is using it. In the example, all domains for adapter 0x01 and domain 0x004d on all adapters have been marked. AP queues that are not handled by the zcrypt device driver are omitted from the non-verbose listing.

```
# lszcrypt -V 0x00 0x03 0x0b
CARD.DOMAIN TYPE   MODE          STATUS  REQUESTS     PENDING   HWTYPE QDEPTH   FUNCTIONS  DRIVER
------------------------------------------------------------------------------------------------------
00          CEX5A  Accelerator   online         2           0       11     08   -MC-A-NF-  cex4card
00.0041     CEX5A  Accelerator   online         2           0       11     08   -MC-A-NF-  cex4queue
03          CEX5C  CCA-Coproc    online      1631           0       11     08   S--D--NF-  cex4card
03.0041     CEX5C  CCA-Coproc    online      1631           0       11     08   S--D--NF-  cex4queue
0b          CEX5P  EP11-Coproc   online      3088           0       11     08  -----XNF-  cex4card
0b.0041     CEX5P  EP11-Coproc   online      3088           0       11     08  -----XNF-  cex4queue
...```

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 -V 0x00 0x03 0x0b
```

This command displays output similar to the following example:

```
CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS PENDING HWTYPE QDEPTH  FUNCTIONS  DRIVER
------------------------------------------------------------------------------------------------------
00          CEX5A Accelerator online    159433      19     11     08  -MC-A-NF-  cex4card
00.0041     CEX5A Accelerator online     53078       5     11     08  -MC-A-NF-  cex4queue
03          CEX5C CCA-Coproc online 0 0 11 08  S--D--NF-  cex4card
03.0041     CEX5C CCA-Coproc online 1631 0 11 08  S--D--NF-  cex4queue
0b          CEX5P EP11-Coproc online 3088 0 11 08  -----XNF-  cex4card
0b.0041     CEX5P EP11-Coproc online 3088 0 11 08  -----XNF-  cex4queue
```

**Tip:** In the device specification you can also use one-digit hexadecimal or decimal notation. The following specifications are all equivalent:

- 0x0 0x2 0xb
- 0x00 0x02 0x0b
- 0 2 11
**lszdev - Display IBM Z device configurations**

Use the `lszdev` command to display the configuration of devices and device drivers that are specific to IBM Z. Supported device types include storage devices (DASD and zFCP) and networking devices (QETH, CTC, and LCS). For more examples of how to use the command, see Chapter 29, “Persistent device configuration,” on page 377.

Configuration information is taken from three sources: the active configuration of the currently running system, the persistent configuration stored in configuration files, and the auto-configuration from the Support Element (SE). By default, `lszdev` displays information from all available sources. If no auto-configuration data is displayed, the mainframe model does not support such data, or none has been specified. For details about the auto-configuration, see Chapter 3, “Device auto-configuration for Linux in LPAR mode,” on page 19.

The `lszdev` command supports two different views:

- The list view provides overview information for selected devices in list form with configurable columns (default)
- The details view provides detailed per-device information
Device selection

- <type>
- <device>
- <from_dev>-<to_dev>
- --all
- --by-attrib <key=value> | <key!=value>
- --by-interface <interface>
- --by-node <device_node>
- --by-path <path>
- --configured
- --configured
- --configured

Device type selection

- <type> --type

Figure 188: lszdev main syntax

Where:

- <type>
  restricts the output to the specified device type. A device type typically corresponds to a device driver. Multiple device types are sometimes provided for the same driver, for example, both "dasd-eckd" and "dasd-fba" are related to the DASD device driver. You can work with types in the following ways:
  - To display data for devices with matching type and ID only, specify a device type and a device ID, for example:
    ```bash
    # lszdev dasd 0.0.8000
    ```
  - To display the configuration of the device type itself, specify a device type together with the --type option, for example:
    ```bash
    # lszdev dasd --type
    ```

To get a list of supported device types, use the --list-types option.
<device>
limits the output to information about a single device or a range of devices by device ID. To select a
range of devices, specify the ID of the first and the last device in the range separated by a hyphen (-).
Specify multiple IDs or ID ranges by separating IDs with a comma (,).

--all
lists all existing and configured devices. This option is the default.

--by-attrib <key=value> | <key!=value>
s-selects devices with a specified attribute, <key> that has a value of <value>. When specified as <key! =value>, lists all devices that do not provide an attribute named <key> with a value of <value>.

Tip: You can use the --list-attributes option to display a list of available attributes and the --help-attribute to get more detailed information about a specific attribute.

--by-interface <interface>
selects devices by network interface, for example, eth0. The <interface> parameter must be the name
of an existing networking interface.

--by-node <node>
selects devices by device node, for example, /dev/sda. The <node> must be the path to a block
device or character device special file.

Note: If <node> is the device node for a logical device (such as a device mapper device), lszdev tries
to resolve the corresponding physical device nodes. The lsblk tool must be available for this
resolution to work.

--by-path <path>
selects devices by file-system path, for example, /usr. The <path> parameter can be the mount point
of a mounted file system, or a path on that file system.

Note: If the file system that provides <path> is stored on multiple physical devices (such as supported
by btrfs), lszdev tries to resolve the corresponding physical device nodes. The lsblk tool must be
available and the file system must provide a valid UUID for this resolution to work.

--configured
narrowsthe selection to those devices for which a persistent configuration exists.

--existing
narrowsthe selection to devices that are present in the active configuration.

--online
narrowsthe selection to devices that are enabled in the active configuration.

--offline
narrowsthe selection to devices that are disabled in the active configuration.

-a or --active
lists information obtained from the active configuration, that is, information from the running system.

-p or --persistent
lists information from the persistent configuration.

--auto-conf
lists information from the auto-configuration, see Chapter 3, “Device auto-configuration for Linux in
LPAR mode,” on page 19.

-i or --info
displays detailed information about the configuration of the selected device or device type.

-c or --columns <columns>
specifies a comma-separated list of columns to display.

Example:

# lszdev --columns TYPE,ID

Tip: To get a list of supported column names, use the --list-columns option.
-n or --no-headings
suppresses column headings for list output.

--pairs
produces output in <key="value"> format. Use this option to generate output in a format more suitable for processing by other programs. In this format, column values are prefixed with the name of the corresponding column. Values are enclosed in double quotation marks. The lszdev command automatically escapes quotation marks and slashes that are part of the value string.

--base <path> | <key=value>
changes file system paths that are used to access files. If <path> is specified without an equal sign (=), it is used as base path for accessing files in the active and persistent configuration. If the specified parameter is in <key=value> format, only those paths that begin with <key> are modified. For these paths, the initial <key> portion is replaced with <value>.

Example:

```
# lszdev --persistent --base /etc=/mnt/etc
```

-t or --type <device_type>
lists information about a device type. Use this option to display configuration information of a device type instead of a device.

-q or --quiet
prints only minimal run-time information.

-V or --verbose
prints additional run-time information.

-L or --list-types
lists all available device types that you can use with the --type option.

-l or --list-columns
lists all available columns that you can use with the --columns option.

-h or --help
displays help information for the command.

-v or --version
displays the version number of lszdev, then exits.

Input files
The lszdev command uses these input files:

/etc/udev/rules.d/
lszdev reads udev rules that represent the persistent configuration of devices from this directory. The udev rules are named 41-<device subtype>-<id>.rules.

/etc/modprobe.d/
lszdev reads modprobe configuration files that represent the persistent configuration of certain device types from this directory. File names start with s390x-.

/usr
lszdev reads udev rules that represent the auto-configuration of devices from this directory.

Examples

- To display a list of all devices:

```
# lszdev
```

- To return type and ID of root device in machine-readable format:

```
# lszdev --columns TYPE,ID --by-path /
```
lszdev

- To display DASD driver settings:

  ```bash
  # lszdev --type dasd
  ```
lszfcp - List zfcp devices

Use the lszfcp command to gather information about zfcp devices, ports, units, and their associated class devices from sysfs and to display it in a summary format.

**lszfcp syntax**

```
lszfcp
-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.
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
0.0.500c/0x50050763030bc562/0x4010403200000000 1:0:0:0
0.0.3c0c/0x500507630303c562/0x4010403200000000 5:0:2:0
```
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 the systemd service unit `mon_fsstatd`. When the daemon is started as a service unit, it reads the configuration file `/etc/sysconfig/mon_fsstatd`.

**Before you begin:**

- Install the monwriter device driver and set up z/VM to start the collection of monitor sample data. See Chapter 41, “Writing z/VM monitor records,” on page 447 for information about the setup for and usage of the monwriter device driver.
- Customize the configuration file `/etc/sysconfig/mon_fsstatd` if you plan to call it as a systemd service.

The following publications provide general information about DCSSs, DIAG x‘DC’, CP commands, and APPLDATA:

- See z/VM: Saved Segments Planning and Administration, SC24-6322 for general information about DCSSs.
- See z/VM: CP Programming Services, SC24-6272 for information about the DIAG x‘DC’ instruction.
- See z/VM: CP Commands and Utilities Reference, SC24-6268 for information about the CP commands.
- See z/VM: Performance, SC24-6301 for information about monitor APPLDATA.

You can run the `mon_fsstatd` command in two ways.

- Calling `mon_fsstatd` as a systemd service, if systemd is provided by your distribution. This method reads the configuration file `/etc/sysconfig/mon_fsstatd`.
- Calling `mon_fsstatd` manually from a command line.

**mon_fsstatd systemd service syntax**

If you run the `mon_fsstatd` daemon as a systemd service unit, you configure the daemon through specifications in a configuration file.

```
systemctl
  start          mon_fsstatd
  stop
  status
  restart
  reload
  enable
  disable
```

Where:

- **start**
  - starts monitoring of guest file system size, using the configuration in `/etc/sysconfig/mon_fsstatd`.
- **stop**
  - stops monitoring of guest file system size.
mon_fsstatd

status
shows current status of guest file system size monitoring.

restart
stops and restarts monitoring.

reload
reloads the configuration. Use reload to re-read the configuration file when it was changed.

enable
starts the service automatically at boot time.

disable
disables automatic start of the service at boot time.

Configuration file keywords

FSSTAT_INTERVAL="<n>"
specifies the wanted sampling interval in seconds.

Examples of systemd service unit use

This example configuration file for mon_fsstatd (/etc/sysconfig/mon_fsstatd) sets the sampling interval to 30 seconds:

```
FSSTAT_INTERVAL="30"
```

Examples of mon_fsstatd use:

- To start guest file system size monitoring:
  ```
  # systemctl start mon_fsstatd
  ```

- To display the status:
  ```
  # systemctl status mon_fsstatd
  | mon_fsstatd.service - Monitor z/VM guest file system size
  |   Active: active (running) since Wed 2018-02-21 14:52:11 CET; 4s ago
  ```

- To stop guest file system size monitoring:
  ```
  # systemctl stop mon_fsstatd
  ```

- To display the status again and check that monitoring is now stopped:
  ```
  # systemctl status mon_fsstatd
  | mon_fsstatd.service - Monitor z/VM guest file system size
  |   Active: inactive (dead)
  ```

- To restart the daemon and re-read the configuration file:
  ```
  # systemctl restart mon_fsstatd
  ```
**mon_fsstatd command-line syntax**

If you call the mon_fsstatd daemon from the command line, you configure the daemon through command parameters.

```
mon_fsstatd [-i <seconds>] [-a] [-h] [-v]
```

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>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__u64</td>
<td>time_stamp</td>
<td>Time at which the file system data was sampled.</td>
</tr>
<tr>
<td>__u16</td>
<td>data_len</td>
<td>Length of data that follows the header.</td>
</tr>
<tr>
<td>__u16</td>
<td>data_offset</td>
<td>Offset from start of the header to the start of the file system data (that is, to the fields that begin with fs_).</td>
</tr>
<tr>
<td>Type</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>__u16</td>
<td>fs_name_len</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>fs_name</td>
<td>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.</td>
</tr>
<tr>
<td>__u16</td>
<td>fs_dir_len</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>fs_dir</td>
<td>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.</td>
</tr>
<tr>
<td>__u16</td>
<td>fs_type_len</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>fs_type</td>
<td>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.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_bsize</td>
<td>File system block size.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_frsize</td>
<td>Fragment size.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_blocks</td>
<td>Total data blocks in file system.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_bfree</td>
<td>Free blocks in fs.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_bavail</td>
<td>Free blocks avail to non-superuser.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_files</td>
<td>Total file nodes in file system.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_ffree</td>
<td>Free file nodes in fs.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_favail</td>
<td>Free file nodes available to non-superuser.</td>
</tr>
<tr>
<td>__u64</td>
<td>fs_flag</td>
<td>Mount flags.</td>
</tr>
</tbody>
</table>

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 ppppppppffnvvrrmm, where for records that are written by mon_fsstatd, these values are:

- **pppppppp**
  - 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 42, “Reading z/VM monitor records,” on page 451.
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 as a systemd service `mon_procd` or call it manually. When it is called as a systemd service unit, it reads the configuration file `/etc/sysconfig/mon_procd`.

**Before you begin:**

- Install the monwriter device driver and set up z/VM to start the collection of monitor sample data. See Chapter 41, “Writing z/VM monitor records,” on page 447 for information about the setup for and usage of the monwriter device driver.
- Customize the configuration file `/etc/sysconfig/mon_procd` if you plan to call it as a systemd service unit.
- The Linux instance on which the proc_mond daemon 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, SC24-6322 for general information about DCSSs.
- See z/VM: CP Commands and Utilities Reference, SC24-6268 for information about the CP commands.
- See z/VM: Performance, SC24-6301 for information about monitor APPLDATA.

You can run the `mon_procd` command in two ways:

- Calling `mon_procd` as a systemd service. This method reads the configuration file `/etc/sysconfig/mon_procd`.
- Calling `mon_procd` manually from a command line.

### `mon_procd` systemd service syntax

If you run the `mon_procd` daemon as a systemd service unit, you configure the daemon through specifications in a configuration file.

![Diagram](image)

Where:

- **start**
  - starts monitoring of guest process data, using the configuration in `/etc/sysconfig/mon_procd`.
- **stop**
  - stops monitoring of guest process data.
- **status**
  - shows current status of guest process data monitoring.
- **restart**
  - stops and restarts guest process data monitoring.
**reload**
reloads the configuration. Use `reload` to re-read the configuration file when it was changed.

**enable**
starts the service automatically at boot time.

**disable**
disables automatic start of the service at boot time.

**Configuration file keywords**

**PROC_INTERVAL="<n>"**
specifies the wanted sampling interval in seconds.

**PROC="yes | no"**
specifies whether to enable the mon_procd daemon. Set to "yes" to enable the daemon. Anything other than "yes" is interpreted as "no".

**Examples of systemd service unit use**
This example configuration file for mon_procd (`/etc/sysconfig/mon_procd`) sets the process monitoring interval to 60 seconds:

```
PROC_INTERVAL=60
```

Examples of mon_procd use:

- To start guest process data monitoring:

  ```
  # systemctl start mon_procd
  ```

- To display the status:

  ```
  # systemctl status mon_procd
  | mon_procd.service - Monitor Linux on z/VM
  `...
  Active: active (running) since Mon 2018-02-26 12:16:00 CET; 4s ago
  ```

- To stop guest process data monitoring:

  ```
  # systemctl stop mon_procd
  ```

- To display the status again and check that monitoring is now stopped:

  ```
  # systemctl status mon_procd
  | mon_procd.service - Monitor Linux on z/VM
  `...
  Active: inactive (dead)
  ```

- To restart the daemon and re-read the configuration file:

  ```
  # systemctl restart mon_procd
  ```

**mon_procd command-line syntax**
If you call the mon_procd daemon from the command line, you configure the daemon through command parameters.

```
mon_procd -i 60
```

```
mon_procd -i <seconds>
```

```
mon_procd -
```
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 699 for details.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__u64</td>
<td>time_stamp</td>
<td>Time at which the process data was sampled.</td>
</tr>
<tr>
<td>__u16</td>
<td>data_len</td>
<td>Length of data that follows the header.</td>
</tr>
<tr>
<td>__u16</td>
<td>data_offset</td>
<td>Offset from start of the header to the start of the process data.</td>
</tr>
<tr>
<td>__u64</td>
<td>uptime</td>
<td>Uptime of the Linux instance.</td>
</tr>
<tr>
<td>__u32</td>
<td>users</td>
<td>Number of users on the Linux instance.</td>
</tr>
<tr>
<td>Type</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>char[6]</td>
<td>loadavg_1</td>
<td>Load average over the last 1 minute.</td>
</tr>
<tr>
<td>char[6]</td>
<td>loadavg_5</td>
<td>Load average over the last 5 minutes.</td>
</tr>
<tr>
<td>char[6]</td>
<td>loadavg_15</td>
<td>Load average over the last 15 minutes.</td>
</tr>
<tr>
<td>__u32</td>
<td>task_total</td>
<td>Total number of tasks on the Linux instance.</td>
</tr>
<tr>
<td>__u32</td>
<td>task_running</td>
<td>Number of running tasks.</td>
</tr>
<tr>
<td>__u32</td>
<td>task_sleeping</td>
<td>Number of sleeping tasks.</td>
</tr>
<tr>
<td>__u32</td>
<td>task_stopped</td>
<td>Number of stopped tasks.</td>
</tr>
<tr>
<td>__u32</td>
<td>task_zombie</td>
<td>Number of zombie tasks.</td>
</tr>
<tr>
<td>__u32</td>
<td>num_cpus</td>
<td>Number of CPUs.</td>
</tr>
<tr>
<td>__u16</td>
<td>puser</td>
<td>A number that represents (100 * percentage of total CPU time used for normal processes executing in user mode).</td>
</tr>
<tr>
<td>__u16</td>
<td>pnice</td>
<td>A number that represents (100 * percentage of total CPU time used for niced processes executing in user mode).</td>
</tr>
<tr>
<td>__u16</td>
<td>psystem</td>
<td>A number that represents (100 * percentage of total CPU time used for processes executing in kernel mode).</td>
</tr>
<tr>
<td>__u16</td>
<td>pidle</td>
<td>A number that represents (100 * percentage of total CPU idle time).</td>
</tr>
<tr>
<td>__u16</td>
<td>piowait</td>
<td>A number that represents (100 * percentage of total CPU time used for I/O wait).</td>
</tr>
<tr>
<td>__u16</td>
<td>pirq</td>
<td>A number that represents (100 * percentage of total CPU time used for interrupts).</td>
</tr>
<tr>
<td>__u16</td>
<td>psoftirq</td>
<td>A number that represents (100 * percentage of total CPU time used for softirqs).</td>
</tr>
<tr>
<td>__u16</td>
<td>psteal</td>
<td>A number that represents (100 * percentage of total CPU time spent in stealing).</td>
</tr>
<tr>
<td>__u64</td>
<td>mem_total</td>
<td>Total memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>mem_used</td>
<td>Used memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>mem_free</td>
<td>Free memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>mem_buffers</td>
<td>Memory in buffer cache in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>mem_pgpgin</td>
<td>Data read from disk in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>mem_pgpgout</td>
<td>Data written to disk in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>swap_total</td>
<td>Total swap memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>swap_used</td>
<td>Used swap memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>swap_free</td>
<td>Free swap memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>swap_cached</td>
<td>Cached swap memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>swap_pswpin</td>
<td>Pages that are swapped in.</td>
</tr>
<tr>
<td>__u64</td>
<td>swap_pswpout</td>
<td>Pages that are swapped out.</td>
</tr>
</tbody>
</table>
The following is the format of a process information data that is passed to the z/VM monitor stream.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__u64</td>
<td>time_stamp</td>
<td>Time at which the process data was sampled.</td>
</tr>
<tr>
<td>__u16</td>
<td>data_len</td>
<td>Length of data that follows the header.</td>
</tr>
<tr>
<td>__u16</td>
<td>data_offset</td>
<td>Offset from start of the header to the start of the process data.</td>
</tr>
<tr>
<td>__u32</td>
<td>pid</td>
<td>ID of the process.</td>
</tr>
<tr>
<td>__u32</td>
<td>ppid</td>
<td>ID of the process parent.</td>
</tr>
<tr>
<td>__u32</td>
<td>euid</td>
<td>Effective user ID of the process owner.</td>
</tr>
<tr>
<td>__u16</td>
<td>tty</td>
<td>Device number of the controlling terminal or 0.</td>
</tr>
<tr>
<td>__s16</td>
<td>priority</td>
<td>Priority of the process</td>
</tr>
<tr>
<td>__s16</td>
<td>nice</td>
<td>Nice value of the process</td>
</tr>
<tr>
<td>__u32</td>
<td>processor</td>
<td>Last used processor.</td>
</tr>
<tr>
<td>__u16</td>
<td>pcpu</td>
<td>A number that represents (100 * percentage of the elapsed cpu time that is used by the process since last sampling).</td>
</tr>
<tr>
<td>__u16</td>
<td>pmem</td>
<td>A number that represents (100 * percentage of physical memory that is used by the process).</td>
</tr>
<tr>
<td>__u64</td>
<td>total_time</td>
<td>Total cpu time the process used.</td>
</tr>
<tr>
<td>__u64</td>
<td>ctotal_time</td>
<td>Total cpu time the process and its dead child processes used.</td>
</tr>
<tr>
<td>__u64</td>
<td>size</td>
<td>Total virtual memory that is used by the task in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>swap</td>
<td>Swapped out portion of the virtual memory in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>resident</td>
<td>Non-swapped physical memory that is used by the task in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>trs</td>
<td>Physical memory that is devoted to executable code in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>drs</td>
<td>Physical memory that is devoted to other than executable code in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>share</td>
<td>Shared memory that is used by the task in KB.</td>
</tr>
<tr>
<td>__u64</td>
<td>dt</td>
<td>Dirty page count.</td>
</tr>
<tr>
<td>__u64</td>
<td>maj_flt</td>
<td>Number of major page faults occurred for the process.</td>
</tr>
<tr>
<td>char</td>
<td>state</td>
<td>Status of the process</td>
</tr>
<tr>
<td>__u32</td>
<td>flags</td>
<td>The process current scheduling flags.</td>
</tr>
<tr>
<td>__u16</td>
<td>ruser_len</td>
<td>Length of real user name of the process owner and should not be larger than 64.</td>
</tr>
<tr>
<td>char[ruser_len]</td>
<td>ruser</td>
<td>Real user name of the process owner. If the name is longer than 64, the name is truncated to the length 64.</td>
</tr>
<tr>
<td>__u16</td>
<td>euser_len</td>
<td>Length of effective user name of the process owner and should not be larger than 64.</td>
</tr>
<tr>
<td>char[euser_len]</td>
<td>euser</td>
<td>Effective user name of the process owner. If the name is longer than 64, the name is truncated to the length 64.</td>
</tr>
</tbody>
</table>
Table 76: Process data format (continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__u16</td>
<td>egroup_len</td>
<td>Length of effective group name of the process owner and should not be larger than 64.</td>
</tr>
<tr>
<td>char [egroup_len]</td>
<td>egroup</td>
<td>Effective group name of the process owner. If the name is longer than 64, the name is truncated to the length 64.</td>
</tr>
<tr>
<td>__u16</td>
<td>wchan_len</td>
<td>Length of sleeping in function's name and should not be larger than 64.</td>
</tr>
<tr>
<td>char [wchan_len]</td>
<td>wchan_name</td>
<td>Name of sleeping in function or ‘-’. If the name is longer than 64, the name is truncated to the length 64.</td>
</tr>
<tr>
<td>__u16</td>
<td>cmd_len</td>
<td>Length of command name or program name that is used to start the process and should not be larger than 64.</td>
</tr>
<tr>
<td>char [cmd_len]</td>
<td>cmd</td>
<td>Command or program name that is used to start the process. If the name is longer than 64, the name is truncated to the length 64.</td>
</tr>
<tr>
<td>__u16</td>
<td>cmd_line_len</td>
<td>Length of command line that is used to start the process and should not be larger than 1024.</td>
</tr>
<tr>
<td>char [cmd_line_len]</td>
<td>cmd_line</td>
<td>Command line that is used to start the process. If the name is longer than 1024, the name is truncated to the length 1024.</td>
</tr>
</tbody>
</table>

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 pppppppfnvrrmm, where for records that are written by mon_procd, these values are:

pppppppp
  is a fixed ASCII string LNXAPPL.

ff
  is the application number for mon_procd = x'0002'.

n
  is the record number as follows:
  • x'00' indicates summary data.
  • x'01' indicates 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 42, “Reading z/VM monitor records,” on page 451.
osasnmpd – Start OSA-Express SNMP subagent

Use the osasnmpd command to start the OSA-Express Simple Network Management Protocol (SNMP) subagent (osasnmpd).

See Chapter 18, “OSA-Express SNMP subagent support,” on page 311 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 250).
- 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

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
Use the `qethconf` command to configure IP address takeover, virtual IP address (VIPA), and proxy ARP for layer3 qeth devices.

See Chapter 17, “qeth device driver for OSA-Express (QDIO) and HiperSockets,” on page 243 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**

<table>
<thead>
<tr>
<th>Function</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPA</td>
<td><code>ipa</code></td>
</tr>
<tr>
<td>VIPA</td>
<td><code>vipa</code></td>
</tr>
<tr>
<td>Proxy ARP</td>
<td><code>parp</code> or <code>rxip</code></td>
</tr>
</tbody>
</table>

**IPA**: configures qeth for IP address takeover (IPA).

**VIPA**: configures qeth for virtual IP address (VIPA).

**Proxy ARP**: 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.

<intface>
  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
Use the `qethqoat` command to query the OSA address table and display physical and logical device information.

### qethqoat syntax

```
qethqoat
```

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

CHPID: 0x0a9
Manufacturer MAC address: 6c:ae:8b:48:0b:68
Configured MAC address: 00:00:00:00:00:00
Data device sub-channel address: 0xf402
CULA: 0x00
Unit address: 0x02
Physical port number: 0
Number of output queues: 1
Number of input queues: 1
Number of active input queues: 0
CHPID Type: OSD
Interface flags: 0x0a000000
OSA Generation: OSA-Express5S
Port speed/mode: 10 Gb/s / full duplex
Port media type: single mode (LR/LX)
Jumbo frames: yes
Firmware: 0x00000c9a
IPv4 router: no
IPv6 router: no
IPv4 vmac router: no
IPv6 vmac router: no
Connection isolation: not active
Connection isolation VEPA: no
IPv4 assists enabled: 0x00111c77
IPv6 assists enabled: 0x00f15c60
IPv4 outbound checksum enabled: 0x0000003a
IPv6 outbound checksum enabled: 0x00000000
IPv4 inbound checksum enabled: 0x0000003a
IPv6 inbound checksum enabled: 0x00000000
IPv4 Multicast Address: 224.0.0.1
MAC Address: 01:00:5e:00:00:01
IPv6 Address: fe80::6cae:8b00:748:b68
MAC Address: 0x00000000
IPv6 Multicast Address: ff01::1
MAC Address: 33:33:00:00:00:01
ff02:::ff48:b68
MAC Address: 33:33:ff:48:0b:68
ff02::13
MAC Address: 33:33:00:01:00:03

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 0000 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 0010 0000 0000 0000 0000 0000
d3c8 4040 0000 00 0000 0000 0000 0000 0000
0000000 0000 0000 0000 0000 0000 0000 0000 0000
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
c000 0100 5e00 0001 0000 0000 0000 0000 0000
c4c8 f4d4 0000 00 0000 0000 0000 0000 0000
0000080 0000 0000 0000 0000 0000 0000 0000 0000
0000090 0000 0000 0000 0000 0000 0000 0000 0000
c4c8 f6d4 0000 00 0000 0000 0000 0000 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
0000110 0000 0000 0000 0000 0000 0000 0000 0000
0000120 0000 0000 0000 0000 0000 0000 0000 0000
0000130 0000 0000 0000 0000 0000 0000 0000 0000
0000140 ff02 0000 0000 0000 0000 0000 0000 0000
0000150 3333 0000 0000 0000 0000 0000 0000 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.

```bash
#> scsi_logging_level -g
Current scsi logging level:
dev.scsi.logging_level = 0
SCSI_LOG_ERROR=0
SCSI_LOG_TIMEOUT=0
SCSI_LOG_SCAN=0
SCSI_LOG_MLQUEUE=0
SCSI_LOG_MLCOMPLETE=0
SCSI_LOG_LLQUEUE=0
SCSI_LOG_LLCOMPLETE=0
SCSI_LOG_HLQUEUE=0
SCSI_LOG_HLCOMPLETE=0
SCSI_LOG_IOCTL=0
```

• This command sets all logging levels to 3:

```bash
#> scsi_logging_level -s -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.

```bash
#> 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=2
SCSI_LOG_HLQUEUE=3
SCSI_LOG_HLCOMPLETE=2
SCSI_LOG_IOCTL=3
```
# 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_MQUEUE=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

Use the `smc_pnet` command to map a RoCE adapter port to an Ethernet interface.

The SMC-R protocol requires grouping of standard Ethernet and RoCE networks. Such groups are called physical networks (PNETs). Within the same Converged Ethernet fabric, any available Ethernet interface can be combined with an available RDMA-capable network interface card.

**Note:** The mapping of a RoCE adapter port to a standard Ethernet interface can be defined in the IOCDS or it can be defined as an entry in a PNET table. Only use the `smc_pnet` command if the IOCDS does not contain the required PNET IDs. IOCDS specifications override PNET table entries that are created with `smc_pnet`.

```
- a or --add <PNET_ID>
  creates a new entry in the PNET table and allocates the specified ID, if it does not already exist. Only one entry can be defined for a specific Ethernet interface and a specific Infiniband device port. A PNET ID consists of up to 16 alphanumeric uppercase characters without blanks.

- I or --interface <Ethernet_IF>
  specifies the name of the Ethernet interface for a new PNET.

- D or --ibdevice <RoCE_device>
  specifies the name of the RoCE device for a new PNET.

- P or --ibport <RoCE_port>
  Optional: specifies the port number of the RoCE device port. Valid values are 1 or 2. The default value is 1.

- s or --show <PNET_ID>
  displays the PNET table entry with the specified ID.

- d or --delete <PNET_ID>
  deletes the PNET table entry with the specified ID.

- f or --flush
  removes all entries from the PNET table.

- h or --help
  displays help information for the command.

- v or --version
  displays the version number of smc_pnet.
```
Examples

• To create a PNET with ID ABC, Ethernet interface name encf500, Infiniband device name mlx4_0, and port number 2:

  # smc_pnet -a ABC -I encf500 -D mlx4_0 -P 2

• To delete a PNET table entry with PNET ID ABC:

  # smc_pnet -d ABC

• To show the mapping of Ethernet interface name and Infiniband device port for PNET ID ABC:

  # smc_pnet -s ABC
  ABC encf500 mlx4_0 2

• To delete all entries in the PNET table:

  # smc_pnet -f

For command return codes, see the man page.
smc_run - Run a TCP socket program with the SMC protocol using a preloaded library

Use the `smc_run` command to start a TCP socket program that uses SMC as the networking protocol.

```
smc_run <program> <program parameters>
```

Where:

- `smc_run <program> <program parameters>`
  - Starts the specified TCP socket program with the specified parameters, using the SMC protocol.
- `-d`
  - Optional: Display diagnostic messages while the program is running.

**Examples**

- To start a program called iperf3 with parameters "-s -p 12345":
  ```
  # smc_run iperf3 -s -p 12345
  Server listening on 12345
  ...
  ```

- To start a program called iperf3 with parameters "-s -p 12345" and diagnostic messages:
  ```
  # smc_run -d iperf3 -s -p 12345
  ```

For command return codes, see the man page.
smcss - Display information about the AF_SMC sockets and link groups

Use the smcss command to display information about the AF_SMC sockets and link groups.

Entering smcss without any parameters displays a list of connecting, closing, or connected SMC sockets.

- **-a or --all**
  lists all SMC sockets: listening, opening, closing, and connected.

- **-d or --debug**
  displays debug information, such as the shutdown state.

- **-D or --smcd**
  lists SMC-D sockets only. Displays additional SMC-D specific information.

- **-R or --smcr**
  lists SMC-R sockets only. Displays additional SMC-R specific information.

- **-l or --listening**
  lists listening sockets only. These are omitted in the default listing.

- **-w or --wide**
  prevents truncation of IP addresses.

- **-h or --help**
  displays help information for the command.

- **-v or --version**
  displays the version number of smcss.

Meaning of the output fields

<table>
<thead>
<tr>
<th>Entry</th>
<th>Values and meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td></td>
</tr>
<tr>
<td>INIT</td>
<td>The SMC socket is being initialized. It is not connected nor listening yet.</td>
</tr>
<tr>
<td>CLOSED</td>
<td>The SMC socket is closed. It is not connected nor listening anymore.</td>
</tr>
<tr>
<td>LISTEN</td>
<td>The SMC socket is a listening socket, waiting for incoming connection requests.</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>The SMC socket has an established connection. In this state, the TCP connection is fully established, rendezvous processing has been completed, and SMC peers can exchange data via RDMA.</td>
</tr>
<tr>
<td>PEERCLW1</td>
<td>No further data will be sent to the peer.</td>
</tr>
<tr>
<td>PEERCLW2</td>
<td>No further data will be sent to or received from the peer.</td>
</tr>
<tr>
<td>APPLCLW1</td>
<td>No further data will be received from the peer.</td>
</tr>
<tr>
<td>Entry</td>
<td>Values and meaning</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>APPLCLW2</td>
<td>No further data will be received from or sent to the peer.</td>
</tr>
<tr>
<td>APPLFINCLW</td>
<td>The peer has closed the socket.</td>
</tr>
<tr>
<td>PEERFINCLW</td>
<td>The socket is closed locally.</td>
</tr>
<tr>
<td>PEERABORTW</td>
<td>The socket was abnormally closed locally.</td>
</tr>
<tr>
<td>PROCESSABORT</td>
<td>The peer has closed the socket abnormally.</td>
</tr>
<tr>
<td>Inode</td>
<td>denotes the inode of the SMC socket.</td>
</tr>
<tr>
<td>UID</td>
<td>denotes the unique ID of the SMC socket.</td>
</tr>
<tr>
<td>Local Address</td>
<td>denotes address and port number of the local end of the SMC socket. Trailing dots indicate a truncated address. Use the -w option to display full addresses.</td>
</tr>
<tr>
<td>Peer Address</td>
<td>denotes address and port number of the remote end of the socket.</td>
</tr>
<tr>
<td>Intf</td>
<td>denotes that if the socket is explicitly bound with setsockopt option SO_BINDTODEVICE, &quot;Intf&quot; shows the interface number of the Ethernet device to which the socket is bound.</td>
</tr>
<tr>
<td>Mode</td>
<td>can have the following values:</td>
</tr>
<tr>
<td>SMCD</td>
<td>The SMC socket uses SMC-D for data exchange.</td>
</tr>
<tr>
<td>SMCR</td>
<td>The SMC socket uses SMC-R for data exchange.</td>
</tr>
<tr>
<td>TCP</td>
<td>An SMC connection could not be established. The SMC socket uses the TCP protocol for data exchange.</td>
</tr>
<tr>
<td>ShutD</td>
<td>(shutdown) can take the following values:</td>
</tr>
<tr>
<td>&lt;-&gt;</td>
<td>The SMC socket has not been shut down.</td>
</tr>
<tr>
<td>R-&gt;</td>
<td>The SMC socket is shut down one-way and cannot receive data.</td>
</tr>
<tr>
<td>&lt;-W</td>
<td>The SMC socket is shut down one-way and cannot send data.</td>
</tr>
<tr>
<td>R-W</td>
<td>The SMC socket is shut down and cannot receive or send data.</td>
</tr>
<tr>
<td>Token</td>
<td>is a unique ID of the SMC socket connection.</td>
</tr>
<tr>
<td>Sndbuf</td>
<td>denotes the size of the to-be-sent window of the SMC socket connection.</td>
</tr>
<tr>
<td>Rcvbuf</td>
<td>denotes the size of the receiving window of the SMC socket connection (filled by peer).</td>
</tr>
<tr>
<td>Peerbuf</td>
<td>denotes the size of the peer receiving window of the SMC socket connection (to fill during data transfer).</td>
</tr>
<tr>
<td>rxprod-Cursor</td>
<td>Describes the current cursor location of the Rcvbuf for data to be received from the peer.</td>
</tr>
<tr>
<td>Entry</td>
<td>Values and meaning</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rxcons-Cursor</td>
<td>Describes the current cursor location of the Peerbuf for data sent to peer and confirmed by the peer.</td>
</tr>
<tr>
<td>rxFlags</td>
<td>SMC socket connection flags set by and received from the peer.</td>
</tr>
<tr>
<td>txprod-Cursor</td>
<td>Describes the current cursor location of the Peerbuf for data sent to peer.</td>
</tr>
<tr>
<td>txcons-Cursor</td>
<td>Describes the current cursor location of the Rcvbuf for data received from the peer and confirmed to the peer.</td>
</tr>
<tr>
<td>txFlags</td>
<td>SMC socket connection flags set locally and sent to the peer.</td>
</tr>
<tr>
<td>txprep-Cursor</td>
<td>Describes the current cursor location of the Sndbuf for data to be sent. The data is to be moved to the Peerbuf.</td>
</tr>
<tr>
<td>txsent-Cursor</td>
<td>Describes the current cursor location of the Sndbuf for data sent. The data was moved to the Peerbuf.</td>
</tr>
<tr>
<td>txfin-Cursor</td>
<td>Describes the current cursor location of the Sndbuf for data sent and send completion confirmed. The data was moved to the Peerbuf and completion was confirmed.</td>
</tr>
<tr>
<td>Role</td>
<td>&quot;Role&quot; can take the following values:</td>
</tr>
<tr>
<td></td>
<td>CLNT</td>
</tr>
<tr>
<td></td>
<td>The link group of the SMC socket is used for client connections.</td>
</tr>
<tr>
<td></td>
<td>SERV</td>
</tr>
<tr>
<td></td>
<td>The link group of the SMC socket is used for server connections.</td>
</tr>
<tr>
<td>IB-Device</td>
<td>Name of the RoCE device used by the link group to which the SMC socket belongs.</td>
</tr>
<tr>
<td>Port</td>
<td>Port of the RoCE device used by the link group to which the SMC socket belongs.</td>
</tr>
<tr>
<td>Linkid</td>
<td>unique link ID of the link within the link group to which the SMC socket belongs.</td>
</tr>
<tr>
<td>GID</td>
<td>Group identifier of the RoCE port used by the link group to which the SMC socket belongs.</td>
</tr>
<tr>
<td>Peer-GID</td>
<td>GID of the foreign RoCE port used by the link group to which the SMC socket belongs.</td>
</tr>
</tbody>
</table>

**Examples**

- To display information about all SMC sockets on the server:

```bash
[root@myserver]# smcss -a
State   UID   Inode   Local Address           Peer Address            Intf Mode
INIT 000000 0000000
ACTIVE 000000 0060177 ::ffff:10.100.80..:6668 ::ffff:10.100.8..:40812 0000 SMCD
ACTIVE 000000 0060173 ::ffff:10.100.80..:6668 ::ffff:10.100.8..:40804 0000 SMCD
LISTEN 000000 0059058 ::6668
```

- To list listening sockets on the server:

```bash
[root@myserver]# smcss -l
State   UID   Inode   Local Address           Peer Address            Intf Mode
LISTEN 000000 0059058 ::6668
```

- To display debug information about all SMC sockets on the server:

```bash
[root@myserver]# smcss
```

718 Linux on Z and LinuxONE: Device Drivers, Features, and Commands Development stream (Kernel 4.19)
For command return codes, see the man page.
sncap - Manage CPU capacity

Use the simple network 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.

With snipl version 3.0.0, the default behavior changed. SNMPv3 is now used to encrypt communication with LPAR servers (SE or HMC), and OpenSSL for communication with z/VM servers.

**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.
- For CPCs with simultaneous multithreading, sncap acts on entire hardware cores, not at the level of individual threads.
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**

```
-Zaap <number>
  -Ziip <number>
  -Icf <number>
  -Ifl <number>
  -Sap <number>
  -m <ID>
```

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.

- 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 for 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 for the password.

  The password depends on whether the connection is encrypted:

  • When encryption is disabled, the password specifies the password (community) from the SNMP configuration settings on the SE or HMC that controls the CPC you want to work with.

  • When encryption is enabled, the password parameter specifies the password for the SNMPv3 username from the -u command line parameter or user keyword value in the sncap configuration file.

- P or --promptpassword
  prompts for a password (community) in protected entry mode.

- u <username> or --userid <username>
  Specifies the user name from the SNMP configuration settings of an SE or HMC that controls the CPC you want to work with. This parameter is required if encryption is active, and can be specified on the command line or in the sncap configuration file.

- e or --noencryption
  Disables SE or HMC connection encryption. By default, connection encryption is enabled. A user name is not allowed if encryption is disabled.

- 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 cpclid keyword. The cpclid keywords can be added only to the support element HWMCAPI 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.

**user**

(optional, at most once per section) specifies the username from the SNMP settings of the HMC or SE. When omitted, you must specify the user name in the sncap command line arguments. The user parameter applies to encrypted connections only, and is not allowed for unencrypted connections.

**password**

(optional, at most once per section) specifies the password for 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 is equivalent to the `--password` command-line argument. The password depends on whether the connection is encrypted:

- When encryption is disabled, the password specifies the password (community) from the SNMP settings of the SE or HMC.
- When encryption is enabled, the password is the SNMPv3 user name password from the SNMPv3 settings of the SE or HMC.

**encryption**

(optional, at most once per section) specifies whether the server connection is encrypted. Valid values are: yes or no. If not specified, encryption is enabled by default. Specifying "encryption = no" is equivalent to the `-e` 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
The following sample summarizes how to specify an encrypted connection to an SE and an unencrypted connection to an HMC. Required specifications are shown in bold.

```
# Comment line (ignored).
#
# A section that defines a support element connection.
#
Server = 192.0.2.4
type = LPAR

encryption = yes
user = hugo
cpcid = SZ01CP00
password = pw42play
#
# A section that defines a hardware management console
# connection.
#
Server = 192.0.2.2
encryption = no
type = LPAR
cpcid = SZ02CP00
  cpcid = SZ02CP01
  cpcid = SZ02CP03
  cpcid = SZ02CP04
password= pw42play
```

Examples
• To activate a CBU temporary capacity record CB7KHB38 on CPC SZ02CP03 to temporarily upgrade it to model capacity identifier 741:

  `sncap SZ02CP03 -S 192.0.2.2 -e -P -a CB7KHB38 -m 741`

• To activate only a subset of processors defined in temporary capacity record CB7KHB38 on the CPC SZ02CP03:

  `sncap SZ02CP03 -S 192.0.2.2 -e -P -a CB7KHB38 --zaap 2 --ziip 2`

• To deactivate a CBU temporary capacity record CB7KHB38 on the CPC SZ01CP00:

  `sncap SZ01CP00 -S 192.0.2.4 -u hugo -P -d CB7KHB38`

• To deactivate only a subset of processors defined in temporary capacity record CB7KHB38 on the CPC SZ01CP00:

  `sncap SZ01CP00 -S 192.0.2.4 -u hugo -P -d CB7KHB38 --zaap 2 --ziip 2`

With a suitable configuration file at /etc/xcfg the previous command can be shortened to:

  `sncap SZ01CP00 -f /etc/xcfg -d CB7KHB38 --zaap 2 --ziip 2`

With a suitable default configuration file the command can be further shortened to:

  `sncap SZ01CP00 -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 z Systems Capacity on Demand User's Guide, SC28-6943 or any updates of this publication that applies to your mainframe system.
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 15, “Channel-attached tape device driver,” on page 225.

```
tape390_crypt syntax
```

```
tape390_crypt -q
-tape390_crypt -e
-tape390_crypt -k
-tape390_crypt -d

Keys

-k <value> [label]
-k <value> [hash]
-d :d [char]
-d <char>

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.

<n>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
ENCRIPTION: OFF
MEDIUM: NOT ENCRYPTED
```

```
tape390_crypt -e on /dev/ntibm0
```

```
tape390_crypt -q /dev/ntibm0
ENCRIPTION: ON
MEDIUM: NOT ENCRYPTED
```

Then, two keys are set, one in label format and one in hash format. The status is queried and there is now additional output for the keys.
Using default keys for encryption

1. Load the cartridge. If the cartridge is already loaded:
   - Switch off encryption:
     ```bash
tape390_crypt -e off /dev/ntibm0
     ```
   - Rewind:
     ```bash
     mt -f /dev/ntibm0 rewind
     ```

2. Switch encryption on:
   ```bash
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:
   ```bash
   mt -f /dev/ntibm0 rewind
   ```

2. Switch encryption on:
   ```bash
   tape390_crypt -e on /dev/ntibm0
   ```

3. Set new keys:
   ```bash
   tape390_crypt -k key1 -k key2 /dev/ntibm0
   ```

4. Write data.

Writing unencrypted data

1. Load the cartridge. If the cartridge is already loaded, rewind:
   ```bash
   mt -f /dev/ntibm0 rewind
   ```

2. If encryption is on, switch off encryption:
   ```bash
   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**

```
<node>  
```

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 154)

**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 146).
- Data gathering must be turned on by writing "on" to /proc/dasd/statistics.

**tunedasd syntax**

```bash
tunedasd -g
-c <mode>
-n <cylinders>
-Q
-S
-L
-O
-R
-P
-I <row>
-p <chpid>
--path_reset_all
```

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.

**-p** or **--path_reset** `<chpid>`
- resets a channel path `<chpid>` of a selected device. A channel path might be suspended due to high IFCC error rates or a High Performance FICON failure. Use this option to resume considering the channel path for I/O.

**--path_reset_all**
- resets all channel paths of a selected device. The channel paths might be suspended due to high IFCC error rates or a High Performance FICON failure. Use this option to resume considering all defined channel paths for I/O.

**-v** or **--version**
- displays version information.

**-h** or **--help**
- displays help information. 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:

  ```bash
  # 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".

  ```bash
  # tunedasd -g /dev/dasdzzz
  normal (0 cyl)
  # tunedasd -c prestage -n 2 /dev/dasdzzz
  Setting cache mode for device </dev/dasdzzz>...
  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.

  ```bash
  # tunedasd -g /dev/dasdzzz /dev/dasdzzy
  prestage (2 cyl)
  normal (0 cyl)
  ```

- The following command displays the usage profile of a DASD.
- 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 /dev/dasdzzz
19617 dasd I/O requests
with 4841336 sectors (512B each)

<table>
<thead>
<tr>
<th>&lt;4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1k</th>
<th>2k</th>
<th>4k</th>
<th>8k</th>
<th>16k</th>
<th>32k</th>
<th>64k</th>
<th>128k</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>441</td>
<td>77</td>
<td>78</td>
<td>87</td>
<td>188</td>
<td>18746</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Histogram of sizes (512B secs)

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

Histogram of I/O times (microseconds)

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

Histogram of I/O times per sector

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

Histogram of I/O times till ssch

19234 | 40 | 32 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Histogram of I/O time between ssch and irq

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

Histogram of I/O time between ssch and irq per sector

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

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   |

# of req in chan at enqueuing (1..32)

0  | 19308| 123| 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   | 0   |

- The following command resets a failed channel path with CHPID 45:

```
# tunedasd -p 45 /dev/dasdc
```
vmcp - Send CP commands to the z/VM hypervisor

Use the vmcp command to send control program (CP) commands to the z/VM hypervisor and display the response from z/VM.

The vmcp command expects the command line as a parameter and returns the response to stdout. Error messages are written to stderr.

You can issue CP commands through the /dev/vmcp device node (see Chapter 46, “z/VM CP interface device driver,” on page 479) or with the vmcp command. In both cases, the vmcp module must be compiled into the kernel.

vmcp syntax

```
  vmcp [-k] [-b <size>] <command>
```

Where:

- **-k or --keepcase**
  preserves the case of the characters in the specified command string. By default, the command string is converted to uppercase characters.

- **-b <size> or --buffer=<size>**
  specifies the buffer size in bytes for the response from z/VM CP. Valid values are from 4096 (or 4k) up to 1048756 (or 1M). By default, vmcp allocates an 8192 byte (8k) buffer. You can use k and M to specify kilo- and megabytes. The suffixes are not case sensitive, so k is equivalent to K and m is equivalent to M.

- **<command>**
  specifies the command that you want to send to CP.

- **-h or --help**
  displays help information. 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:
every user on the system can run the `indicate` command by using:

```bash
# sudo vmcp indicate
```

- If you need a larger response buffer, use the `--buffer` option:

```bash
# vmcp --buffer=128k q 1-ffff
```
Use the `vmur` command to work with z/VM spool file queues. The `vmur` command provides these main functions:

**Receive**
Read data from the z/VM reader file queue. The command performs the following steps:
- Places the reader queue file to be received at the top of the queue.
- Changes the reader queue file attribute to NOHOLD.
- Closes the z/VM reader after the file is received.

The `vmur` command detects z/VM reader queue files in:
- VMDUMP format as created by CP VMDUMP.
- NETDATA format as created by CMS SENDFILE or TSO XMIT.

**Punch or print**
Write data to the z/VM punch or printer file queue and transfer it to another user's virtual reader, optionally on a remote z/VM node. The data is sliced up into 80-byte or 132-byte chunks (called records) and written to the punch or printer device. If the data length is not an integer multiple of 80 or 132, the last record is padded.

**List**
Display detailed information about one or all files on the specified spool file queue.

**Purge**
Remove one or all files on a spool file queue.

**Order**
Position a file at the top of a spool file queue.

**Before you begin:** To use the receive, punch, and print functions, the `vmur` device driver must be loaded and the corresponding unit record devices must be set online.

**Serialization**
The `vmur` command provides strict serialization of all its functions other than list, which does not affect a file queue's contents or sequence. Thus concurrent access to spool file queues is blocked to prevent unpredictable results or destructive conflicts.

For example, this serialization prevents a process from issuing `vmur purge -f` while another process is running `vmur receive 1234`. However, `vmur` is not serialized against concurrent CP commands that are issued through `vmcp`: if one process is running `vmur receive 1234` and another process issues `vmcp purge rdr 1234`, then the received file might be incomplete. To avoid such unwanted effects, always use `vmur` to work with z/VM spool file queues.

**Spooling options**
With the `vmur` command, you can temporarily override the z/VM settings for the CLASS, DEST, FORM, and DIST spooling options for virtual unit record devices. The `vmur` command restores the original settings before it returns control.

For details about the spooling options, see the z/VM product information. In particular, see the sections about the z/VM CP SPOOL, QUERY VIRTUAL RDR, QUERY VIRTUAL PUN, and QUERY VIRTUAL PRT commands in z/VM: CP Commands and Utilities Reference, SC24-6268.
vmur syntax

Where these are the main command options:

Receive options

Punch and print options

Where these are the main command options:
**vmur**

**Re or receive**
receives a file from the z/VM reader queue.

**Pun or punch**
writes to the z/VM punch queue.

**Pr or print**
writes to the z/VM printer queue.

**Li or list**
lists information about one or all files on a z/VM spool file queue.

**Pur or purge**
purges one or all files from a z/VM spool file queue.

**Or or order**
places a file on a z/VM spool file queue at the top of the queue.

**Note:** The short forms that are given for receive, punch, print, list, purge, and order are the shortest possible abbreviations. In keeping with z/VM style, you can abbreviate commands by dropping any number of letters from the end of the full keywords until you reach the short form. For example, `vmur re`, `vmur rec`, or `vmur rece` are all equivalent.

The remaining specifications are listed alphabetically by switch. Variable specifications that do not require a switch are listed first.

**<file>**
specifies a file, in the Linux file system, with data to be punched or printed. If this specification is omitted, the data is read from standard input.

**<out file>**
specifies a file, in the Linux file system, to receive data from the reader spool file. If neither a file name nor `--stdout` are specified, the name and type of the spool file to be received (see the NAME and TYPE columns in `vmur list` output) are used to build an output file name of the form `<name>.<type>`. If the spool file to be received is an unnamed file, an error message is issued.

Use the `--force` option to overwrite existing files without a confirmation prompt.

**<spoolid>**
specifies the spool ID of a file on the z/VM reader, punch, or printer queue. Spool IDs are decimal numbers in the range 0-9999.

For the list or purge function: omitting the spool ID lists or purges all files in the queue.

**-b <sep>,<pad>**
receives or writes a file in blocked mode, where `<sep>` specifies the separator and `<pad>` specifies the padding character in hexadecimal notation. Example: `<sep>

```
  -b Blocked 0x55,0x40
```

Use this option to use character sets other than IBM037 and ISO-8859-1 for conversion.

- For the receive function: All trailing padding characters are removed from the end of each record that is read from the virtual reader and the separator character is inserted afterward. The receive function's output can be piped to `iconv` by using the appropriate character sets. Example:

```
  # vmur rec 7 -b 0x25,0x40 -O | iconv -f EBCDIC-US -t ISO-8859-1 > myfile
```

- For the punch or print function: The separator is used to identify the line end character of the file to punch or print. If a line has fewer characters than the record length of the used unit record device, the residual of the record is filled up with the specified padding byte. If a line exceeds the record size, an error is printed. Example:

```
  # iconv test.txt -f ISO-8859-1 -t EBCDIC-US | vmur pun -b 0x25,0x40 -N test
```
-c or --convert
converts a VMDUMP spool file into a format appropriate for further analysis with crash.

-C <class> or --class <class>
specifies a spool class.

  • For the receive function: The file is received only if it matches the specified class.
  • For the purge function: Only files with the specified class are purged.
  • For the punch or printer function: Sets the spool class for the virtual reader or virtual punch device.
    Output files inherit the spool class of the device.

  The class is designated by a single alphanumeric character. For receive, it can also be an asterisk (*) to match all classes. Lowercase alphabetic characters are converted to uppercase.

  See also “Spooling options” on page 739.

--dest <device>
sets the destination device for spool files that are created on the virtual punch or printer device. The value can be ANY, OFF, or it must be a valid device as defined on z/VM.

  See also “Spooling options” on page 739.

-d or --device
specifies the device node of the virtual unit record device.

  • If omitted in the receive function, /dev/vmrdr-0.0.000c is assumed.
  • If omitted in the punch function, /dev/vmpun-0.0.000d is assumed.
  • If omitted in the print function, /dev/vmprt-0.0.000e is assumed.

--dist <distcode>
sets the distribution code for spool files that are created on the virtual punch or printer device. The value can be an asterisk (*), OFF, or it must be a valid distribution code as defined on z/VM.

  OFF and * are equivalent. Both specifications reset the distribution code to the value that is set in the user directory.

  See also “Spooling options” on page 739.

-f or --force
suppresses confirmation messages.

  • For the receive function: overwrites an existing output file without prompting for a confirmation.
  • For the punch or print option: automatically converts the Linux input file name to a valid spool file name without any error message.
  • For the purge function: purges the specified spool files without prompting for a confirmation.

--form <form_name>
sets the form name for spool files that are created on the virtual punch or printer device. The value can be OFF, to use the system default, or it must be a valid z/VM form name.

  See also “Spooling options” on page 739.

-h or --help
displays help information for the command. To view the man page, enter man vmur.

-H or --hold
keeps the spool file to be received in the reader queue. If omitted, the spool file is purged after it is received.

-n <node> or --node <node>
specifies the node name of the z/VM system to which the data is to be transferred. Remote Spooling Communications Subsystem (RSCS) must be installed on the z/VM systems and the specified node must be defined in the RSCS machine’s configuration file.

  The default node is the local z/VM system. The node option is valid only with the -u option.
-N <name><type> or --name <name><type>
specifies a name and, optionally, a type for the z/VM spool file to be created by the punch or print option. To specify a type after the file name, enter a period followed by the type. For example:

```
vmur pun -r /boot/parmfile -N myname.mytype
```

Both the name and the type must comply with z/VM file name rules, for example, they must be 1 - 8 characters long.

If omitted, a spool file name is generated from the Linux input file name, if applicable.

Use the --force option to suppress warning messages about automatically generated file names or about specified file names that do not adhere to the z/VM file naming rules.

-O or --stdout
writes the reader file content to standard output.

-q or --queue
specifies the z/VM spool file queue to be listed, purged, or ordered. If omitted, the reader file queue is assumed.

-r or --rdr
transfers a punch or print file to a reader.

-t or --text
converts the encoding between EBCDIC and ASCII according to character sets IBM037 and ISO-8859-1.

• For the receive function: receives the reader file as text file. That is, it converts EBCDIC to ASCII and inserts an ASCII line feed character (0x0a) for each input record that is read from the z/VM reader. Trailing EBCDIC blanks (0x40) in the input records are stripped.

• For the punch or print function: punches or prints the input file as text file. That is, converts ASCII to EBCDIC and pads each input line with trailing blanks to fill up the record. The record length is 80 for a punch and 132 for a printer. If an input line length exceeds 80 for punch or 132 for print, an error message is issued.

The --text and the --blocked attributes are mutually exclusive.

-u <user> or --user <user>
specifies the z/VM user ID to whose reader the data is to be transferred. If omitted, the data is transferred to your own machine’s reader. The user option is valid only with the -r option.

-v or --version
displays version information.

Examples
These examples illustrate common scenarios for unit record devices.

In all examples the following device nodes are used:

• /dev/vmrd\r-0.0.000c as virtual reader.
• /dev/vmpun-0.0.000d as virtual punch.

The vmur commands access the reader device, which has to be online. To set it online, it needs to be freed from cio_ignore. Example:

```
# cio_ignore -r c
# chccwdev -e c
Setting device 0.0.000c online
Done
```

Besides the vmur device driver and the vmur command, these scenarios require that:

• The vmcp 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
   ```

<table>
<thead>
<tr>
<th>ORIGINID</th>
<th>FILE</th>
<th>CLASS</th>
<th>RECORDS</th>
<th>CPY</th>
<th>HOLD</th>
<th>DATE</th>
<th>TIME</th>
<th>NAME</th>
<th>TYPE</th>
<th>DIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>T6360025</td>
<td>0463</td>
<td>V DMP</td>
<td>00020222</td>
<td>001</td>
<td>NONE</td>
<td>06/11</td>
<td>15:07:42</td>
<td>VMDUMP</td>
<td>FILE</td>
<td>T6360025</td>
</tr>
</tbody>
</table>
   ```

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 rec <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 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:
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 000065200 001 NONE 06/11 15:30:52 image T6360025
   ```

4. Move image to the first and parmfile to the second position in the reader queue:

   ```
   # vmur or 465
   # vmur or 466
   ```

5. Configure the z/VM reader as the re-IPL device:

   ```
   # chreipl ccw 0.0.000c
   ```
   For details about the chreipl command, see "chreipl - Modify the re-IPL configuration" on page 581. To pass additional options, use the -p option.

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 `lnxpathile.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.

   vmuc pun lnxprofile.exec -t -r -u t2930020 -n boet2930 -N PROFILE
   vmuc pun lnxprofile.exec -t -r -u t6360025 -n boet6360 -N PROFILE

2. Log on to t2930020 at boet2930, IPL CMS, and issue the CP command:

   QUERY RDR ALL

   The command output shows the spool ID of PROFILE in the FILE column.

3. Issue the CMS command:

   RECEIVE <spoolid> PROFILE EXEC A (REPL

   In the command, `<spoolid>` is the spool ID of PROFILE found in step “2” on page 746.

4. Repeat steps “2” on page 746 and “3” on page 746 for t6360025 at boet6360.

Sending a file to a z/VSE instance
You can use the `vmuc` command to send files to a z/VSE instance.

Procedure

To send `lserv.job` to user ID vseuser at node vse01sys, issue:

vmuc pun lserv.job -t -r -u vseuser -n vse01sys -N LSERV
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 165 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 <zdsfs-options> <fuse-options> -l <file-name> <node-list> <mount-point>
```
where:

**<zdfs-options>**

zdfs-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 zdfs 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.
  zdfs 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.
  zdfs saves offset information about a data set in the seek history buffer to speed up the performance of a seek operation.

- **-o check_host_count**
  checks the host-access open count to ensure that the device is not online to another operating system instance. The operation is canceled if another operating system instance is accessing the volume.

**<fuse-options>**

options for FUSE. The following options are supported by the zdfs 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.

<nodelist>
  one or more device nodes for the DASDs, separated by blanks.

<filename>
  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 <filename> 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>(<filename>) 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:

<table>
<thead>
<tr>
<th>dsn</th>
<th>recfm</th>
<th>lrecl</th>
<th>dsorg</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOOBAR.TESTF.TXT</td>
<td>FB</td>
<td>80</td>
<td>PS</td>
</tr>
<tr>
<td>FOOBAR.TESTVF.TXT</td>
<td>FB</td>
<td>100</td>
<td>PS</td>
</tr>
<tr>
<td>FOOBAR.PDSF.DAT</td>
<td>F</td>
<td>80</td>
<td>PO</td>
</tr>
<tr>
<td>FOOBAR.PDSF.DAT(TEST1)</td>
<td>F</td>
<td>80</td>
<td>PS</td>
</tr>
<tr>
<td>FOOBAR.PDSF.DAT(TEST2)</td>
<td>F</td>
<td>80</td>
<td>PS</td>
</tr>
<tr>
<td>FOOBAR.PDSF.DAT(TEXT3)</td>
<td>F</td>
<td>80</td>
<td>PS</td>
</tr>
</tbody>
</table>

- 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
    drwxr-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, Dec 13 3 15:54 /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 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, Dec 13 3 15:54 /dev/dasde
$ ls -al /mnt
  total 121284
  dr-xr-x---  2 myuser zosimpoz  0 Dec  3 14:22  .
drwxr-xr-x 23 root root 4096 Dec  3 13:59  ..
-r--r-----  1 myuser zosimpoz 981 Dec  3 14:22  metadata.txt
-r--r-----  1 myuser zosimpoz 2833200 Jun 27  2012  EXPORT.BIN1.DAT
-r--r-----  1 myuser zosimpoz 2833200 Jun 27  2012  EXPORT.BIN2.DAT
-r--r-----  1 myuser zosimpoz 2833200 Feb 14  2013  EXPORT.BIN3.DAT
-r--r-----  1 myuser zosimpoz 2833200 Jun 27  2012  EXPORT.BIN4.DAT
dr-xr-x---  2 myuser zosimpoz 13599360 Aug  9  2012  EXPORT.PDS1.DAT
dr-xr-x---  2 myuser zosimpoz 13599360 Aug  9  2012  EXPORT.PDS2.DAT
dr-xr-x---  2 myuser zosimpoz 55247400 Aug  9  2012  EXPORT.PDS3.DAT
dr-xr-x---  2 myuser zosimpoz 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
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, ctcn, 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

```
znetconf [-a | --add] <device_bus_id>
znetconf -A <device_bus_id>
znetconf -e <device_bus_id>
znetconf -o <attribute> = <value>
znetconf -d <driver>
znetconf -r <device_bus_id>
znetconf -R <device_bus_id>
znetconf -u
```

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 portno option with the value 1:

```
znetconf -a f500 -o layer2=0 -o portno=1
```
To list configured network devices:

```
znetconf -c
```

<table>
<thead>
<tr>
<th>Device IDs</th>
<th>Type</th>
<th>Card Type</th>
<th>CHPID</th>
<th>Drv. Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.f500,0.0.f501,0.0.f502</td>
<td>1731/01</td>
<td>Virt.NIC QDIO</td>
<td>00</td>
<td>qeth eth2</td>
<td>online</td>
</tr>
<tr>
<td>0.0.f503,0.0.f504,0.0.f505</td>
<td>1731/01</td>
<td>Virt.NIC QDIO</td>
<td>01</td>
<td>qeth eth1</td>
<td>online</td>
</tr>
<tr>
<td>0.0.f5f0,0.0.f5f1,0.0.f5f2</td>
<td>1731/01</td>
<td>OSD_1000</td>
<td>76</td>
<td>qeth eth0</td>
<td>online</td>
</tr>
</tbody>
</table>

To remove network device 0.0.f503:

```
znetconf -r 0.0.f503
```

or

```
znetconf -r f503
```

To remove all configured network devices except the devices with bus IDs 0.0.f500 and 0.0.f5f0:

```
znetconf -R -e 0.0.f500 -e 0.0.f5f0
```

To configure all potential network devices except the device with bus ID 0.0.f503:

```
znetconf -A -e 0.0.f503
```
zpcictl - Manage defective PCIe devices

Use the zpcictl to manage PCI devices on the IBM Z platform. In particular, use this command to report defective PCI devices to the support element (SE).

zpcictl syntax

```
  zpcictl [options] <function_address>
```

Where:
- `--reset` resets the device.
- `--deconfigure` deconfigures the device to prepare for any repair action.
- `--report-error` reports the device error to the SE.

`<function_address>`
the PCI function address of the device, for example, 0000:00:00.0.

- `-h` or `--help`
  displays help information for the command. To view the man page, enter `man zpcictl`.

- `-v` or `--version`
  displays version information.

Example

In this example, an error is reported for a PCI device with slot ID 0000:00:00.0.

```
  # zpcictl --report-error 0000:00:00.0
```
Chapter 66. 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 4, "Kernel and module parameters," on page 23 for information about specifying kernel parameters.
cio_ignore - List devices to be ignored

Use the cio_ignore= kernel parameter to list specifications for I/O devices that are to be ignored.

When an instance of Linux on Z boots, it senses and analyzes all available I/O devices. You can use the cio_ignore= kernel parameter to list specifications for devices that are to be ignored. 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 759.

Format

```
cio_ignore=all,<device_spec>,!<device_spec>,<device_bus_id>,<from_device_bus_id>-<to_device_bus_id>,ipldev,condev
```

![Figure 193: cio_ignore syntax](image)

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.

  \texttt{cio\_ignore=0.0.b100-0.0.b1ff,0.0.a100}

- This example specifies that all devices except the console are to be ignored.

  \texttt{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.

  \texttt{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.

  \texttt{cio\_ignore=0.0.1000-0.0.1500,0.0.1100-0.0.1120}

  This is equivalent to the following specification:

  \texttt{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.

  \texttt{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 \texttt{cio\_ignore} command or the procfs interface to view or change the list of I/O device specifications that are ignored.

When an instance of Linux on Z boots, it senses and analyzes all available I/O devices. You can use the \texttt{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 \texttt{cio\_ignore} command (see "\texttt{cio\_ignore - Manage the I/O exclusion list}" on page 596). This information describes the procfs interface.

After booting Linux you can display the exclusion list by issuing:

\texttt{# cat /proc/cio\_ignore}

To add device specifications to the exclusion list issue a command of this form:

\texttt{# 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 \texttt{lscs\_ss} 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:

\texttt{# 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:
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:

```
<device_list>
```

Where the keywords and variables have the same meaning as in “Format” on page 758.

**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 596) 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.

```bash
# 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.

```bash
# echo free 0.0.c104 > /proc/cio_ignore
```

• This command adds the device with bus ID 0.0.c104 to the exclusion list.

```bash
# 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.

```bash
# 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**

```
Format

```

**Examples**

This specification disables the CMMA support:

```
cmma=off
```

Alternatively, you can use the following specification to disable the CMMA support:

```
cmma=no
```
**maxcpus - Limit the number of CPUs Linux can use at IPL**

Use the `maxcpus` kernel parameter to limit the number of CPUs that Linux can use at IPL and that are online after IPL.

If the real or virtual hardware provides more than the specified number of CPUs, these surplus CPUs are initially offline. For example, if five CPUs are available, `maxcpus=2` results in two online CPUs and three offline CPUs after IPL.

Offline CPUs can be set online dynamically unless the `possible_cpus` parameter is set and specifies a maximum number of online CPUs that is already reached. The `possible_cpus` parameter sets an absolute limit for the number of CPUs that can be online at any one time (see `possible_cpus`). If both `maxcpus` and `possible_cpus` are set, a lower value for `possible_cpus` overrides `maxcpus` and makes it ineffective.

**Format**

```
maxcpus = <number>
```

*Figure 195: maxcpus syntax*

**Examples**

```
maxcpus=2
```
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**

```plaintext
noinitrd
```

*Figure 196: noinitrd syntax*
nosmt - Disable simultaneous multithreading

By default, Linux in LPAR mode uses simultaneous multithreading if it is supported by the hardware. Specify the nosmt kernel parameter to disable simultaneous multithreading. See also “smt - Reduce the number of threads per core” on page 770.

For more information about simultaneous multithreading, see “Simultaneous multithreading” on page 361.

**Format**

```
 nosmt
```

*Figure 197: nosmt syntax*
possible_cpus - Limit the number of CPUs Linux can use

Use the possible_cpus= parameter to set an absolute limit for the number of CPUs that can be online at any one time. If the real or virtual hardware provides more than the specified maximum, the surplus number of CPUs must be offline. Alternatively, you can use the common code kernel parameter nr_cpus.

Use the maxcpus= parameter to limit the number of CPUs that are online initially after IPL (see maxcpus).

Format

```
possible_cpus = <number>
```

*Figure 198: possible_cpus syntax*

Examples

```
possible_cpus=8
```
ramdisk_size - Specify the ramdisk size

Use the `ramdisk_size=` kernel parameter to specify the size of the ramdisk in kilobytes.

**Format**

```
ramdisk_size = <size>
```

*Figure 199: ramdisk_size syntax*

**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
```

*Figure 200: ro syntax*
root - Specify the root device

Use the `root=` kernel parameter to tell Linux what to use as the root when mounting the root file system.

**Format**

```
root = <rootdevice>
```

*Figure 201: root syntax*

**Examples**

This example makes Linux use `/dev/dasda1` when mounting the root file system:

```
root=/dev/dasda1
```
smt - Reduce the number of threads per core

By default, Linux in LPAR mode uses the maximum number of threads per core that is supported by the hardware. Use the `smt=` kernel parameter to use fewer threads. The value can be any integer in the range 1 to the maximum number of threads that is supported by the hardware.

Specifying `smt=1` effectively disables simultaneous multithreading. See also “nosmt - Disable simultaneous multithreading” on page 765.

For more information about simultaneous multithreading, see “Simultaneous multithreading” on page 361.

**Format**

```
smt = <hwmax>
smt = <number>
```

*Figure 202: smt syntax*

where `<hwmax>` is the maximum number of threads per core that is supported by the hardware, and `<number>` is an integer in the range 1 - `<hwmax>`.

**Examples**

```
smt=1
```
**vdso - Optimize system call performance**

Use the `vdso=` kernel parameter to control the vdso support for the `gettimeofday`, `clock_getres`, and `clock_gettime` system calls.

The virtual dynamic shared object (vdso) support is a shared library that the kernel maps to all dynamically linked programs. The glibc detects the presence of the vdso and uses the functions that are provided in the library.

Because the vdso library is mapped to all user-space processes, this change is visible in user space. In the unlikely event that a user-space program does not work with the vdso support, you can disable the support.

The default, which is to use vdso support, works well for most installations. Do not override this default, unless you observe problems.

The vdso support is included in the Linux kernel.

**Format**

```
vdso = on
```

```
vdso = off
```

*Figure 203: vdso syntax*

**Examples**

This example disables the vdso support:

```
vdao=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 = <COMMAND>
```

*Figure 204: vmhalt syntax*

**Examples**

This example specifies that an initial program load of CMS is to follow the Linux `halt` command:

```
vmhalt="CPU 00 CMD I CMS"
```

**Note:** The command must be entered in uppercase.
vmpanic - Specify CP command to run after a kernel panic

Use the vmpanic= kernel parameter to specify a command to be issued to CP after a kernel panic.

This command applies only to Linux on z/VM.

**Note:** Ensure that the dumpconf service is disabled when you use this kernel parameter. Otherwise, dumpconf will override the setting.

**Format**

```
vmpanic = <COMMAND>
```

*Figure 205: vmpanic syntax*

**Examples**

This example specifies that a VMDUMP is to follow a kernel panic:

```
vmpanic="VMDUMP"
```

**Note:** The command must be entered in uppercase.
**vmpoff** - Specify CP command to run after a power off

Use the `vmpoff=` kernel parameter to specify a command to be issued to CP after a system power off. This command applies only to Linux on z/VM.

**Format**

```
vmpoff = <COMMAND>
```

*Figure 206: vmpoff syntax*

**Examples**

This example specifies that CP is to clear the guest virtual machine after the Linux power off or `halt -p` command:

```
vmpoff="SYSTEM CLEAR"
```

**Note:** The command must be entered in uppercase.
**vmreboot - Specify CP command to run on reboot**

Use the `vmreboot=` kernel parameter to specify a command to be issued to CP on reboot. This command applies only to Linux on z/VM.

**Format**

```
vmreboot = <COMMAND>
```

*Figure 207: vmreboot syntax*

**Examples**

This example specifies a message to be sent to the z/VM guest virtual machine OPERATOR if a reboot occurs:

```
vmreboot="MSG OPERATOR Reboot system"
```

**Note:** The command must be entered in uppercase.
vmreboot
Chapter 67. Linux diagnose code use

Linux on Z issues several diagnose instructions to the hypervisor (LPAR or z/VM).

Table 77 on page 777 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>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Linux use</th>
<th>Required/ Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x008</td>
<td>z/VM CP command console interface</td>
<td>• The <code>vmcp</code> command</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The 3215 and 3270 console drivers</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The z/VM recording device driver (vmilogrd)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>smsgiucv</code></td>
<td></td>
</tr>
<tr>
<td>0x010</td>
<td>Release pages</td>
<td>CMM</td>
<td>Required</td>
</tr>
<tr>
<td>0x014</td>
<td>Input spool file manipulation</td>
<td>The <code>vmur</code> device driver</td>
<td>Required</td>
</tr>
<tr>
<td>0x044</td>
<td>Voluntary time-slice end</td>
<td>In the kernel for spinlock and <code>udelay</code></td>
<td>Required</td>
</tr>
<tr>
<td>0x064</td>
<td>Allows Linux to attach a DCSS</td>
<td>The DCSS block device driver (<code>dcssblk</code>), <code>xip</code>, and the MONITOR record device driver (<code>monreader</code>).</td>
<td>Required</td>
</tr>
<tr>
<td>0x09c</td>
<td>Voluntary time slice yield</td>
<td>Spinlock.</td>
<td>Optional</td>
</tr>
<tr>
<td>0xc</td>
<td>z/VM management time data</td>
<td>The <code>hyptop</code> command</td>
<td>Optional</td>
</tr>
<tr>
<td>0x0dc</td>
<td>Monitor stream</td>
<td>The <code>APPLDATA</code> monitor record and the MONITOR stream application support (<code>monwriter</code>).</td>
<td>Required</td>
</tr>
<tr>
<td>0x204</td>
<td>LPAR Hypervisor data</td>
<td>• The hypervisor file system (<code>hypfs</code>).</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The <code>hyptop</code> command</td>
<td></td>
</tr>
<tr>
<td>0x210</td>
<td>Retrieve device information</td>
<td>• The common I/O layer</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The DASD driver DIAG access method</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DASD read-only query</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The <code>vmur</code> device driver</td>
<td></td>
</tr>
<tr>
<td>0x224</td>
<td>CPU type name table</td>
<td>The hypervisor file system (<code>hypfs</code>).</td>
<td>Required</td>
</tr>
<tr>
<td>0x250</td>
<td>Block I/O</td>
<td>The DASD driver DIAG access method.</td>
<td>Required</td>
</tr>
<tr>
<td>0x258</td>
<td>Page-reference services</td>
<td>In the kernel, for <code>pfault</code>.</td>
<td>Optional</td>
</tr>
<tr>
<td>0x26c</td>
<td>z/VM: Access Certain System Information</td>
<td>The <code>qeth</code> device driver, for MAC address</td>
<td>Optional</td>
</tr>
<tr>
<td>0x288</td>
<td>Virtual machine time bomb</td>
<td>The watchdog device driver</td>
<td>Required</td>
</tr>
</tbody>
</table>
### Table 77: Linux diagnoses (continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Linux use</th>
<th>Required/Optional</th>
</tr>
</thead>
</table>
| 0x2fc  | Hypervisor cpu and memory accounting data | • The hypervisor file system (hypfs).  
                   • The hypstop command.            | Required          |
| 0x308  | Re-ipl                                   | Re-ipl and dump code.                   | Required          |
| 0x500  | Virtio functions                         | Operate virtio-ccw devices               | 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 68. Kernel configuration menu options

There are numerous kernel configuration options that are specific to IBM 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 780
- “Device driver-related options” on page 791

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 208 on page 780 and Figure 210 on page 791). 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 208 on page 780 and Figure 210 on page 791). 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, go to the root of the Linux source tree and 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 z/Architecture specific options that are not in the Device Drivers main menu.

Figure 208 on page 780 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 210 on page 791.

... Processor type and features --->
  Processor type (choice)
  • 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_ZEC12)
  • IBM z13 and z13 (CONFIG_MARCH_Z13)
  • IBM z14 ZR1 and z14 (CONFIG_MARCH_Z14)

Tune code generation (choice)
  • Default (CONFIG_TUNE_DEFAULT)
  • 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_ZEC12)
  • IBM z12 and zEC12 (CONFIG_TUNE_Z13)
  • IBM z13 (CONFIG_TUNE_Z14)

Kernel support for 31 bit emulation (CONFIG_COMPAT)

Symmetric multi-processing support (CONFIG_SMP)
  Maximum number of CPUs (2-512) (CONFIG_NR_CPUS)
  Support for hot-pluggable CPUs
  NUMA support (common code option CONFIG_NUMA)
    Select NUMA modes (depends on NUMA) --->
      NUMA emulation (CONFIG_NUMA_EMU)
      NUMA emulation memory chunk size (CONFIG_EMU_SIZE)
    Topology scheduler support (CONFIG_SCHED_TOPOLOGY)

... kexec file based system call
  s390 architectural random number generation API (CONFIG_ARCH_RANDOM)

Memory setup --->
  Maximum size of supported physical memory in bits (42-53) (CONFIG_MAX_PHYSMEM_BITS)
  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 --->
    Maximum number of PCI functions (1-4096) (CONFIG_PCI_NR_FUNCTIONS)
    Support for PCI Hotplug ---> (common code option CONFIG_HOTPLUG_PCI)
      System z PCI Hotplug Support (CONFIG_HOTPLUG_PCI_S390)

... Support for CHSC subchannels (CONFIG_CHSC_SCH)
  SCM bus driver (CONFIG_SCM_BUS)
  Support for EADM subchannels (CONFIG_EADM_SCH)
  Support for VFIO-CCW subchannels (CONFIG_VFIO_CCW+)

Figure 208: General architecture-specific kernel configuration menu options. The  |  symbols indicate dependencies on preceding options. Options with more complex dependencies are marked with an asterisk (*).
Enable seccomp to safely compute untrusted bytecode (CONFIG_SECCOMP)*

Virtualization --->
- Pseudo page fault support (CONFIG_PFAULT)
- 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)

KVM --->
- s390 support for virtio devices (CONFIG_S390_GUEST)

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)
  - Kernel API for protected key handling (CONFIG_PKEY)
  - PAES cipher algorithms (CONFIG_CRYPTO_PAES_S390)
  - SHA1 digest algorithm (CONFIG_CRYPTO_SHA1_S390)
  - SHA256 digest algorithm (CONFIG_CRYPTO_SHA256_S390)
  - SHA384 and SHA512 digest algorithm (CONFIG_CRYPTO_SHA384_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)
  - CRC-32 algorithms (CONFIG_CRYPTO_CRC32_S390)

Figure 209: 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 (*).
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_ARCH_RANDOM

Enable the s390 architectural random number generation API to provide random data for all consumers within the Linux kernel.

When enabled the arch_random_* functions declared in linux/random.h are implemented. The implementation is based on the s390 CPACF instruction subfunction TRNG which provides a real true random number generator.

If unsure, say Y.

CONFIG_CHECK_STACK

This option enables the compiler option -mstack-guard and -mstack-size if they are available. If the compiler supports them it will emit additional code to each function prolog to trigger an illegal operation if the kernel stack is about to overflow.

Say N if you are unsure.

CONFIG_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".

**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_CRC32_S390**

Select this option if you want to use hardware accelerated implementations of CRC algorithms. With this option, you can optimize the computation of CRC-32 (IEEE 802.3 Ethernet) and CRC-32C (Castagnoli).

It is available with IBM z13 or later.

**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_PAES_S390**

This is the s390 hardware accelerated implementation of the AES cipher algorithms for use with protected key.

Select this option if you want to use the paes cipher for example to use protected key encrypted devices.

Depends on PKEY.

**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_EMU_SIZE

Select the default size by which the memory is chopped and then assigned to emulated NUMA nodes.

This can be overridden by specifying

emu_size=<n>

on the kernel command line where also suffixes K, M, G, and T are supported.

Depends on NUMA_EMU.

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_HOTPLUG_PCI_S390

Say Y here if you want to use the System z PCI Hotplug driver for PCI devices. Without this driver it is not possible to access stand-by PCI functions nor to deconfigure PCI functions.

When in doubt, say Y.

Depends on S390 & & 64BIT.

64BIT is a common code option. S390 is an implicitly selected option.

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_KEXEC_FILE

Enable the kexec file based system call. In contrast to the normal kexec system call this system call takes file descriptors for the kernel and initramfs as arguments.

Depends on CRYPTO [=y] && CRYPTO_SHA256 [=y] && CRYPTO_SHA256_S390 [=y].

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_Z10

This option is part of a choice section (MARCH_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12 | MARCH_Z13 | MARCH_Z14).

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_Z13

This option is part of a choice section (MARCH_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12 | MARCH_Z13 | MARCH_Z14).

Select this to enable optimizations for IBM z13s and z13 (2965 and 2964 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_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12 | MARCH_Z13 | MARCH_Z14).

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_Z14

This option is part of a choice section (MARCH_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12 | MARCH_Z13 | MARCH_Z14).

Select this to enable optimizations for IBM z14 ZR1 and z14 (3907 and 3906 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_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12 | MARCH_Z13 | MARCH_Z14).

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_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12 | MARCH_Z13 | MARCH_Z14).

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_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12 | MARCH_Z13 | MARCH_Z14).

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_Z900 | MARCH_Z990 | MARCH_Z9_109 | MARCH_Z10 | MARCH_Z196 | MARCH_ZEC12 | MARCH_Z13 | MARCH_Z14).

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_MAX_PHYSMEM_BITS

This option specifies the maximum supported size of physical memory in bits. Supported is any size between $2^{42}$ (4TB) and $2^{53}$ (8PB). Increasing the number of bits also increases the kernel image size. By default 46 bits (64TB) are supported.

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. Default is 64.

This is purely to save memory - each supported CPU adds approximately sixteen kilobytes to the kernel image.

Depends on SMP.

CONFIG_NUMA_EMU

Numa emulation mode will split the available system memory into equal chunks which then are distributed over the configured number of nodes in a round-robin manner.

The number of fake nodes is limited by the number of available memory chunks (i.e. memory size / fake size) and the number of supported nodes in the kernel.

The CPUs are assigned to the nodes in a way that partially respects the original machine topology (if supported by the machine). Fair distribution of the CPUs is not guaranteed.

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.

CONFIG_PCI_NR_FUNCTIONS

This allows you to specify the maximum number of PCI functions which this kernel will support.
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_PKEY

With this option enabled the pkey kernel module provides an API for creation and handling of protected keys. Other parts of the kernel or userspace applications may use these functions.
Select this option if you want to enable the kernel and userspace API for protected key handling.
Please note that creation of protected keys from secure keys requires to have at least one CEX card in coprocessor mode available at runtime.
Depends on ZCRYPT.

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_GUEST

Enabling this option adds support for virtio based paravirtual device drivers on s390.
Select this option if you want to run the kernel as a guest under the KVM hypervisor.

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_TOPOLOGY

Topology scheduler support improves the CPU scheduler’s decision making when dealing with machines that have multi-threading, multiple cores or multiple 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_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_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12 | TUNE_Z13 | TUNE_Z14).

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.

Tune the generated code for the target processor for which the kernel will be compiled.

CONFIG_TUNE_Z10

This option is part of a choice section (TUNE_DEFAULT | TUNE_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12 | TUNE_Z13 | TUNE_Z14).

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_Z13

This option is part of a choice section (TUNE_DEFAULT | TUNE_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12 | TUNE_Z13 | TUNE_Z14).

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_Z14

This option is part of a choice section (TUNE_DEFAULT | TUNE_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12 | TUNE_Z13 | TUNE_Z14).

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_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12 | TUNE_Z13 | TUNE_Z14).

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_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12 | TUNE_Z13 | TUNE_Z14).

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_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12 | TUNE_Z13 | TUNE_Z14).

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_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12 | TUNE_Z13 | TUNE_Z14).

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_Z900 | TUNE_Z990 | TUNE_Z9_109 | TUNE_Z10 | TUNE_Z196 | TUNE_ZEC12 | TUNE_Z13 | TUNE_Z14).

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_VFIO_CCW

This driver allows usage of I/O subchannels via VFIO-CCW.
To compile this driver as a module, choose M here: the module will be called vfio_ccw.
Depends on S390_CCW_IOMMU && VFIO_MDEV.
S390_CCW_IOMMU and VFIO_MDEV are common code options.

CONFIG_ZCRYPT

Select this option if you want to enable support for s390 cryptographic adapters like:
+ PCI-X Cryptographic Coprocessor (PCIXCC)
+ Crypto Express 2,3,4 or 5 Coprocessor (CEXxC)
+ Crypto Express 2,3,4 or 5 Accelerator (CEXxA)
+ Crypto Express 4 or 5 EP11 Coprocessor (CEXxP)
Device driver-related options

The device driver-related options comprise all z/Architecture specific options in the Device Drivers main menu.

Figure 210 on page 791 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 780.

---

Device Drivers -->

---

Block devices -->

---

S/390 block device drivers (depends on S390 && BLOCK) ---

XPRAM disk support
DCSSBLK support
Support for DASD devices
  - Profiling support for dasd devices
  - Support for ECKD Disks
  - Support for FBA Disks
  - Support for DIAG access to Disks
  - Extended error reporting (EER)
Support for Storage Class Memory

Misc devices -->

GenWQE PCIe Accelerator

SCSI device support -->

SCSI low-level drivers -->

FCP host bus adapter driver for IBM eServer zSeries

Network device support -->

S/390 network device drivers (depends on NETDEVICES && S390) -->

Lan Channel Station Interface
CTC and MPC SNA device support
IUCV network device support (VM only)
IUCV special message support (VM only)
  - Deliver IUCV special messages as uevents (VM only)
Gigabit Ethernet device support
  - qeth layer 2 device support
  - qeth layer 3 device support
Support for ISM vPCI Adapter

---

Figure 210: 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)

Hardware Random Number Generator Core support ---> (common code option CONFIG_HW_RANDOM)

S390 True Random Number Generator support (CONFIG_HW_RANDOM_S390)

--- S/390 character device drivers (depends on S390) ---

- Support for locally attached 3270 terminals (CONFIG_TN3270)
- Support for full-screen 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 console on SCLP VT220-compatible terminal (CONFIG_SCLP_VT220_TTY)

Support for Call Home via Asynchronous SCLP Records (0000000000) Component ID for Call Home (CONFIG_SCLP_ASYNC_ID)

Support for file transfers from HMC drive CD/DVD-ROM (CONFIG_HMC_DRV)

--- S/390 tape device support ---

- Support for 3480/3490 tape hardware (CONFIG_S390_TAPE_34XX)
- Support for 3590 tape hardware (CONFIG_S390_TAPE_3590)

Support for the z/VM recording system services (VM only) (CONFIG_VMLGRDR)*

Support for the z/VM CP interface (CONFIG_VMCP)

- Memory in MiB reserved for z/VM CP interface (CONFIG_MONREADER)*

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 --->

System z diag288 Watchdog (CONFIG_DIAG288_WATCHDOG)

...
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 diag288_wdt.

**CONFIG_GENWQE**
Enables PCIe card driver for IBM GenWQE accelerators. The user-space interface is described in include/linux/genwqe/genwqe_card.h.

Depends on PCI.

CONFIG_HMC_DRV

This option enables support for file transfers from a Hardware Management Console (HMC) drive CD/DVD-ROM. It is available as a module, called hmcdrv, and also as kernel built-in. There is one optional parameter for this module: cachesModule=N, which modifies the transfer cache size from its default value 0.5 MB to N bytes. If N is zero, then no caching is performed.

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_HW_RANDOM_S390

This driver provides kernel-side support for the True Random Number Generator available as CPACF extension on modern s390 hardware platforms.
To compile this driver as a module, choose M here: the module will be called s390-trng.
Depends on the common code HWRANDOM option.

CONFIG_ISM

Select this option if you want to use the Internal Shared Memory vPCI Adapter.
To compile as a module choose M. The module name is ism. If unsure, choose N.
Depends on PCI && SMC.
SMC is a common code option.

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 && ETHERNET.

IP_MULTICAST and ETHERNET are common code options. 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_ASYNC_ID

The Component ID for Call Home is used to identify the correct problem reporting queue the call home records should be sent to.

If you are unsure, please use the default value "000000000".

CONFIG_SCLP_CONSOLE

Include support for using an IBM HWC line-mode terminal as the Linux system console.

Depends on SCLP_TTY.

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_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_VMCP_CMA_SIZE**

Specify the default amount of memory in MiB reserved for the z/VM CP interface. If needed this memory is used for large contiguous memory allocations. The default can be changed with the kernel command line parameter "vmcp_cma".  
Depends on VMCP.

**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.

Depends on SCSI_FC_ATTRS.

SCSI and SCSI_FC_ATTRS are common code options. S390 is an implicitly selected option.
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 Z and LinuxONE publications are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF file and want to request a Web-based format for this publication, 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:
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:

```
vmhalt = <COMMAND>
```

**Repetition**

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

```
<repeat>
```

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

```
<repeat>
```

**Defaults**

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

```
A
B
C
```

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

**Required Choices**

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

```
A
B
C
```

Here you must enter either A or B or C.

**Optional Choice**

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

```
A
B
C
```

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

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

© Copyright IBM Corp. 2000, 2018
Glossary

This glossary includes IBM product terminology as well as selected other terms and definitions.

Additional information can be obtained in:


• The Information Technology Vocabulary developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1).


• Internet Request for Comments: 1208, Glossary of Networking Terms

• Internet Request for Comments: 1392, Internet Users' Glossary


Numerics

10 Gigabit Ethernet
An Ethernet network with a bandwidth of 10000-Mbps.

3215
IBM console printer-keyboard.

3270
IBM information display system.

3370, 3380 or 3390
IBM direct access storage device (disk).

3480, 3490, 3590
IBM magnetic tape subsystem.

3DES
See Triple Data Encryption Standard.

9336 or 9345
IBM direct access storage device (disk).

A

address space
The range of addresses available to a computer program or process. Address space can refer to physical storage, virtual storage, or both.
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 Z (CCL) replaces the 3745/6 Communication Controller so that the Network Control Program (NCP) software can continue to provide business critical functions like SNI, XRF, BNN, INN, and SSCP takeover. This allows you to leverage your existing NCP functions on a "virtualized" communication controller within the Linux on Z environment.

CEC
(Central Electronics Complex). A synonym for CPC.

channel subsystem
The programmable input/output processors of IBM 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.

compatible disk layout
A disk structure for Linux on Z which allows access from other IBM Z operating systems. This replaces the older Linux disk layout.

Console
In Linux, an output device for kernel messages.

CPC
(Central Processor Complex). A physical collection of hardware that includes main storage, one or more central processors, timers, and channels. Also referred to as a CEC.

CRC
cyclic redundancy check. A system of error checking performed at both the sending and receiving station after a block-check character has been accumulated.

CSMA/CD
carrier sense multiple access with collision detection

CTC
channel to channel. A method of connecting two computing devices.
CUU
control unit and unit address. A form of addressing for IBM 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 Z, a device that is online, but to which Linux can no longer find a connection. Reasons include:
• The device was physically removed
• The device was logically removed, for example, with a CP DETACH command in z/VM
• The device was varied offline

E

ECKD
extended count-key-data device. A disk storage device that has a data transfer rate faster than some processors can utilize and that is connected to the processor through use of a speed matching buffer. A specialized channel program is needed to communicate with such a device.

ESCON
enterprise systems connection. A set of IBM products and services that provide a dynamically connected environment within an enterprise.

Ethernet
A 10-Mbps baseband local area network that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and delayed retransmission. Ethernet uses CSMA/CD.

F

Fast Ethernet (FENET)
Ethernet network with a bandwidth of 100 Mbps
FBA
fixed block architecture. An architecture for a virtual device that specifies the format of and access mechanisms for the virtual data units on the device. The virtual data unit is a block. All blocks on the device are the same size (fixed size). The system can access them independently.

FDDI

fibre channel
A technology for transmitting data between computer devices. It is especially suited for attaching computer servers to shared storage devices and for interconnecting storage controllers and drives.

FTP
file transfer protocol. In the Internet suite of protocols, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

G

Gigabit Ethernet (GbE)
An Ethernet network with a bandwidth of 1000-Mbps

H

hardware console
A service-call logical processor that is the communication feature between the main processor and the service processor.

Host Bus Adapter (HBA)
An I/O controller that connects an external bus, such as a Fibre Channel, to the internal bus (channel subsystem).

In a Linux environment HBAs are normally virtual and are shown as an FCP device.

HMC
hardware management console. A console used to monitor and control hardware such as the IBM Z microprocessors.

HFS
hierarchical file system. A system of arranging files into a tree structure of directories.

I

intraensemble data network (IEDN)
A private 10 Gigabit Ethernet network for application data communications within an ensemble. Data communications for workloads can flow over the IEDN within and between nodes of an ensemble. All of the physical and logical resources of the IEDN are configured, provisioned, and managed by the Unified Resource Manager.
**intranode management network (INMN)**

A private 1000BASE-T Ethernet network operating at 1 Gbps that is required for the Unified Resource Manager to manage the resources within a single zEnterprise node. The INMN connects the Support Element (SE) to the zEnterprise 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**


**IUCV**

inter-user communication vehicle. A z/VM facility for passing data between virtual machines and z/VM components.

**K**

**kernel**

The part of an operating system that performs basic functions such as allocating hardware resources.

**kernel module**

A dynamically loadable part of the kernel, such as a device driver or a file system.

**kernel image**

The kernel when loaded into memory.
**L**

**LCS**
LAN channel station. A protocol used by OSA.

**LDP**
Linux Documentation Project. An attempt to provide a centralized location containing the source material for all open source Linux documentation. Includes user and reference guides, HOW TOs, and FAQs. The homepage of the Linux Documentation Project is [www.linuxdocs.org](http://www.linuxdocs.org)

**Linux**
a variant of UNIX which runs on a wide range of machines from wristwatches through personal and small business machines to enterprise systems.

**Linux disk layout**
A basic disk structure for Linux on Z. Now replaced by compatible disk layout.

**Linux on Z**
the port of Linux to the z/Architecture.

**LPAR**
logical partition of an IBM Z mainframe.

**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.

**NIC**
Network interface card. The physical interface between the IBM mainframe and the network.

**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.

**POR**
Power-on reset

**POSIX**
Portable Operating System Interface for Computer Environments. An IEEE operating system standard closely related to the UNIX system.

**router**
A device or process which allows messages to pass between different networks.

**SA/SE**
Stand alone support element. See SE.
SE

• An internal control element of a processor that assists in many of the processor operational functions.
• A hardware unit that provides communications, monitoring, and diagnostic functions to a central processor complex.

SNA

systems network architecture. The IBM architecture that defines the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. The layered structure of SNA allows the ultimate origins and destinations of information (the users) to be independent of and unaffected by the specific SNA network services and facilities that are used for information exchange.

SNMP (Simple Network Management Protocol)

In the Internet suite of protocols, a network management protocol that is used to monitor routers and attached networks. SNMP is an application layer protocol. Information about devices managed is defined and stored in the application's Management Information Base (MIB).

Sysctl

system control programming manual control (frame). A means of dynamically changing certain Linux kernel parameters during operation.

T

TDEA

See Triple Data Encryption Standard.

TDES

See Triple Data Encryption Standard.

Telnet

A member of the Internet suite of protocols which provides a remote terminal connection service. It allows users of one host to log on to a remote host and interact as if they were using a terminal directly attached to that host.

Terminal

A physical or emulated device, associated with a keyboard and display device, capable of sending and receiving information.

Triple Data Encryption Standard

A block cipher algorithm that can be used to encrypt data transmitted between managed systems and the management server. Triple DES is a security enhancement of DES that employs three successive DES block operations.

U

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 or more LANs that are configured (using management software) so that they can communicate as if they were attached to the same wire, when in fact they are located on a number of different LAN segments. Because VLANs are based on logical rather than physical connections, they are extremely flexible.

**volume**
A data carrier that is usually mounted and demounted as a unit, for example a tape cartridge or a disk pack. If a storage unit has no demountable packs the volume is the portion available to a single read/write mechanism.

**Z**

**z114**
IBM zEnterprise 114.

**z13**
IBM z13.

**z13s**
IBM z13s.

**z14**
IBM z14.

**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 Z and LinuxONE publications

You can find publications for Linux on Z and LinuxONE on IBM Knowledge Center and on developerWorks. These publications are available on IBM Knowledge Center at www.ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_lib.html

- Device Drivers, Features, and Commands (distribution-specific editions)
- Using the Dump Tools (distribution-specific editions)
- Running Docker Containers on IBM Z, SC34-2781
- KVM Virtual Server Quick Start, SC34-2753
- KVM Virtual Server Management, SC34-2752
- How to use FC-attached SCSTI devices with Linux on z Systems, SC33-8413
- libica Programmer's Reference, SC34-2602
- Exploiting Enterprise PKCS #11 using openCryptoki, SC34-2713
- Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide, SC33-8294
- Pervasive Encryption for Data Volumes, SC34-2782
- Linux on z Systems Troubleshooting, SC34-2612
- Kernel Messages, SC34-2599
- How to Improve Performance with PAV, SC33-8414
- How to Set up a Terminal Server Environment on z/VM, SC34-2596

You can also find these publications on developerWorks at www.ibm.com/developerworks/linux/linux390/documentation_dev.html

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

z/VM publications

The publication numbers listed are for z/VM version 7.

For the complete library including other versions, see

www.ibm.com/vm/library

- z/VM: Connectivity, SC24-6267
- z/VM: CP Commands and Utilities Reference, SC24-6268
- z/VM: CP Planning and Administration, SC24-6271
- z/VM: CP Programming Services, SC24-6272
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
• z/VM: Secure Configuration Guide, SG24-6323
• 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
• Reduce Storage Occupancy and Increase Operations Efficiency with IBM zEnterprise Data Compression, SG24-8259

Other IBM Z publications

General IBM Z publications that might be of interest in the context of Linux on Z and LinuxONE.

• 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, zEC12/zBC12, z13/ z13s, and z14, SA23-2261

Networking publications

• HiperSockets Implementation Guide, SG24-6816
• Open Systems Adapter-Express Customer’s Guide and Reference, SA22-7935
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 Z and LinuxONE including z/VM, I/O connectivity, and cryptography.

- For CMS and CP Data Areas, Control Block information, and the layout of the z/VM monitor records see
  

- For I/O connectivity on IBM 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
## Index

### Special Characters

*ACCOUNT, z/VM record 457
*LOGREC, z/VM record 457
*MONITOR record reader 451
*SYMPTOM, z/VM record 457
/debug, mount point x
/proc, mount point x
/proc, sysinfo
/sys, mount point x
/sys/kernel/debug, mount point x

### Numerics

10 Gigabit Ethernet
SNMP 311
1000Base-T Ethernet
LAN channel station 319
SNMP 311
1000Base-T, Ethernet 245
1750, control unit 137
2105, control unit 137
2107, control unit 137
3088, control unit 319, 325
31-bit ix
3270 emulation
3270 terminal device driver
switching the views of 51
3370, DASD 137
3380, DASD 137
3390, DASD 137
3480 tape drive 225
3490 tape drive 225
3590 tape drive 225
3592 tape drive 225
3880, control unit 137
3990, control unit 137
3DES, See TDES
6310, control unit 137
64-bit ix
9336, DASD 137
9343, control unit 137
9345, DASD 137

### A

acceleration, in-kernel cryptography 527
access control
osasnmrd 313
access_denied
zfcp attribute (FCP LUN) 208
zfcp attribute (port) 197
zfcp attribute (SCSI device) 208
access_shared
zfcp attribute 208
accessibility 799
ACCOUNT, z/VM record 457
actions, shutdown 105
ACTIVE_CONSOLES 44
adapter outage 294
adapter virtualization 498
add, DCSS attribute 472
adding and removing cryptographic adapters 507
Address Resolution Protocol, See ARP
AES 527
aes_s390, kernel module 530
AF_IUCV
  
  addressing sockets in applications 347
  set up devices for addressing 346
AF_IUCV address family
  set up support for 346
af_iucv, kernel module 346, 347
AgentX protocol 311
alias
DASD attribute 172
allow_lun_scan=, module parameters 186
AP
devices 7
AP bus
attributes 510
ap module
parameters 28
AP queue 499
ap_functions
  
  cryptographic adapter attribute 502
ap_interrupts
  
  cryptographic adapter attribute 506
API
cryptographic 511
FC-HBA 181
GenWQE zlib 425
zfcp HBA 218
apmask, cryptographic device driver attribute 508
apmask=
  
  kernel parameter 500
app, messages 570
APPLDATA monitor records
  
  monitoring Linux instances 435
appdata_mem, kernel module 439
appdata_net_sum, kernel module 439
appdata_os, kernel module 439
APPLDATA, monitor stream 439
applet
  
  emulation of the HMC Operating System Messages 55
applications
  
  addressing AF_IUCV sockets in 347
aqmask, cryptographic device driver attribute 508
aqmask=
  
  kernel parameter 500
arch
  
  trng counter 520
ARP
  
  proxy ARP 287
ARP (continued)
query/purge OSA-Express ARP cache 702
attributes
device 9
for CCW devices 9
for subchannels 13
qeth 260–263
setting 10
authorization
CPU-measurement counter facility 544
auto-configuration
managing 20
override fails 567
auto-detection
DASD 147
autoconfiguration, IPv6 252
automatic problem reporting
activating 559
autopurge, z/VM recording attribute 461
autorecording, z/VM recording attribute 460
availability
common CCW attribute 9
DASD attribute 154
avg_*, cmf attributes 540, 541
avg_control_unit_queuing_time, cmf attribute 541
avg_device_active_only_time, cmf attribute 541
avg_device_busy_time 541
avg_device_busy_time, cmf attribute 541
avg_device_connect_time, cmf attribute 541
avg_device_disconnect_time, cmf attribute 541
avg_function_pending_time, cmf attribute 541
avg_initial_command_response_time, cmf attribute 541
avg_sample_interval, cmf attribute 541
avg_utilization, cmf attribute 541
buffer-overflow protection 535
buffer, CPU-measurement sampling facility 547
buffer, CTCM attribute 331
buffer, IUCV attribute 341
bus ID 9
byte_counter
prandom attribute 516
trng attribute 520

C

cache-size=, module parameters 414
Call Home
callhome attribute 560
kernel configuration menu option 559
call logical processor device driver 405
callhome
Call Home attribute 560
capability
CPU sysfs attribute 362
capped
S/390 hypervisor file system attribute, z/VM guest 397
card_type, qeth attribute 269
card_version, zfcp attribute 189
case conversion 55
Castagnoli 527
CBC 527
CCA coprocessor 495
CCW
channel measurement facility 539
common attributes 9
devices 7
group devices 7
hotplug events 17
setting attributes 576
setting devices online/offline 576
CCW terminal device
switching on- or offline 52
CD-ROM, loading Linux 92
Central Processor Assist for Cryptographic Function, See
CPACF
CEX2A (Crypto Express2) 495
CEX2C (Crypto Express2) 495
CEX3A (Crypto Express3) 495
CEX3C (Crypto Express3) 495
CEX4A (Crypto Express4S) 495
CEX4C (Crypto Express4S) 495
CEX4P (Crypto Express4S) 495
CEX5A (Crypto Express5S) 495
CEX5C (Crypto Express5S) 495
CEX5P (Crypto Express5S) 495
CEX6A (Crypto Express6S) 495
CEX6C (Crypto Express6S) 495
CEX6P (Crypto Express6S) 495
change, CPU capability 362
changes
summary vii
channel measurement facility
cmb_enable attribute 540
features 539
kernel parameters 539
read-only attributes 540
channel path
changing status 578
channel path (continued)
determining usage 565
ensuring correct status 565
list 565
channel path availability
planned changes 565
unplanned changes 565
channel path ID 14
channel path measurement 13
channel subsystem view 12
channel-attached tape 225
chccwdev 10
chccwdev, Linux command 576
chcp, Linux command 578
chcpu, Linux command 361
chcpumf, Linux command 580
checksum
inbound 274
outbound 275
CHID
mapping physical to virtual 16
Chinese-Remainder Theorem 495
chiucvallow, Linux command 43
CHPID
in sysfs 14
map to PCHID 16
online attribute 14, 15
chpids, subchannel attribute 13
chreipl, Linux command 581
chsc
device driver 401
kernel configuration menu options 401
CHSC
device names 402
device nodes 402
CHSC subchannel device driver
programming interfaces 402
set up 401
chsc_sch, kernel module 401
chshut, Linux command 585
chunksize
prandom attribute 516
chunksize=, module parameters 514
chzcrypt, Linux command 587
chzdev 10
chzdev command 377
chzdev, Linux command 589
cio_ignore
disabled wait 566
procs interface 759
cio_ignore, Linux command 596
cio_ignore=, kernel parameter 758
clock synchronization
enabling and disabling 409
switching on and off 409
CLP
device node 405
kernel configuration menu options 405
CLP device driver
programming interface 405
cm_enable
channel subsystem sysfs attribute 13
cmb_enable
cmf attribute 540
cmb_enable (continued)
common CCW attribute 9
tape attribute 232
cmd= 112
cmd=, module parameters 112
cmf.format=, kernel parameter 539
cmf.maxchannels=, kernel parameter 539
cmm
avoid swapping with 437
background information 436
CMM
kernel configuration menu options 489
unload module 566
cmm, kernel module 489, 490
cmm.sender=, kernel parameters 490
CMMA 762
cmma=, kernel parameter 762
CMS disk layout 141
CMS1 labeled disk 141
cmsfs-fuse, Linux command 599
code page
for x3270 49
Collaborative Memory Management Assist 762
collecting QETH performance statistics 277
commands, Linux
chccwdev 576
chcp 578
chcpu 361
chcpumf 580
chiucvallow 43
chreipl 581
chshut 585
chzcrypt 587
cio_ignore 596
cmsfs-fuse 599
cpacfstats 603
cpuplugd 607
dasdfmt 615
dasdrat 620
dasview 622
dmesg 6
dumpconf 105
execstack 535
fdasdrat 633
genwqe_echo 424
genwqe_gzip 417
gunzip 422
gzip 422
gzunzip 417
hmcdrunfs 641
htop 644
icainfo 575
icastats 575
ip 4
iucvconn 43
iucvty 43
lschp 654
lscpumf 656
lsscss 659
lsldap 662
lshmc 665
lsldasds 666
lsqeth 669
commands, Linux (continued)
lsreipl 670
lsscm 671
lsshut 673
lstape 674
lszfcp 687
mknod 3
mon_fsstatd 689
mon_procd 694
osasnmpd 701
qetharp 702
qethconf 704
qethqoat 707
readelf 535
readlink 6
scsi_logging_level 710
sg_inq 674
sncap 720
snipl 115
stonith 132
tape390_crypt 727
tape390_display 731
tar 422
time 422
tunedasd 733
vmconvert 743
vmcp 737
vmur 739
zdfts 747
zfcp_ping 219
zfcp_show 219
zipl 59
znetconf 752
zpctl 755
commands, z/VM
sending from Linux 737
communication facility
Inter-User Communication Vehicle 345
compatible disk layout 139
compression
GenWQE 417
tape 233
conceal= 112
conceal=, module parameters 112
CONFIG_AFIUCV 346, 781
CONFIG_APPLDATA_BASE 439, 781
CONFIG_APPLDATA_MEM 439, 781
CONFIG_APPLDATA_NET_SUM 439, 782
CONFIG_APPLDATA_OS 439, 782
CONFIG_ARCH_RANDOM 782
CONFIG_BLK_DEV_INTEGRITY 183
CONFIG_BLK_DEV_XPRAM 236, 792
CONFIG_CCW 146
CONFIG_CHECK_STACK 782
CONFIG_CHSC_SCH 401, 782
CONFIG_CMM 489, 783
CONFIG_CMM_IUCV 490, 783
CONFIG_COMPAT 783
CONFIG_CRYPTO_AES_S390 528, 783
CONFIG_CRYPTO_CRC32_S390 528, 783
CONFIG_CRYPTO_DES_S390 528, 783
CONFIG_CRYPTO_GHASH_S390 528, 783
CONFIG_CRYPTO_PAES_S390 528, 783
CONFIG_CRYPTO_SHA1_S390 528, 784
CONFIG_CRYPTO_SHA256_S390 528, 784
CONFIG_CRYPTO_SHA512_S390 528, 784
CONFIG_CTCM 326
CONFIG_DASD 146
CONFIG_DASD_DIAG 146
CONFIG_DASD_ECKD 146
CONFIG_DASD_EER 146
CONFIG_DASD_FBA 146
CONFIG_DASD_PROFILE 146
CONFIG_DCSSBLK 793
CONFIG_DIAG288_WATCHDOG 110
CONFIG_EADM_SCH 222
CONFIG_EMU_SIZE 368
CONFIG_ETHERNET 320
CONFIG_FUSE_FS 413
CONFIG_GENWQE 420
CONFIG_HIBERNATION 101
CONFIG_HMC_DRV 413
CONFIG_HOTPLUG_CPU 784
CONFIG_HOTPLUG_PCI 539
CONFIG_HUGETLBFS 391
CONFIG_HVIC_IUCV 40
CONFIG_HW_RANDOM_S390 519
CONFIG_IP_MULTICAST 258
CONFIG_ISM 357
CONFIG_IUCV 339
CONFIG_KEXEC_FILE 785
CONFIG_KMSG_IDS 569
CONFIG_LCS 320
CONFIG_MAGIC_SYSRQ 53
CONFIG_MARCH_Z10 785
CONFIG_MARCH_Z14 785
CONFIG_MARCH_Z196 785
CONFIG_MARCH_Z9_109 786
CONFIG_MARCH_Z900 785
CONFIG_MARCH_Z990 785
CONFIG_MARCH_ZEC12 786
CONFIG_MAX_PHYSEM_MEM_BITS 786
CONFIG_MEMORY_HOTPLUG 372
CONFIG_MEMORY_HOTREMOVE 372
CONFIG_MIGRATION 372
CONFIG_MONREADER 794
CONFIG_MONWRITER 447
CONFIG_NETIUCV 339
CONFIG.Nodes_SHIFT 368
CONFIG_NR_CPUS 786
CONFIG_NUMA 368
CONFIG_NUMA_BALANCING 368
CONFIG_NUMA_BALANCING_DEFAULT_ENABLED 368
CONFIG_NUMA_EMU 368
CONFIG_PACKAGE 786
CONFIG_PKEY 787
CONFIG_PFKEY 787
CONFIG_PM 101
CONFIG_QDI0 183
CONFIG_QETH 258
CONFIG_QETH_L2 258
CONFIG_QETH_L3 259
CONFIG_S390_GUEST 787
CONFIG_S390_HYPFS_FS 395, 787
CONFIG_S390_PRNG 513, 787
CONFIG_S390_TAPE 229, 795
CONFIG_S390_TAPE_34XX 229, 795
CONFIG_S390_TAPE 3590 229, 795
CONFIG_S390_VMUR 465, 795
CONFIG_SCHED_TOPOLOGY 368, 787
CONFIG_SCLP_ASYNC 559, 795
CONFIG_SCLP_ASYNC_ID 796
CONFIG_SCLP_CONSOLE 40, 796
CONFIG_SCLP_TTY 40, 796
CONFIG_SCLP_VT220_CONSOLE 40, 796
CONFIG_SCLP_VT220_TTY 40, 796
CONFIG_SCM_BLOCK 222, 796
CONFIG_SCM_BUS 222, 787
CONFIG_SCSI 183
CONFIG_SCSI_FC_ATTRS 183
CONFIG_SECCOMP 787
CONFIG_SMC 350
CONFIG_SMC_DIAG 350
CONFIG_SMP 788
CONFIG_SMSGIUCV 484, 490, 796
CONFIG_SMSGIUCV_EVENT 484, 796
CONFIG_STACK_GUARD 788
CONFIG_TN3215 40, 796
CONFIG_TN3215_CONSOLE 40, 797
CONFIG_TN3270 40, 797
CONFIG_TN3270_CONSOLE 40, 797
CONFIG_TN3270_CONSOLE 40, 797
CONFIG_TN3270_FS 40, 797
CONFIG_TN3270_TTY 40, 797
CONFIG_TR 320
CONFIG_TRANSPARENT_HUGEPAGE 391
CONFIG_TUNE_DEFAULT 788
CONFIG_TUNE_210 788
CONFIG_TUNE_213 788
CONFIG_TUNE_214 789
CONFIG_TUNE_2196 789
CONFIG_TUNE_29_109 789
CONFIG_TUNE_2900 789
CONFIG_TUNE_2990 789
CONFIG_TUNE_2EC12 789
CONFIG_VFIO_CCW 790
CONFIG_VMCP 479, 797
CONFIG_VMCP_CMA_SIZE 479, 797
CONFIG_VMLOGDR 459, 797
CONFIG_WATCHDOG 797
CONFIG_WATCHDOG_NOWAYOUT 111
CONFIG_ZCRYPT 499, 790
CONFIG_ZFCP 183, 798
configuration file
CPU control 608
cpuplugd 614
memory control 609
configuration options 146
configure LPAR I/O devices 565
configuring standby CPU 362
conmode=, kernel parameter 41
connection, IUCV attribute 339
console
definition 34
device names 35
device nodes 35, 36
mainframe versus Linux 34
console device driver
kernel configuration menu options 40
kernel parameter 42
overriding default driver 41
restricting access to HVC terminal devices 43
SCLP line-mode buffer page reuse 42
SCLP line-mode buffer pages 43
specifying preferred console 42
specifying the number of HVC terminal devices 43
console device drivers
device and console names 35
features 33
terminal modes 36
console=, kernel parameter 42
control characters 53
control program identification 555
control unit
1750 137
2105 137
2107 137
3880 137
3990 137
6310 137
9343 137
controlling automatic port scans 195
cooperative memory management
set up 490
coprocessor, cryptographic 495
core 361
core_siblings
CPU sysfs attribute 364
count
S/390 hypervisor file system attribute, z/VM guest 397
CP Assist for Cryptographic Function, See CPACF
CP commands
send to z/VM hypervisor 737
VINPUT 57
CP Error Logging System Service 457
CP VINPUT 57
CP1047 599
CPACF
in-kernel cryptography 527
kernel configuration menu options 519, 528
protected key 523
pseudorandom number 513
support modules, in-kernel cryptography 530
cpacfstats, Linux command 603
cpc_name attribute 411
CPI
set attribute 557
sysplex_name attribute 556
system_level attribute 556
system_name attribute 555
system_type attribute 556
CPI (control program identification) 555
CPU
controlling 606
managing 361
CPU capability change 362
CPU capacity
manage 720
CPU configuration 603
CPU control
complex rules 612
CPU control (continued)
configuration file 608
CPU hotplug
sample configuration file 614
CPU hotplug rules 610
CPU sysfs attributes
book_siblings 364
capability 362
core_siblings 364
dispatching 366
online 363
polarization 366
thread_siblings 364
CPU sysfs attributes
location of 361
cpu_delay
S/390 hypervisor file system attribute, z/VM guest 398
cpu_using
S/390 hypervisor file system attribute, z/VM guest 398
CPU-measurement counter facility 548
CPU-measurement facilities
chcpumf command 580
lscpumf command 656
CPU-measurement sampling facility
buffer limits 547
CPU, configuring standby 362
CPU, state 362
cpuplugd
complex rules 612
configuration file 614
service utility syntax 606
cpuplugd, Linux command 607
cputstat
cpuplugd keywords
use with historical data 611
cputime
S/390 hypervisor file system attribute, LPAR 397
cputime_us
S/390 hypervisor file system attribute, z/VM guest 397
CRC32 527
CRT 495
Crypto Express2 495
Crypto Express3 495
Crypto Express4 495
Crypto Express5 495
Crypto Express6S 495
cryptographic request processing 497
cryptographic adapter attributes 502
cryptographic adapters
adding and removing dynamically 507
detection 497
cryptographic coprocessor 495
cryptographic device
display information 678
Linux 499
LPAR 498
z/VM 499
cryptographic device driver
API 511
features 495
hardware and software prerequisites 496
setup 499
cryptographic device driver (continued)
See also zcrypt
cryptographic device nodes 497
cryptographic domain
control 498
usage 498
csulincl.h 511
CTC
activating an interface 331
CTC interface
recovery 332
CTC network connections 326
CTCM
buffer attribute 331
device driver 325
group attribute 327
kernel configuration menu options 326
online attribute 330
protocol attribute 329
subchannels 325
type attribute 329
ungroup attribute 328
cctm module 327
ctcm, kernel module 327
CTR 527
cutype
common CCW attribute 9	
tape attribute 232

D
DASD
access by udev-created device nodes 145
access by VOLSER 144
alias attribute 172
availability attribute 154
boot menu, LPAR example 88
boot menu, z/VM example 83
booting from 82, 86
boxed 154
CMS disk layout 141
compatible disk layout 139
control unit attached devices 137
device driver 137
device names 142
device nodes 143
discipline attribute 172
disk layout summary 142
displaying information 622
displaying overview 662
eer_enabled attribute 156
erplog attribute 158
expires attribute 160
extended error reporting 137
failfast attribute 159
features 137
forcing online 154
formatting ECKD 615
High Performance FICON 164
host_access_count attribute 171
hpf attribute 170
kernel configuration menu options 146
last_known_reservation_state attribute 168
Linux disk layout 141
DASD (continued)

module parameter 147, 149
online attribute 157, 158
partitioning 633
partitions on 138
path_autodisable attribute 169
path_interval attribute 169
path_threshold attribute 169
PAV 164
performance statistics 620
performance tuning 733
raw_track_access attribute 165
readonly attribute 173
reservation_policy attribute 167
safe_offline attribute 157
statistics 161
status attribute 174
timeout attribute 160, 174
uid attribute 174
use_diag attribute 155, 174
vendor attribute 174
virtual 137
volume label 139
dasd_diag_mod, kernel module 146
dasd_eckd_mod, kernel module 146
dasd_eer, kernel module 146
dasd_fba_mod, kernel module 146
dasd_mod, kernel module 146
DASD, SCSI, and tape 67
dasd=
  kernel parameter 147
  module parameter 147, 149
dasdfmt, Linux command 615
dasdstat, Linux command 620
dasdview, Linux command 622
data
  compression 417
  consistency checking, SCSI 216
  integrity extension 216
  integrity field 216
datarouter=, module parameters 186
dbfsize=, module parameters 186
DCSS
  access mode 473
  add attribute 472
  adding 472
  device driver 467
  device names 467
  device nodes 467
  exclusive-writable mode 467
  kernel configuration menu options 468
  minor number 473
  performance monitoring using 436
  remove attribute 475
  save attribute 474
  saving with properties 475
  seglist attribute 472
  shared attribute 473
  with options 468
dcssblk, kernel module 468
dcssblk.segments=, kernel parameter 468
dcssblk.segments=, module parameter 469
deactivating a qeth interface 273
ddebug feature 438
debugfs
  DASD statistics 161
  QDIO statistics 553
decompression, GenWQE 417
decryption 495
dedicated
  S/390 hypervisor file system attribute, z/VM guest 397
default_hugepagesize=, kernel parameters 392
delete
  zfcp sysfs attribute 214
delete, zfcp attribute 214
depth
  cryptographic adapter attribute 502
des_s390, kernel module 530
determine channel path usage 565
developerWorks ix
device
  interoperability matrix 177
device bus-ID
  of a qeth interface 271
device driver
  chsc 401
  CLP 405
  crypto 495
  CTCM 325
  DASD 137
  DCSS 467
  Generic Work Queue Engine 417
  HiperSockets 243
  HMC media 413
  internal shared memory 357
  LCS 319
  mix4_en 353
  monitor stream application 447
  NETIUCV 337
  OSA-Express (QDIO) 243
  overview 7
  PCle 427
  protected key 523
  pseudorandom number 513
  qeth 243
  SCLP_ASYNC 559
  SCSI-over-Fibre Channel, See zfcp
  smsgiucv_app 483
  storage-class memory 221
tape 225
  true random number 519
  vmcp 479
  vmur 465
  watchdog 109
  XPRAM 235
  z/VM *MONITOR record reader 451
  z/VM recording 457
  zcrypt 495
device drivers
  support of the FCP environment 178
device names
  CHSC 402
  console 35
  DASD 142
  DCSS 467
  random number 515
  storage-class memory 221
tape 225
device names (continued)
  vmcp 479
  vmur 465
  XPRAM 235
  z/VM *MONITOR record 454
  z/VM recording 457
device node
  prandom, non-root users 515
device nodes
  CHSC 402
  CLP 405
  console 35, 36
  DASD 143, 144
  DCSS 467
extended error reporting 150
  GenWQE 419
  random number 515
  SCSI 179
  SCSI, creating 181
  storage-class memory 221
  tape 227
  udev 4
  udev-created, SCSI 179
  vmcp 479
  vmur 465
  watchdog 113
  XPRAM 236
  z/VM *MONITOR record 454
  z/VM recording 457
  zcrypt 501
  zfcp 179
device numbers 3
device special file, See device nodes
device view
  by category 11
  by device drivers 11
device_blocked
  zfcp attribute (SCSI device) 208
devices
  alias 172
  attributes 9
  base 172
  corresponding interfaces 6
  ignoring 758
  in sysfs 9
  initialization errors 10
  working with newly available 10
devs=, module parameter 238
devtype
  common CCW attribute 9
  tape attribute 232
dhcp 306
DHCP
  required options 306
DIAG
  access method 155
DIAG access method
  for ECKD 142
  for FBA 142
DIAG call 777
  diag288_wdt, kernel module 110
  diag288_wdt.cmd=, kernel parameters 110
  diag288_wdt.conceal=, kernel parameters 110
  diag288_wdt.nowayout=, kernel parameters 110
diagnose call 777
diagnosis
  using XPRAM 236
diagnostics and troubleshooting 549
DIF 216
dif=, module parameters 186
Direct Access Storage Device, See DASD
Direct SNMP 311
disabled wait
  booting stops with 567
  cio_ignore 566
discipline
  DASD attribute 172
  qeth 259
discontiguous saved segments, See DCSS
disk layout
  CMS 141
  LDL 141
  summary 142
  Z compatible 139
dispatching
  CPU sysfs attribute 366
displaying information
  FCP channel and device 189
DIX 216
dmesg 6
Doc Buddy 570
domain, cryptographic 498
domain=
  kernel parameter 500
DPM 19
drivers, See device driver
dsn
  metadata file attribute 747
dsorg
  metadata file attribute 747
dump
  creating automatically after kernel panic 567
  dump device
  ECKD DASD 69
  dump file
    receive and convert 744
  dumpconf, Linux command 105
  dumped_frames, zfcp attribute 191
  DVD drive, HMC 413
  DVD, loading Linux 92
Dynamic Host Configuration Protocol, See DHCP
Dynamic Partitioning Manager 19
dynamic routing, and VIPA 294

E

EADM subchannels
  list 223
    working with 223
  eadm_sch, kernel module 222
EBCDIC
  conversion through cmsfs-fuse 599
  kernel parameters 81
ECB 527
ECKD
  devices 137
  disk layout summary 142
  raw_track_access attribute 165
ECKD type DASD
preparing for use 152
edit characters, z/VM console 57
EEDK 727
eer_enabled
   DASD attribute 156
EKM 727
emu_nodes=, kernel parameters 368
emu_size=, kernel parameters 368
emulation of the HMC Operating System Messages applet 55
encoding 599
encryption 495
encryption key manager 727
end-of-line character 56
end-to-end data consistency, SCSI 216
Enterprise PKCS#11 495
Enterprise Storage Server 137
environment variable 485
environment variables
   for CP special messages 485
   TERM 44
   ZIPLCONF 74
   ZLIB_CARD 421
   ZLIB_DEFLATE_IMPL 421
   ZLIB_INFLATE_IMPL 421
   ZLIB_TRACE 421
EP11
display device information 678
supported functions 495
EP11 coprocessor 495
ep11.h 511
erplog, DASD attribute 158
Error Logging System Service 457
error_frames, zfcp attribute 190
errorflag
   prandom attribute 516
escape character
   for terminals 56
ESS 137
Ethernet
   interface name 252
   interface name for LCS 320
   LAN channel station 319
etr
   online attribute 409
ETR 407, 409
etr=
kerneld parameter 408
etr=, kernel parameter 408
exclusive-writable mode
   DCSS access 467
execstack, Linux command 535
expanded memory 235
expires, DASD attribute 160
extended error reporting
   DASD 156
   device node 150
extended error reporting, DASD 137
extended remote copy 407
external encrypted data key 727
external time reference 407
failed
zfcp attribute (channel) 193
zfcp attribute (port) 199
failfast, DASD attribute 159
fake_broadcast, qeth attribute 283
Fast Ethernet
   LAN channel station 319
FBA
disk layout summary 142
FBA devices 137
FBA type DASD
   preparing for use 153
FC-HBA 181
FC-HBA API functions 218
FCP
   channel 177
   debugging 182
   device 177
   traces 182
FCP channel
displaying information 189
FCP device
displaying information 189
FCP devices
   listing 217
   status information 194
   sysfs structure 178
FCP environment 178
fcp_control_requests zfcp attribute 191
fcp_input_megabytes zfcp attribute 191
fcp_input_requests zfcp attribute 191
fcp_output_megabytes zfcp attribute 191
fcp_output_requests zfcp attribute 191
fdasd
   menu commands 636
   menu example 637
   options, example 639
fdasd menu 636
fdasd, Linux command 633
fdisk command 181
Federal Information Processing Standard 527, 533
Federal Information Processing Standard 140-2 533
Fibre Channel 177
file system
   hugetlbs 391
file systems
   cmsfs-fuse for z/VM minidisk 599
   sysfs 7
   XFS 216
   zdsfs for z/OS DASD 747
FIPS
   kernel configuration menu options 533
FIPS 140-2 533
FIPS restrictions 528
fips=, kernel parameter 533
Flash Express memory 221
flooding, qeth attribute 291
for performance measuring 537
formatting 152
FPGA 417
FTP server, loading Linux 92
full ECKD tracks 165
full-screen mode terminal 44
function_handle
    PCIe attribute 429
function_id
    PCIe attribute 429
functions
    qeth device driver 247

G
GB xi
Generic Work Queue Engine 417
GenWQE 417
genwqe_echo, command 424
genwqe_gzip, command 417
genwqe, kernel module 420
getxattr 599, 747
GHASH 527
ghash_s390, kernel module 530
giga xi
Gigabit Ethernet
    SNMP 311
GNU_STACK 535
gpfs, partition type 633
group
    CTCM attribute 327
    LCS attribute 321
    qeth attribute 263
group devices
    CTCM 325
    LCS 319
    qeth 251
guest console transcript
    vmur command 744
guest LAN sniffer 308
guest memory dump
    vmur command 744
guest swapping 566
gunzip, command 422
gzip, command 422
gunzip, command 417

H
hardware
    configuration 19
        random number 519
        service level 566
    hardware counter
        reading with perf tool 545
    hardware facilities 537
    hardware information 561, 562
    Hardware Management Console, See HMC
    hardware status, z90crypt
        hardware status, zcrypt 504
        online
            zcrypt sysfs attribute 504
            zcrypt
            hardware status 504
            zcrypt sysfs attribute
            online 504
    hardware_version, zfcp attribute 189
    hardware-acceleration, in-kernel cryptography 527
HBA API
    functions 218
HBA API support
    zfcp 218
high availability project 132
High Performance FICON 164
High Performance FICON, suppressing 147
high resolution polling timer 587
HiperSockets
    bridge port 256
    device driver 243
    interface name 252
    network traffic analyzer 307
    HiperSockets Network Concentrator 301
historical data
cuplugd keywords 611
HMC
    as terminal 50
    definition 35
    for booting Linux 80
    Integrated ASCII console applet 37
    Operating System Messages applet
        on HMC 37
    using in LPAR 37
    using on z/VM 37
    HMC DVD drive 415, 416
    HMC media
        list media contents 665
        mount media 641
    HMC media device driver
        kernel configuration menu options 413
    HMC media, device driver 413
    HMC Operating System Messages applet
        emulation of the 55
    HMC removable media
        assign to LPAR 415
        hmc_network attribute 411
        hmcdrv, kernel module 413
        hmcdrv.cachesize=, kernel parameters 414
        hmcdrvfs, kernel module 414
        hmcdrvfs, Linux command 641
    host_access_count
        DASD attribute 171
    hotplug
        adding memory 374
        CCW devices 17
        memory 371
    hotplug memory
        defining to LPAR 373
        defining to z/VM 373
        kernel configuration menu options 372
        large pages 392
    hotplug rules
        CPU 610
        memory 611
    hpf
        DASD attribute 170
        hsuid, qeth attribute 288
        hugepages=, kernel parameters 392
        hugetlbfs
            virtual file system 391
        HVC device driver 39
        hvc_iucv_allow=, kernel parameter 43
        hvc_iucv=, kernel parameter 43
interface (continued)
interface control check 169
interface names
CHSC 402
ctc 326
IUCV 340
LCS 320
mpc 326
overview 5
qeth 252, 270
storage-class memory 221
versus devices 6
vmcp 479
vmur 465
interfaces
CTC 326
FC-HBA 181
internal shared memory
device driver 357
invalid_crc_count zfcp attribute 191
invalid_tx_word_count zfcp attribute 191
IOCDs 19
iocounterbits
zfcp attribute 208
ioctl
CHSC 402
protected key device driver 524
iodone_cnt
zfcp attribute (SCSI device) 208
ioerr_cnt
zfcp attribute (SCSI device) 208
iorequest_cnt
zfcp attribute (SCSI device) 208
ip 4
IP address
confirming 272
duplicate 273
takeover 284
virtual 288
IP address takeover, activating and deactivating 285
ip-link
command 300
ipa_takeover, qeth attributes 284
IPL
displaying current settings 670
IPL configurations 75
IPL devices
for booting 80
preparing 59
IPv6
qeth support for 252
stateless autoconfiguration 252
ISM
device driver 357
ism, kernel module 357
ISO-8859-1 599
isolation, qeth attribute 275
IUCV
accessing terminal devices over 50
activating an interface 342
authorizations 346
buffer attribute 341
connection attribute 339
IBM compatible disk layout 139
IBM Doc Buddy 570
IBM Java 423
IBM label partitioning scheme 138
IBM TotalStorage Enterprise Storage Server 137
ica_api.h 511
icainfo, Linux command 575
icastats, Linux command 575
cmd
S/390 hypervisor file system attribute, z/VM guest 398
IDRC compression 233
IEEE 802.3 Ethernet 527
IEP 535
if_name
qeth attribute 270
IFCC 169
immediate failure of I/O requests 159
Improved Data Recording Capability compression 233
in_recovery
zfcp attribute (channel) 193
zfcp attribute (port) 197, 199
zfcp attribute (SCSI device) 208
in_recovery, zfcp attribute 190
in-kernel cryptography 527
inbound checksum
offload operation 274
inbound checksum, qeth 274
inbuf_size, qeth attribute 268
Initial Program Load, See IPL
initial RAM disk 81
inittab
module parameters 28
network 4
input/output configuration data set 19
instruction execution protection 535
Integrated ASCII console applet
on HMC 37
Inter-User Communication Vehicle 337
interface
MTIO 228
network 4
input/output configuration data set 19
instruction execution protection 535
Integrated ASCII console applet
on HMC 37
Inter-User Communication Vehicle 337
interface
MTIO 228
IBM compatible disk layout 139
IBM Doc Buddy 570
IBM Java 423
IBM label partitioning scheme 138
IBM TotalStorage Enterprise Storage Server 137
ica_api.h 511
icainfo, Linux command 575
icastats, Linux command 575
cmd
S/390 hypervisor file system attribute, z/VM guest 398
IDRC compression 233
IEEE 802.3 Ethernet 527
IEP 535
if_name
qeth attribute 270
IFCC 169
immediate failure of I/O requests 159
Improved Data Recording Capability compression 233
in_recovery
zfcp attribute (channel) 193
zfcp attribute (port) 197, 199
zfcp attribute (SCSI device) 208
in_recovery, zfcp attribute 190
in-kernel cryptography 527
inbound checksum
offload operation 274
inbound checksum, qeth 274
inbuf_size, qeth attribute 268
Initial Program Load, See IPL
initial RAM disk 81
inittab
module parameters 28
network 4
input/output configuration data set 19
instruction execution protection 535
Integrated ASCII console applet
on HMC 37
Inter-User Communication Vehicle 337
interface
MTIO 228
IBM compatible disk layout 139
IBM Doc Buddy 570
IBM Java 423
IBM label partitioning scheme 138
IBM TotalStorage Enterprise Storage Server 137
ica_api.h 511
icainfo, Linux command 575
icastats, Linux command 575
cmd
S/390 hypervisor file system attribute, z/VM guest 398
IDRC compression 233
IEEE 802.3 Ethernet 527
IEP 535
if_name
qeth attribute 270
IFCC 169
immediate failure of I/O requests 159
Improved Data Recording Capability compression 233
in_recovery
zfcp attribute (channel) 193
zfcp attribute (port) 197, 199
zfcp attribute (SCSI device) 208
in_recovery, zfcp attribute 190
in-kernel cryptography 527
inbound checksum
offload operation 274
inbound checksum, qeth 274
inbuf_size, qeth attribute 268
Initial Program Load, See IPL
initial RAM disk 81
inittab
module parameters 28
network 4
input/output configuration data set 19
instruction execution protection 535
Integrated ASCII console applet
on HMC 37
Inter-User Communication Vehicle 337
interface
MTIO 228
IUCV (continued)
devices 338
direct and routed connections 337
enablement 346
kernel configuration menu option 338, 345
maximum number of connections 347
MTU 341
OPTION MAXCONN 347
remove attribute 343
user attribute 340
z/VM enablement 339

iucv, kernel module 338, 346
iucvconn
set up a z/VM guest virtual machine for 43
using on z/VM 39
iucvtty 44
iucvtty, Linux command 43

J
Java
    GenWQE 417
        large page support 393
Java, GenWQE acceleration 423
journaling file systems
    write barrier 153

K
KB xi
KEK 727
kernel
    with message documentation support, building 569
kernel compression 81
kernel configuration menu options
    Call Home 559
    channel measurement facility 539
    chsc 401
    CLP 405
    CMM 489
    CONFIG_CRYPTO_FIPS 533
    CONFIG_PERF_EVENTS 543
    CONFIG_PKEY 523
    console 40
    CPACF, in-kernel cryptography 528
    CPACF, rnd 513
    CPACF, true random numbers 519
    CTCM 326
    DASD 146
    DCSS 469
    FIPS 533
    HMC media device driver 413
    hotplug memory 372
    IUCV 338, 345
    kernel compression 81
    large page support 391
    LCS 320
    messages 569
    monitor stream 439
    MONWRITER 447
    PCIe 427
    protected key 523
    qeth 258

kernel configuration menu options (continued)
    rnd 513
    RoCE 353
    S/390 hypervisor file system 395
    SMC 349
    special message support 484
    storage-class memory 222
    tape 229
    watchdog 110
    XPRAM 236
    z/VM *MONITOR record device driver 452
    z/VM CP interface 480
    z/VM recording 459
    z/VM unit record 465
    zcrypt 499
    zfcp 183
kernel cryptographic API 527
kernel image 81
kernel messages
    generating man pages 569
    mainframe-specific 570
kernel module
    aes_s390 530
    af_iucv 346, 347
    appldata_mem 439
    appldata_net_sum 439
    appldata_os 439
    chsc_sch 401
    cmm 489, 490
    ctcn 327
    dasd_diag_mod 146
    dasd_eckd_mod 146
    dasd_eer 146
    dasd_fba_mod 146
    dasd_mod 146
    dcssblk 468
    des_s390 530
    diag288_wdt 110
    eadm_sch 222
    genwqe 420
    ghash_s390 530
    hmcdrv 413
    hmcdrvfs 414
    ism 357
    iucv 338, 346
    lcs 320, 321
    monreader 452, 453
    MONWRITER 447
    netiucv 339
    paes_s390 530
    pkey 524
    prng 513
    qdio 258
    qeth 258, 259
    qeth_i2 258, 259
    qeth_i3 259
    sclp_async 559
    scm_block 222
    sha_256 530
    sha_512 530
    sha1_s390 530
    smc 350
    smsgiucv_app 484
    tape 229
kernel module (continued)
tape_34xx 229
tape_3590 229
vmlogrdr 459
vmur 465
xpram 237
zfcp 183

kernel panic
creating dump automatically after 567

kernel parameter
etr= 408

kernel parameter file
for z/VM reader 25

kernel parameter line
length limit for booting 25, 26
module parameters 28

kernel parameters
and zipl 64
apmask= 500
aqmask= 500
channel measurement facility 539
cio_ignore= 758
cmf.format= 539
cmf.maxchannels= 539
cmm.sender= 490
cmma= 762
conmode= 41
console= 42
DASD 147
dasd= 147
dcssblk.segments= 468
default_hugepagesz= 392
diag288_wdt.cmd= 110
diag288_wdt.conceal= 110
diag288_wdt.nowayout= 110
domain= 500
emu_nodes= 368
emu_size= 368
encoding 23
fips= 533
genra17e 757
hmcdv.cachesize= 414
hugepages= 392
hvc_iucv_allow= 43
hvc_iucv= 43
maxcpu= 763
no_console_suspend 101
noexec= 535
noinitrd 764
noresume 101
nosmt 765
numa_balancing= 368
numa_debug 368
numa= 368
possible_cpus= 766
prng.chunk_size= 513
prng.mode= 513
prng.reseed_limit= 513
ramdisk_size= 767
reboot 27
resume= 101
ro 768
root= 769
sched_debug 368

kernel parameters (continued)
sclp_con_drop= 42
sclp_con_pages= 43
smsgiucv_app.sender= 484
smtp= 770
specifying 23
stp= 409
topology= 365
vdso= 771
vmcp_cma= 480
vmhalt= 772
vppanic= 773
vmoff= 774
vmreboot= 775
xpram.parts= 237
zcrypt 500
zfcp.device= 183
zipl 24

kernel source tree ix
kernel-default-man 570
key encrypting key 727
kilo xi

LAN
sniffer 306
z/VM guest LAN sniffer 308
LAN channel station, See LCS
LAN, virtual 299
lancmd_timeout, LCS attribute 322
large page support
change number of 393
display information about 393
kernel configuration menu options 391
read current number of 393
large page support attribute
nr_hugepages 393
large pages
hotplug memory 392
last_known_reservation_state, DASD attribute 168
layer 2
qeth discipline 250
layer 3
qeth discipline 250
layer2
qeth attribute 265
layer2, qeth attribute 253
lcs
recover attribute 324

LCS
activating an interface 324
device driver 319
group attribute 321
interface names 320
kernel configuration menu option 320
lancmd_timeout attribute 322
online attribute 323
subchannels 319
ungroup attribute 322
LCS device driver
setup 321
lcs, kernel module 320
LDL disk layout 141

831
learning_timeout, qeth attribute 291
learning, qeth attribute 291
LGR 438
LIB_ZFCP_HBAAPI_LOG_FILE 219
LIB_ZFCP_HBAAPI_LOG_LEVEL 219
libcard, GenWQE 417
libfuse
  package 599, 747
libica 496
libzHW 417
lic_version, zfcp attribute 189
line edit characters, z/VM console 57
line-mode terminal
  control characters 53
  special characters 53
link_failure_count, zfcp attribute 191
Linux
  as LAN sniffer 306
Linux commands
  generic options 575
Linux device special file, See device nodes
Linux guest relocation 438
Linux in LPAR mode, booting 86
Linux on z/VM
  booting 81
  reducing memory of 437
lip_count, zfcp attribute 190
list media contents 415
listxattr 599, 747
LNX1 labeled disk 141
load balancing and VIPA 296
LOADDEV 84
LOADNNSHR operand
  DCSS 467
log file
  zfcp environment variable 219
log file, osasnmrp 316
log information
  FCP devices 194
log level
  zfcp environment variable 219
logging
  I/O subchannel status 551
logical boot device
  base parameters 65
  helper script 67
login at terminals 44
LOGREC, z/VM record 457
loss_of_signal_count, zfcp attribute 191
loss_of_sync_count, zfcp attribute 191
lost DASD reservation 167
LPAR
  configuration
    device pre-configuration 19
    storage-class memory 221
  DPM mode 19
  hardware counters 544
  I/O devices, configuring 565
LPAR configuration 221
LPAR Linux, booting 86
LPARs
  list using snipl 122
lrecl
  metadata file attribute 747
lschp, Linux command 654
lscpu, Linux command 361
lscpumf, Linux command 656
lsdss, Linux command 223, 659
Isdasp, Linux command 662
Ishm, Linux command 665
Islns, Linux command 666
lsqeth
  command 270
lsqeth, Linux command 669
lsreipl, Linux command 670
lsssc, Linux command 223, 671
Issshut, Linux command 673
Istape, Linux command 674
lszcrypt, Linux command 678
lszdev command 377
lszdev, Linux command 682
lszfcp, Linux command 687
LUNs
  finding available 217
LVM 224
lvm, partition type 633
M
MAC
  address learning 291
  MAC addresses 253
MAC header
  layer2 for qeth 253
  magic sysrequest 53
  magic sysrequest functions
    hvc0 53
    line-mode terminal 53
    VT220 terminal 53
major number
  CHSC 402
  CLP 405
  console devices 35
  DASD devices 142
  DCSS devices 467
  pseudo-random number 515
  SCSI 209
  tape devices 226
  vmcp 479
  vmur 465
  XPRAM 235
  z/VM *MONITOR record 454
  z/VM recording 457
  zcrypt with udev 501
  zfcp 209
man pages
  kernel messages, generating 569
man pages, messages 569
manage
  CPU capacity 720
management information base 311
max_KiB
  S/390 hypervisor file system attribute, z/VM guest 398
maxcpus=, kernel parameter 763
maxframe_size
  zfcp attribute 190
maximum frame size
  qeth 268
MB xi
mcast_flooding, qeth attribute 291
measurement
channel path 13
measurements
PCIe attribute 430
Media Access Control (MAC) addresses 253
Medium Access Control (MAC) header 254
medium_state, tape attribute 232
mega xi
mem_dela
S/390 hypervisor file system attribute, z/VM guest 398
memory
adding hotplug 374
block_size_bytes attribute 373
Flash Express 221
guest, reducing 437
hotplug 371
kernel configuration menu options for hotplug 372
state attribute 374
storage-class 221
memory control
complex rules 612
configuration file 609
memory hotplug
sample configuration file 614
memory hotplug rules 611
memory sysfs attribute
block_size_bytes 373
memory, expanded 235
menu configuration
z/VM example 83
messages
building a kernel with 569
mainframe-specific kernel 570
messages app 570
metadata file for z/OS DASD 747
MFS 268
mgmtime
S/390 hypervisor file system attribute 396
S/390 hypervisor file system attribute, LPAR 397
MIB (management information base) 311
min_KiB
S/390 hypervisor file system attribute, z/VM guest 398
minor number
CHSC 402
CLP 405
console devices 35
DASD devices 142
DCSS devices 473
pseudo-random number 515
SCSI 209
tape devices 225
vmcp 479
vmur 465
XPRAM 235
z/VM *MONITOR record 454
z/VM recording 457
z90crypt with udev 501
zfcp 209
mknod command 181
mknod, Linux command 3
mlx4
debugging 355
mlx4_en
device driver 353
mlx4, debug 355
modalias
common CCW attribute 9
cryptographic adapter attribute 502
mode
prandom attribute 516
mode terminal
full-screen 44
model
zfcp attribute (SCSI device) 208
module parameter
sender= 485
module parameters
allow Lun scan= 186
boot configuration 28
cachesize= 414
chunksize= 514
dasd= 147, 149
datarouter= 186
dbsize= 186
dcssblk.segments= 469
devs= 238
dif= 186
kernel parameter line 28
mode=
module parameters 514
mondcss= 453, 454
queue_depth= 186
reseed_limit= 514
scm_block= 222
sender= 484
sizes= 238
XPRAM 238
modules
qeth, removing 260
modulus-exponent 495
mon_fsstatd
command-line syntax 691
monitor data, processing 691
monitor data, reading 692
systemd service unit syntax 689
mon_fsstatd, command 689
mon_procd
command-line syntax 695
monitor data, reading 699
systemd service unit syntax 694
mon_procd, command 694
mon_statd
monitor data
read 436
monitor DCSS 453
monitor stream
kernel configuration menu options 439
module activation 440
on/off 440
sampling interval 441
monitor stream application
device driver 447
monitoring
  z/VM performance 435
  monitoring Linux instances 435
monreader, kernel module 452
MONWRITER
  kernel configuration menu options 447
  MONWRITER, kernel module 447
mount media contents 416
mount point
  debugfs x
  procfs x
  sysfs x
mt-st, package 233
MTIO interface 228
MTU
  IUCV 341
  qeth 271
multicast forwarding 291
multicast_router, value for qeth router attribute 281
multithreading 361

N
name
  devices, See device names
  network interface, See base name
names
  DASD 142
  native, partition type 633
  net-snmp 311
NETIUCV
  device driver 337
  netiucv, kernel module 339
network
  interface names 5
  network concentrator examples 303
Network Concentrator 301
network interfaces 4
network traffic analyzer
  HiperSockets 307
no_console_suspend, kernel parameters 101
no_prio_queueing 266
no_router, value for qeth router attribute 281
node_name
  zfcp attribute 190
  zfcp attribute (port) 197
node, device, See device nodes
noexec=
kernal parameter 535
noinitrd, kernel parameter 764
non-operational terminals
  preventing re-spawns for 47, 49
non-priority commands 55
non-rewinding tape device 225
noresume, kernel parameters 101
nos_count, zfcp attribute 190
nosmt, kernel parameter 765
nowayout= 112
nowayout=, module parameters 112
NPIV
  (continued)
  removing SCSI devices 214
nr_hugepages
  large page support attribute 393
NUMA emulation 367
numa_balancing=, kernel parameters 368
numa_debug, kernel parameters 368
numa=, kernel parameters 368
O
object ID 311
offline
  CHPID 14, 15
devices 9
offload operations
  inbound checksum 274
  outbound checksum 274
  TCP segmentation offload (TSO) 274
OID (object ID) 311
online
  CHPID 14, 15
  common CCW attribute 9
  CPU attribute 363
cryptographic adapter attribute 504
  CTCM attribute 330
  DASD attribute 157, 158
etr attribute 409
  LCS attribute 323
  qeth attribute 270
  stp attribute 409
tape attribute 230, 232
  TTY attribute 52
  zfcp attribute 188
onlinetime
  S/390 hypervisor file system attribute, LPAR 397
Open Source Development Network, Inc. 311
openCryptoki, library 511
Operating System Messages applet
  emulation of the HMC 55
operation, tape attribute 232
OPTION MAXCONN 347
optional properties
  DCSS 468
OSA-Express
  device driver 243
  LAN channel station 319
  SNMP subagent support 311
OSA-Express MIB file 312
osasnmpd
  access control 313
  checking the log file 316
  master agent 311
  setup 312
  starting the subagent 316
  stopping 318
  subagent 311
osasnmpd, command 701
osasnmpd, OSA-Express SNMP subagent 311
OSDN (Open Source Development Network, Inc.) 311
other
  S/390 hypervisor file system attribute, z/VM guest 398
  outbound checksum
  offload operation 274
outbound checksum, qeth 275
overlap with guest storage 453

P

paes_s390, kernel module 530
page pool
  static 437
  timed 437
parallel access volume (PAV) 172
parameter
  kernel and module 23
partition
  on DASD 138
  schemes for DASD 138
  table 140
  XPRAM 235
partitioning
  SCSI devices 181
partitions
  XPRAM 237
path_autodisable
  DASD attribute 169
path_interval
  DASD attribute 169
path_threshold
  DASD attribute 169
PAV (parallel access volume) 172
PAV enablement, suppression 147
pchid
  PCIe attribute 429
PCHID
  map to CHPID 16
PCIe
defective 429
device driver 427
  function_handle attribute 429
  function_id attribute 429
  kernel configuration menu options 427
  pchid attribute 429
  pfid attribute 429
  pfip attribute 429, 430
  power attribute 428
  recover attribute 429
  set up 428
  statistics attribute 430
  uid attribute 429
  util_string attribute 429
  vfn attribute 429, 430
peer_d_id, zfcp attribute 189
peer_wwnn, zfcp attribute 189
peer_wwpn, zfcp attribute 189
pendingq_count
  cryptographic adapter attribute 503, 504
perf tool
  building 544
  reading a hardware counter 545
  reading sample data 546
performance
  CPU-measurement facilities 543
  DASD 161, 620
  QDIO 553
performance measuring
  with hardware facilities 537
performance monitoring
  z/VM 435
performance statistics, QETH 277
Peripheral Component Interconnect 427
permanent_port_name, zfcp attribute 190, 193
permissions
  S/390 hypervisor file system 399
  persistent device configuration 377
pfid
  PCIe attribute 429
pfip
  PCIe attribute 429, 430
physical channel ID
  for CHPID 16
physical_s_id, zfcp attribute 193
pimpampom, subchannel attribute 13
pkey, kernel module 524
PNET ID 357
polarization
  CPU sysfs attribute 366
  values 366
poll thread
  enable using chcrypt 587
poll_thread
  AP bus 510
  cryptographic adapter attribute 505
poll_timeout
  cryptographic adapter attribute 506
  set using chcrypt 587
port scan
  controlling 195
port_id
  zfcp attribute (port) 197
port_id, zfcp attribute 190
port_name
  zfcp attribute (port) 197
port_name, zfcp attribute 190
port_remove, zfcp attribute 200
port_rescan, zfcp attribute 194
port_scan_backoff 195
port_scan_ratelimit 195
port_state
  zfcp attribute (port) 197
port_type, NPIV 201
port_type, zfcp attribute 190
portno, qeth attribute 269
ports
  listing 217
  possible_cpus=, kernel parameter 766
power attribute
  PCIe 428
power/state attribute 102
prandom
  byte_counter attribute 516
  chunksize attribute 516
  errorflag attribute 516
  for non-root users 515
  mode attribute 516
preferred console 42
preparing as dump device 67
preparing ECKD 152
preparing FBA 153
prerequisites 137
pri=, fstab parameter 102
prim_seq_protocol_err_count, zfcp attribute 191
primary_connector, value for qeth router attribute 281
primary_router, value for qeth router attribute 281
prio_queueing_prec 266
prio_queueing_skb 266
prio_queueing_tos (deprecated) 266
prio_queueing_vlan 266
prio_queueing, value for qeth priority_queueing attribute 267
priority command 55
priority_queueing, qeth attribute 266
prng
  reseed 517
  reseed_limit 517
prng, kernel module 513
prng.chunk_size=, kernel parameters 513
prng.mode=, kernel parameters 513
prng.reseed_limit=, kernel parameters 513
processors
  cryptographic 7
procs
  appldata 440
  cio_ignore 759
  magic sysrequest function 54
  VLAN 300
programming interfaces
  CHSC subchannel device driver 402
  CLP subchannel device driver 405
  protected key device driver 524
promiscuous mode 256
prot_capabilities
  zfcp attribute 216
protected key 523
protected key device driver
  programming interfaces 524
protected key support
  kernel configuration menu options 523
protocol, CTCM attribute 329
proxy ARP 287
proxy ARP attributes 262
pseudo-random number
  device names 515
  device nodes 515
pseudorandom number
  device driver 513
pseudorandom number device driver
  setup 513
PSW
  disabled wait 567
purge, z/VM recording attribute 461
PVMSG 55
Q
qclib 562
QDIO
  performance 553
qdio, kernel module 258
qeth
  activating an interface 271
  activating and deactivating IP addresses for takeover 285
  auto-detection 251
  bridge_hostnotify attribute 256
  bridge_invisible attribute 291
qeth (continued)
  bridge_role attribute 256, 289
  bridge_state attribute 256
  buffer_count attribute 267
  card_type attribute 269
  configuration tool 704
  deactivating an interface 273
  device driver 243
  discipline, switching 259
  displaying device overview 669
  enable attribute for IP takeover 284
  fake_broadcast attribute 283
  flooding attribute 291
  function summary 247
  group attribute 263
  group devices, names of 250
  hsuid attribute 288
  hw_trap attribute 279
  if_name attribute 270
  inbuf_size attribute 268
  ipa_takeover attributes 284
  isolation attribute 275
  kernel configuration menu options 258
  layer 2 250
  layer 3 250
  layer2 attribute 253, 265
  learning attribute 291
  learning_timeout attribute 291
  mcast_flooding attribute 291
  MTU 271
  online attribute 270
  portno attribute 269
  priority_queueing attribute 266
  problem determination attribute 261
  proxy ARP attributes 262
  recover attribute 273
  removing modules 260
  route4 attribute 280
  route6 attribute 280
  rx_bcast attribute 291
  sniffer attributes 262
  subchannels 251
  summary of attributes 260–263
  takeover_learning attribute 291
  takeover_setvmac attribute 291
  TCP segmentation offload 282
  ungroup attribute 264
  VIPA attributes 262
  vnicc attributes 291
qeth device driver
  building 258
  IPv6 252
qeth interfaces, mapping 6
QETH performance statistics 277
qeth_l2, kernel module 258
qeth_l3, kernel module 259
qeth, kernel module 258
qetharp, Linux command 702
qethconf, Linux command 704
qethqoat, Linux command 707
query host access
  DASD 171
query HPF
  DASD 170
queue_depth, zfcp attribute 210
queue_depth=, module parameters 186
queue_ramp_up_period, zfcp attribute 210
queueing, priority 266

R

raid, partition type 633
RAM disk, initial 81
ramdisk_size=, kernel parameter 767
random number
  device driver 513, 519
  device names 515
  device nodes 515
random number device driver
  kernel configuration menu options 513
random numbers
  reading 516, 520
raw_track_access, DASD attribute 165
raw-track access mode 165, 622, 747
RDMA 427
read monitor data 436
readelf, Linux command 535
readlink, Linux command 6
readonly
  DASD attribute 173
reboot
  kernel parameters 27
recfm
  metadata file attribute 747
record layout
  z/VM 458
recording, z/VM recording attribute 460
recover
  PCIe attribute 429
recover, lcs attribute 324
recover, qeth attribute 273
recovery, CTC interfaces 332
reflective relay mode 275
relative port number
  qeth 269
Remote Direct Memory Access (RDMA) 427
Remote Spooling Communications Subsystem 739
Removable media, loading Linux 92
remove channel path
  DASD 169
remove, DCSS attribute 475
remove, IUCV attribute 343
request processing
  cryptographic 497
request_count
  cryptographic adapter attribute 502
requestq_count
  cryptographic adapter attribute 503, 504
rescan
  zfcp attribute (SCSI device) 212
reseed
  prandom attribute 516
  prng 517
reseed_limit
  prandom attribute 516
  prng 517
reseed_limit=, module parameters 514
reservation state (continued)
  DASD 168
reservation_policy, DASD attribute 167
reset_statistics
  zfcp attribute 190
respawn prevention 47, 49
restrictions 137
resume 99
resume=, kernel parameters 101
retrieving hardware information 562
reuse 236
rev
  zfcp attribute (SCSI device) 208
rewinding tape device 225
RFC
  1950 (zlib) 417
  1951 (deflate) 417
  1952 (gzip) 417
  2131 (DHCP) 305
  2132 (DHCP options and BOOTP Vendor Extensions) 305
  zlib
    RFC 417
Rivest-Shamir-Adleman 495
ro, kernel parameter 768
RoCE
  kernel configuration menu options 353
roles
  zfcp attribute (port) 197
root=, kernel parameter 769
route4, qeth attribute 280
route6, qeth attribute 280
router
  IPv4 router settings 280
  IPv6 router settings 280
RPM
  kernel-default-man 570
  libfuse 599, 747
  libhugetlbfs 391
  libica 496
  mt-st 233
  openCryptoki 511
  s390-tools 575
  sg3_utils 674
  snipl 115, 720
  util-linux 361
RSA 495
RSA exponentiation 495
RSCS 739
rx_bcast, qeth attribute 291
rx_frames, zfcp attribute 190
rx_words, zfcp attribute 190
S
s_id, zfcp attribute 193
S/390 hypervisor file system
  defining access rights 399
  directory structure 395
kernel configuration menu options 395
LPAR directory structure 395
updating hypfs information 399
z/VM directory structure 397
s390_sthyi() 562

837
s390-tools, package 575
s390dbf 438
safe_offline
  DASD attribute 157
sample_count, cmf attribute 541
sampling facility
  reading data 546
save, DCSS attribute 474
sched_debug, kernel parameters 368
SCLP_ASYNC 559
SCLP_ASYNC device driver 559
sclp_async, kernel module 559
sc卜_con_drop=, kernel parameter 42
sc卜_con_pages=, kernel parameter 43
SCM 224
scm_block, kernel module 222
scm_block=, module parameters 222
script
  base device 67
SCSI
data consistency checking 216
device nodes, creating with mknod 181
multipath devices 179
SCSI device
  automatically attached, configuring 201
  configuring manually 201
SCSI devices
  information in sysfs 207
  partitioning 181
  removing 214
  sysfs structure 178
  udev-created nodes 179
SCSI tape
  lstape data 676
scsi_host_no, zfcp attribute 203
scsi_id, zfcp attribute 203
scsi_level
  zfcp attribute (SCSI device) 208
scsi_llogging_level, Linux command 710
scsi_lun, zfcp attribute 203
scsi_target_id
  zfcp attribute (port) 197
SCSI-over-Fibre Channel
  kernel configuration menu options 183
SCSI-over-Fibre Channel device driver 177
SCSI, booting from 84, 86
SE (Support Element) 80
secondary_connector, value for qeth router attribute 281
secondary_router, value for qeth router attribute 281
seconds_since_last_reset
  zfcp attribute 190
seglist, DCSS attribute 472
segmentation offload, TCP 282
send files
  vmur command 745
  send files to z/VSE
    vmur command 746
sender=, module parameter 484, 485
serial_number, zfcp attribute 190
service levels
  reporting to IBM Support 566
service utility
  cpuplugd 606
set, CPI attribute 557
setup
  LCS device driver 321
  source VIPA 296
  standard VIPA 294
setxattr 599
sg_iq, Linux command 674
sg3_utils, package 674
sha_256, kernel module 530
sha_512, kernel module 530
SHA-1 527
SHA-256 527
SHA-512
  in-kernel cryptography 527
  mode for PRNG 513
sha卜_s390, kernel module 530
shared, DCSS attribute 473
Shoot The Other Node In The Head 132
shutdown actions 105
SIE capability 563
SIGNAL SHUTDOWN, suspend 103
simple network IPL 115
Simple Network Management Protocol 311
  simultaneous multithreading 361
  sizes=, module parameter 238
SMC
  kernel configuration menu option 349
  smc_pnet, Linux command 713
  smc_run, Linux command 715
  smc, kernel module 350
  smcss, Linux command 716
  SMSG_ID 485
  SMSG_SENDER 485
  smsgiucv_app
    device driver 483
  smsgiucv_app, kernel module 484
  smsgiucv_app.sender=, kernel parameters 484
  SMT 361
  sm卜, kernel parameter 770
  snclp, Linux command 720
  sniffer
    attributes 262
    sniffer, guest LAN 308
  snipl
    list LPARs 122
    package 115, 720
  snipl, Linux command 115
  SNMP 311
  SNMP queries 317
  snmpcmd command 317
  source VIPA
    setup 296, 298
  special characters
    line-mode terminals 53
    z/VM console 57
  special file
    DASD 144
    See also device nodes
  special message support
    kernel configuration menu options 484
    speed, zfcp attribute 190
  ssch_rsch_count, cmf attribute 540
standard VIPA
838
standard VIPA (continued)
adapter outage 294
setup 294
standby CPU, configuring 362
state
sysfs attribute 374
zfcp attribute (SCSI device) 213
state attribute, power management 102
state, tape attribute 232
stateless autoconfiguration, IPv6 252
static page pool
reading the size of the 491
static page pool size
setting to avoid guest swapping 566
static routing, and VIPA 294
statistics
DASD 161, 620
PCIe attribute 430
QDIO 553
status
DASD attribute 174
status information
FCP devices 194
status, CHPID attribute 14, 15
STHYI instruction 562
STONITH 132
stonith, Linux command 132
storage
memory hotplug 371
storage-class memory
device driver 221
device names 221
device nodes 221
displaying overview 671
kernel configuration menu options 222
working with increments 223
Store Hypervisor Information instruction 562
stp
online attribute 409
STP
sysfs interface 409
stp=, kernel parameter 409
strength
prandom attribute 516
stripe size, NUMA emulation 367
subchannel
status logging 551
subchannels
attributes in sysfs 13
CCW and CCW group devices 7
CTCM 325
displaying overview 659
EADM 221
in sysfs 12
LCS 319
qeth 251
summary of changes vii
support
AF _IUCV address family 345
Support Element 80
supported_classes
zfcp attribute (port) 197
supported_classes, zfcp attribute 190
supported_speeds, zfcp attribute 190
suspend
on SIGNAL SHUTDOWN 103
swap partition
for suspend resume 101
priority 102
swap, partition type 633
swapping
avoiding 437
symbolic_name, zfcp attribute 190
SYMPOTM, z/VM record 457
syntax diagrams 801
syntax overview
zipl 60
sysfs
channel subsystem view 12
device view 12
device view by category 11
device view by drivers 11
FCP devices 178
information about SCSI devices 207
SCSI devices 178
sysfs attribute
block_size_bytes 373
cm_enable 13
state 374
sysinfo 561
sysplex_name, CPI attribute 556
sysrequest 53
system states
displaying current settings 673
system time 407
system time protocol 407
system_level, CPI attribute 556
system_name, CPI attribute 555
system_type, CPI attribute 556
systemd
suspend 103
T
T10 DIF 216
takeover_learning, qeth attribute 291
takeover_setvmac, qeth attribute 291
tape
access by bus ID 228
blocksize attribute 232
booting from 82, 86, 91
cmb_enable attribute 232
cutype attribute 232
device names 225
device nodes 227
devtype attribute 232
display support 731
displaying overview 674
encryption support 727
IDRC compression 233
kernel configuration menu options 229
loading and unloading 234
medium_state attribute 232
MTIO interface 228
online attribute 230, 232
operation attribute 232
state attribute 232
uid attribute 430
tape device driver 225
tape devices
  typical tasks 230
tape_34xx, kernel module 229
tape_3590, kernel module 229
tape, kernel module 229
tape390_crypt, Linux command 727
tape390_display, Linux command 731
tar command, acceleration 422
tar, command 422
TCP segmentation offload 282
TCP/IP
  ARP 255
  DHCP 306
  IUCV 337
  kernel configuration option 258
  point-to-point 325
  service machine 326, 343, 353, 357
TDEA, See TDES
TDES
  in-kernel cryptography 527
  mode for PRNG 513
TERM, environment variable 44
terminal
  3270, switching the views of 51
  accessing over IUCV 50
  CCW, switching device on- or offline 52
  enabling user logins with /etc/sysconfig/init 44
  enabling user logins with initab 45
  enabling user logins with Upstart 45
  line-mode 44
  mainframe versus Linux 34
  non-operational, preventing re-spawns for 47, 49
  provided by the 3270 terminal device driver 44
terminals
  escape character 56
tgid_bind_type, zfcp attribute 190
thread_siblings
  CPU sysfs attribute 364
time
  command 422
  cpuplugd keyword
  use with historical data 611
time-of-day clock 407
time, command 422
timed page pool
  reading the size of the 491
timed page pool size
  setting to avoid guest swapping 566
timeout
  DASD attribute 174
  DASD I/O requests 160
  zfcp attribute (SCSI device) 212
timeout for LCS LAN commands 322
timeout, DASD attribute 160
TOD clock 407
topology=, kernel parameter 365
total
  S/390 hypervisor file system attribute, z/VM guest 398
Triple Data Encryption Standard, See TDES
triple DES, See TDES
trng
  byte_counter attribute 520
TRNG device driver (continued)
  setup 519
troubleshooting 549, 565
true random numbers
  reading 520
  true random-number device driver
  setup 519
TTY
  console devices 35
  kernel configuration menu options 40
  online attribute 52
  routines 36
ttyrun
  systemd 49
tunedasd, Linux command 733
  tuning automatic port scans 195
tx_frames, zfcp attribute 190
tx_words, zfcp attribute 190
type
  cryptographic adapter attribute 502
  S/390 hypervisor file system attribute 396
  S/390 hypervisor file system attribute, LPAR 396
  zfcp attribute (SCSI device) 208
  type, CTCM attribute 329
U
udev
  DASD device nodes 144
  handling CP special messages 486
  SCSI device nodes 179
uevent 485
uid
  DASD attribute 174
  PCIe attribute 429, 430
ungroup
  CTCM attribute 328
  LCS attribute 322
  qeth attribute 264
unit_add, zfcp attribute 201
unit_remove, zfcp attribute 214
update
  S/390 hypervisor file system attribute 396
  S/390 hypervisor file system attribute, z/VM 397
  updating information
  S/390 hypervisor file system 399
Upstart
  user login to terminal 45
USB storage, HMC 413
USB-attached storage, loading Linux 92
use_diag
  DASD attribute 174
  zfcp attribute 155
used_KiB
  S/390 hypervisor file system attribute, z/VM guest 398
user
terminal login 46
user, IUCV attribute 340
user.dsorg
  extended attribute for z/OS data set 747
user.lrecl
  extended attribute for z/OS data set 747
user.recfm
  extended attribute for z/OS data set 747
using SCM devices with 224
V

VACM (View-Based Access Control Mechanism) 313
vdso=, kernel parameter 771
vendor
  DASD attribute 174
  zfcp attribute (SCSI device) 208
VEPA mode 275
versus guest storage 470
vfn
  PCIe attribute 429, 430
view
  channel subsystem 12
device 12
device by category 11
device by drivers 11
View-Based Access Control Mechanism (VACM) 313
VINPUT
  CP command 56
VIPA (virtual IP address)
  attributes 262
description 288, 293
diagnosis 290
example 294
  high-performance environments 296
  source 296
  static routing 294
  usage 294
VIPA, source
  setup 296
VIPA, standard
  adapter outage 294
  setup 294
virtual
  DASD 137
  IP address 288
  LAN 299
virtual dynamic shared object 771
Virtual Ethernet Port Aggregator mode 275
Virtual Flash Memory 221
VLAN
  configure 300
  introduction to 299
VLAN (virtual LAN) 299
VLAN example 301
vmconvert, Linux command 743
vmcp
  device driver 479
device names 479
device nodes 479
vmcp_cma=, kernel parameters 480
vmcp, Linux command 737
vmhalt=, kernel parameter 772
vmlogdr, kernel module 459
vmpanic=, kernel parameter 773
vmpoff=, kernel parameter 774
vmreboot=, kernel parameter 775
VMRM 438
VMSG 55
vmur
  device driver 465
device names 465
vmur (continued)
  device nodes 465
vmur command
  FTP 744
guest memory dump 744
log console transcript 744
read console transcript 744
send files 745
send files to z/VSE 746
z/VM reader as IPL device 745
vmur, kernel module 465
vmur, Linux command 739
VNIC characteristics 291
vnicc, qeth attributes 291
VOL1 labeled disk 139
VOLSER 139
VOLSER, DASD device access by 144
volume label 139
Volume Table Of Contents 140
VTOC 139, 140
W

watchdog
  device driver 109
device node 113
  kernel configuration menu options 110
  when adding DCSS 472
weight_cur
  S/390 hypervisor file system attribute, z/VM guest 397
weight_max
  S/390 hypervisor file system attribute, z/VM guest 397
write barrier 153
wwpn, zfcp attribute 193
X

x3270 code page 49
XFS 216
XPRAM
  device driver 235
device nodes 236
diagnosis 236
features 235
  kernel configuration menu options 236
  kernel parameter 237
  module parameter 238
  partitions 235
  setting up partitions 237
xpram, kernel module 237
xpram.parts=, kernel parameter 237
XRC, extended remote copy 407
XTS 527
Z

z/VM
  guest LAN sniffer 308
  monitor stream 439
  performance monitoring 435
z/VM *MONITOR record
  device name 454
device node 454
zfcp

access_denied attribute (FCP LUN) 208
access_denied attribute (port) 197
access_denied attribute (SCSI device) 208
access_shared attribute 208
card_version attribute 189
delete attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
error_frames attribute 190
failed attribute 214
device driver 177
device nodes 179
device_blocked attribute (SCSI device) 208
dumped_frames attribute 191
zfcp (continued)
fcp_output_requests attribute 191
hardware_version attribute 189
HBA API support 218
in_recovery attribute 190
in_recovery attribute (channel) 193
in_recovery attribute (port) 197, 199
in_recovery attribute (SCSI device) 208
invalid_crc_count attribute 191
invalid_tx_word_count attribute 191
iocounterbits attribute 208
iodone_cnt attribute (SCSI device) 208
ioerr_cnt attribute (SCSI device) 208
iorequest_cnt attribute (SCSI device) 208
kernel configuration menu options 183
kernel module 183
lic_version attribute 189
link_failure_count attribute 191
lip_count attribute 190
loss_of_signal_count attribute 191
loss_of_sync_count attribute 191
major/minor 209
maxframe_siz attribute 190
model attribute (SCSI device) 208
node_name attribute 190
node_name attribute (port) 197
nos_count attribute 190
online attribute 188
peer_d_id attribute 189
peer_wwwn attribute 189
peer_wwpnn attribute 189
permanent_port_name attribute 190, 193
physical_s_id attribute 193
port_id attribute 190
port_id attribute (port) 197
port_name attribute 190
port_name attribute (port) 197
port_remove attribute 200
port_rescan attribute 194
port_state attribute (port) 197
port_type attribute 190
prim_seq_protocol_err_count attribute 191
prot_capabilities attribute 216
queue_depth attribute 210
queue_ramp_up_period attribute 210
rescan attribute (SCSI device) 212
reset_statistics attribute 190
rev attribute (SCSI device) 208
roles attribute (port) 197
rx_frames attribute 190
rx_words attribute 190
s_id attribute 193
scsi_host_no attribute 203
scsi_id attribute 203
scsi_level attribute (SCSI device) 208
scsi_lun attribute 203
scsi_target_id attribute (port) 197
seconds_since_last_reset attribute 190
serial_number attribute 190
speed attribute 190
state attribute (SCSI device) 213
supported_classes attribute 190
supported_classes attribute (port) 197
supported_speeds attribute 190
zfcp (continued)
symbolic_name attribute 190
tgid_bind_type attribute 190
timeout attribute (SCSI device) 212
tx_frames attribute 190
tx_words attribute 190
type attribute (SCSI device) 208
unit_add attribute 201
unit_remove attribute 214
vendor attribute (SCSI device) 208
wwpn attribute 193
zfcp_access_denied attribute (SCSI device) 208
zfcp_failed attribute (SCSI device) 211
zfcp_in_recovery attribute (SCSI device) 208, 211
zfcp HBA API 181
zfcp traces 182
zfcp_access_denied
  zfcp attribute (SCSI device) 208
zfcp_failed
  zfcp attribute (SCSI device) 211
zfcp_in_recovery
  zfcp attribute (SCSI device) 208, 211
zfcp_ping 219
zfcp_show 219
zfcp.allow_lun_scan= 183
zfcp.allow_lun_scan=, kernel parameters 183
zfcp.datarouter= 183
zfcp.datarouter=, kernel parameters 183
zfcp.dbfsize= 183
zfcp.dbfsize=, kernel parameters 183
zfcp.device=, kernel parameter 183
zfcp.dif= 183
zfcp.dif=, kernel parameters 183
zfcp.queue_depth= 183
zfcp.queue_depth=, kernel parameters 183
zipl
  and kernel parameters 64
  base functions 59
  base parameters 65
  configuration file 74
  Linux command 59
  menu configurations 76
  modes 60
  parameters 71
  syntax overview 60
zipl boot menu 35
ZIPLCONF, environment variable 74
zlib
  GenWQE 417
  ZLIB_CARD, environment variable 421
  ZLIB_DEFLATE_IMPL, environment variable 421
  ZLIB_INFLATE_IMPL, environment variable 421
  ZLIB_TRACE, environment variable 421
  znetconf, Linux command 752
  zpcictrl, Linux command 755